NEST 08

Post -Recibir y validar la data-

- El comando de generar el módulo ya creó el dto (data transfer object) de create-pokemon. Ahora hay que implementar la lógica
- En el dto debe de haber el número de pokemon y el name
- Para la validación de la petición post necesito que el numero de pokemon sea un número, que sea mayor de 1 y que siempre se reciba el name como string
- Primero: hay que instalar el class-transformer y el class-validator
- Hay que hacer la validación global en el main

```
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
    })
  )

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

- Con esto tengo las validaciones globales
- Ahora coloco un par de decoradores en el dto

```
import { IsInt, IsPositive, IsString, Min, MinLength} from 'class-validator';

export class CreatePokemonDto {

    @IsInt()
    @IsPositive()
    @Min(1)
    no: number;

    @IsString()
    @MinLength(1)
```

```
    name: string;
}
```

- Recuerda: el endpoint es

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

- Porque así lo puse en el main con app.setGlobalPrefix('api/v2'), y 'pokemon' en el controller
- Ahora voy al servicio, y en create regreso el CreatePokemonDto.
- Podría hacerlo en el controlador pero es mejor no ensuciarlo
- Faltan validaciones, por ejemplo que el mismo pokemon no se vuelva a crear
- Quiero grabar el nombre en minúscula. En el servicio, en el método create:

```
create(createPokemonDto: CreatePokemonDto) {
    createPokemonDto.name = createPokemonDto.name.toLocaleLowerCase()
    return createPokemonDto;
  }
```

Crear Pokemon en base de datos

- Al tener el modelo listo y conectado, el insertar un usuario es algo bastante sencillo
- En el constructor del servicio voy a hacer inyección de dependencias (sólo en el constructor se hacen)
- pokemonModel va aser de tipo Model (importarlo de mongoose) y el generico Pokemon haciendo referencia a la entity
- Este modelo por si solo no es un provider. Para ello le añado el decorador de @InjectModel de @nest/mongoose

```
import { Injectable } from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
import { Model } from 'mongoose';
import { CreatePokemonDto } from './dto/create-pokemon.dto';
import { UpdatePokemonDto } from './dto/update-pokemon.dto';
import {Pokemon} from './entities/pokemon.entity'

@Injectable()
export class PokemonService {

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


  create(createPokemonDto: CreatePokemonDto) {
    createPokemonDto.name = createPokemonDto.name.toLocaleLowerCase()
    return createPokemonDto;
  }
```

```
  findAll() {
    return `This action returns all pokemon`;
  }

  findOne(id: number) {
    return `This action returns a #${id} pokemon`;
  }

  update(id: number, updatePokemonDto: UpdatePokemonDto) {
    return `This action updates a #${id} pokemon`;
  }

  remove(id: number) {
    return `This action removes a #${id} pokemon`;
  }
}
```

- Pero hace falta el nombre de ese modelo

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

- Ahora puedo inyectar el modelo en este servicio
- Dejando a un lado las validaciones y todo, cómo creo algo facilmente usando este modelo?
- Cómo las inserciones a la base de datos son asincronas coloco el async await

```
  async create(createPokemonDto: CreatePokemonDto) {
    createPokemonDto.name = createPokemonDto.name.toLocaleLowerCase()

    const pokemon= await this.pokemonModel.create(createPokemonDto)

    return pokemon;
  }
```

- Hay que manejar el error cuando se repite nombre, ya que lanza un server error 500 y no es el caso

Responder a un error específico

- Cuanto menos consultas a la base de datos, mejor.
- Cuando se repite un elemento lanza el error 11000
- Para atrapar el error meto la consulta en un try y un catch

```
  async create(createPokemonDto: CreatePokemonDto) {
    createPokemonDto.name = createPokemonDto.name.toLocaleLowerCase()

   try {
     const pokemon= await this.pokemonModel.create(createPokemonDto)
    return pokemon;

  } catch (error) {
    console.log(error)
   }

}
```

- Ahora, si repito el nombre en el body me devuelve un 201 vacío, pero esto es porque estoy manjenado la excepción
- Si miro en la terminal tengo el error de objeto duplicado 11000
- Puedo usar este código para no hacer otra consulta
- Si no es este error ya puedo lanzar un error del server
- El throw detiene la ejecución como el return

```
  async create(createPokemonDto: CreatePokemonDto) {
    createPokemonDto.name = createPokemonDto.name.toLocaleLowerCase()

   try {
     const pokemon= await this.pokemonModel.create(createPokemonDto)
    return pokemon;

  } catch (error) {
    if(error.code === 11000){
      throw new BadRequestException(`Pokemon exists in DB
${JSON.stringify(error.keyValue)}`)
    }
    console.log(error)
    throw new InternalServerErrorException("Can't create Pokemon. Check server
logs")
   }
  }
```

- Si quiero cambiar el codigo de error y otros, puedo usar el decorador @HttpCode en el controller

```
  @Post()
  @HttpCode(200)
  create(@Body() createPokemonDto: CreatePokemonDto) {
    return this.pokemonService.create(createPokemonDto);
  }
```

- Puedo usar @HttpCode(HttpStatus) (importo el HttpStatus también) y ahi tengo todas las opciones, OK, UNAUTHORIZED
- Apretando ctrl y pinchando sobre HttpStatus me aparece la lista completa

```
  @Post()
  @HttpCode(HttpStatus.OK)
  create(@Body() createPokemonDto: CreatePokemonDto) {
    return this.pokemonService.create(createPokemonDto);
  }
```

findOneBy - Buscar por nombre, Mongold y no

- Cambio el id a string en el controlador (el @Get(':id')) y en el servicio

```
 @Get(':id')
  findOne(@Param('id') id: string) {
    return this.pokemonService.findOne(id);
  }
```

- Me sitúo en el pokemon.service, en el findOne. Hay tres validaciones que debo hacer
- Defino el pokemon de tipo Pokemon (entity)
- Primero voy a verificar si el termino de busqueda es un numero

```
  async findOne(term: string) {

    let pokemon: Pokemon

    if( !isNaN(+term)){
      pokemon = await this.pokemonModel.findOne({ no: term })
    }

    return pokemon
  }
```

- Cambio id por term tambien en el controller

```
 @Get(':term')
  findOne(@Param('term') term: string) {
    return this.pokemonService.findOne(term);
  }
```

- Si el pokemon no existe mandaré un notFoundException

```
if(!pokemon) throw new NotFoundException(`Pokemon with id ${term} not exists`)
```

- Para el MongoID hay que evaluarlo con isValidObjectId de mongoose

```
 if(isValidObjectId(term)){
      pokemon= await this.pokemonModel.findById(term)
    }
```

- Si a estas alturas no ha encontrado ningún pokemon, voy a tratar de encontrarlo por el nombre

```
 if( !pokemon){
      pokemon = await this.pokemonModel.findOne({name: term.toLowerCase()})
    }
```

- El código queda así

```
  async findOne(term: string) {

    let pokemon: Pokemon
    if( !isNaN(+term)){
      pokemon = await this.pokemonModel.findOne({ no: term })
    }


    if(isValidObjectId(term)){
      pokemon= await this.pokemonModel.findById(term)
    }

    if( !pokemon){
      pokemon = await this.pokemonModel.findOne({name: term.toLowerCase().trim()})
    }

   if(!pokemon) throw new NotFoundException(`Pokemon with id ${term} not exists`)

    return pokemon

  }
```

- Para que no evalue si es un objectId le agrego una condición de si no tengo pokemon entonces evalúe el id

```


    if(!pokemon && isValidObjectId(term)){
```

```
      pokemon= await this.pokemonModel.findById(term)
    }
```

Actualizar Pokemon en DB

- En la entity puedo ver que el name está indexado y el no también, entonces es igual de rápido buscarlo por cualquiera de los dos
- Para manejarlo igual que con el @Get anteriormente, cambiaré el id por la palabra term (en el @Patch)
- Hay varias validaciones. Primero yo no puedo actualizar un pokemon si no existe
- Uso el método creado anteriormente

```
async update(term: string, updatePokemonDto: UpdatePokemonDto) {

  const pokemon= await this.findOne(term)
}
```

- Ahora en pokemon tengo todos los metodos que un modelo de mongoose nos ofrece
- Para asegurarme que el nombre venga en minúsculas evalúo si viene el nombre
- Guardo con el updateOne
- Para retornar el valor actualizado, yo se que el dto tiene la versión actualizada.
    - Esparzo todas las propiedades del viejo pokemon y sobreescribo con el dto

```
async update(term: string, updatePokemonDto: UpdatePokemonDto) {

  const pokemon= await this.findOne(term)

  if(updatePokemonDto.name)
    updatePokemonDto.name = updatePokemonDto.name.toLowerCase()

  await pokemon.updateOne(updatePokemonDto)

  return { ...pokemon.toJSON(), ...updatePokemonDto}

}
```

- El problema es que si coloco un id que ya existe en el body para hacer la actualización me devuelve un error 11000 de llave duplicada

Validar valores únicos

- Debo enviar un error que diga que ya existe ese id
- Meto la linea del await y el return en el try y en el catch copio y pego la validación anterior con 11000

```
 async update(term: string, updatePokemonDto: UpdatePokemonDto) {

    const pokemon= await this.findOne(term)

    if(updatePokemonDto.name)
      updatePokemonDto.name = updatePokemonDto.name.toLowerCase()

      try {
        await pokemon.updateOne(updatePokemonDto)

        return { ...pokemon.toJSON(), ...updatePokemonDto}

      } catch (error) {
        if(error.code === 11000){
          throw new BadRequestException(`Pokemon exists in DB
${JSON.stringify(error.keyValue)}`)
        }
        console.log(error)
        throw new InternalServerErrorException("Can't create Pokemon. Check server
logs")

      }

  }
```

- Como parece que es un código reutilizable bien podría crearse un método

```
  private handleException(error: any){
    if(error.code === 11000){
      throw new BadRequestException(`Pokemon exists in DB
${JSON.stringify(error.keyValue)}`)
    }
    console.log(error)
    throw new InternalServerErrorException("Can't create Pokemon. Check server
logs")
   }
```

- Utilizo el método

```
import { BadRequestException, Injectable, InternalServerErrorException,
NotFoundException } from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
import { isValidObjectId, Model } from 'mongoose';
import { CreatePokemonDto } from './dto/create-pokemon.dto';
import { UpdatePokemonDto } from './dto/update-pokemon.dto';
import {Pokemon} from './entities/pokemon.entity'

@Injectable()
```

```
export class PokemonService {

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


  async create(createPokemonDto: CreatePokemonDto) {
    createPokemonDto.name = createPokemonDto.name.toLocaleLowerCase()

   try {
     const pokemon= await this.pokemonModel.create(createPokemonDto)
    return pokemon;

  } catch (error) {
      this.handleException(error)
   }
  }

  findAll() {
    return `This action returns all pokemon`;
  }

  async findOne(term: string) {

    let pokemon: Pokemon
    if( !isNaN(+term)){
      pokemon = await this.pokemonModel.findOne({ no: term })
    }


    if(!pokemon && isValidObjectId(term)){
      pokemon= await this.pokemonModel.findById(term)
    }

    if( !pokemon){
      pokemon = await this.pokemonModel.findOne({name: term.toLowerCase().trim()})
    }
    if(!pokemon) throw new NotFoundException(`Pokemon with id ${term} not exists`)

    return pokemon

  }

  async update(term: string, updatePokemonDto: UpdatePokemonDto) {

    const pokemon= await this.findOne(term)

    if(updatePokemonDto.name)
      updatePokemonDto.name = updatePokemonDto.name.toLowerCase()

      try {
        await pokemon.updateOne(updatePokemonDto)
```

```
        return { ...pokemon.toJSON(), ...updatePokemonDto}

      } catch (error) {
        this.handleException(error)
      }


  }

  remove(id: number) {
    return `This action removes a #${id} pokemon`;
  }

  private handleException(error: any){
    if(error.code === 11000){
      throw new BadRequestException(`Pokemon exists in DB
${JSON.stringify(error.keyValue)}`)
    }
    console.log(error)
    throw new InternalServerErrorException("Can't create Pokemon. Check server
logs")
   }
}
```

- Si el findOne lanzara una excepción habría que controlarlo
- las excepciones del handleException serían excepciones no-controladas

Eliminar un Pokemon

- Cambio a string el id en el .service y .controller
- Debo validar que el id exista. Uso el metodo findOne que creé
- Si no hay un pokemon esto ya devuelve el error
- Pero si existe puedo colocar el await pokemon.deleteOne()

```
  async remove(id: string) {

    const pokemon = await this.findOne(id)
    await pokemon.deleteOne()
  }
```

- Para el custom pipe que voy a hacer, el delete tiene que ser el id de la base de datos
- No es por ninguna razón en específico, es para introducir la materia de los customPipes

CustomPipes -ParseMongoIdPipe

- Los pipes transforman físicamente la data

- En este caso un mongoID siempre sigue siendo una string, solo que ahora pasará por una validación previa
- Creo la carpeta fuera de la carpeta pokemon, ya que es una validación de ID de Mongo y vale para cualquiera
- Creo con el cli un modulo llamado common

> nest g mo common

- Esto me crea la carpeta y el módulo
- Para trabajar solo con los pipes no hace falta crear un módulo, pero puede que haya algun servicio o provider que caiga en los commons
- Ahora creo el pipe con el CLI

> nest g pi common/pipes/parseMongoId --no-spec

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

@Injectable()
export class ParseMongoIdPipe implements PipeTransform {
  transform(value: any, metadata: ArgumentMetadata) {
    return value;
  }
}
```

- Todos los pipes tienen que implementar el PipeTransform (que tiene que satisfacer su interface)
- le pongo un console.log({value, metadata})

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

@Injectable()
export class ParseMongoIdPipe implements PipeTransform {
  transform(value: any, metadata: ArgumentMetadata) {
    console.log({value, metadata})
    return value;
  }
}
```

- Lo inyecto en el controlador

```
 @Delete(':id')
  remove(@Param('id', ParseMongoIdPipe) id: string) {
    return this.pokemonService.remove(id);
  }
```

- El console.log() del pipe, si yo le añado al endpoint de delete el id bulbasur (en ThunderClient o POSTMAN)

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

```
{
  value: 'bulbasur',
  metadata: { metatype: [Function: String], type: 'param', data: 'id' }
}
```

- Si ahora uso en el return del pipe value.toUpperCasse() me devuelve BULBASUR

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

@Injectable()
export class ParseMongoIdPipe implements PipeTransform {
  transform(value: any, metadata: ArgumentMetadata) {
    console.log({value, metadata})
    return value.toUpperCase();
  }
}
```

- Hago uso del metodo de mongoose isValidObjectId
- Si es válido regresa el valor

```
import { ArgumentMetadata, BadRequestException, Injectable, PipeTransform } from
'@nestjs/common';
import { isValidObjectId } from 'mongoose';

@Injectable()
export class ParseMongoIdPipe implements PipeTransform {
  transform(value: any, metadata: ArgumentMetadata) {

    if(!isValidObjectId(value)){
      throw new BadRequestException(`${value} is not a valid id`)
    }

    return value;
  }
}
```

- Colocamos el método en el delete

```
async remove(id: string) {

  await  this.pokemonModel.findByIdAndDelete(id)
  }
```

- Hecho así, si mando un id que parece de mongo pero no existe, me manda un 200. Tengo que validarlo
- Pero quiero evitar hacer dos consultas y hacerlo todo en una misma linea

Validar y eliminar en una sola consulta

- Hay que validar el id con la base de datos

```
  async remove(id: string) {

    const result = await this.pokemonModel.deleteOne({_id: id})
    return result
  }
```

- Si el id no existe el return me devuelve esto:

```
{
  "acknowledged": true,
  "deletedCount": 0
}
```

- Puedo desestructurar estos dos valores del result

```
  async remove(id: string) {

    const {deletedCount} = await this.pokemonModel.deleteOne({_id: id})

    if(deletedCount===0)
      throw new BadRequestException(`Pokemon with id ${id} not found`)

    return
  }
```