NEST HERRERA - POKEDEX

Contenido estático

- Instalación para servir contenido estático

> npm i @nestjs/serve-static

- Creo carpeta public en la raíz con un index.html
- Configuración en app.module.ts

```
import { Module } from '@nestjs/common';
import { ServeStaticModule } from '@nestjs/serve-static';
import { join } from 'path';


@Module({
  imports: [ServeStaticModule.forRoot({
    rootPath: join(__dirname, '..', 'public')
  })],
  controllers: [],
  providers: [],
})
export class AppModule {}
```

- Creo la API de pokemon

> nest g res pokemon

Global prefix

- Global prefix en el main

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

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

  app.setGlobalPrefix('api/v2')

  await app.listen(process.env.PORT ?? 3000);
}
bootstrap();
```

Docker MongoDB

- Creo docker-compose.yaml en la raíz

```
version: '3'

services:
  db:
    image: mongo:5
    restart: always
    ports:
      - 27017:27017
    environment:
      - MONGODB_DATABASE=nest-pokemon
    volumes:
      - ./mongo:/data/db
```

> docker-compose up -d

- El string de conexión es **mongodb://localhost:27017/nest-pokemon**
- Puedo probarlo en TablePlus

Conexión con Mongo

- Para conectar Nest con Mongo instalo

> npm i @nestjs/mongoose mongoose

- En app.module

```
import { Module } from '@nestjs/common';
import { ServeStaticModule } from '@nestjs/serve-static';
import { join } from 'path';
import { PokemonModule } from './pokemon/pokemon.module';
import { MongooseModule } from '@nestjs/mongoose';


@Module({
  imports: [ServeStaticModule.forRoot({
    rootPath: join(__dirname, '..', 'public')
  }),
  MongooseModule.forRoot('mongodb://localhost:27017/nest-pokemon'),
  PokemonModule],
  controllers: [],
  providers: [],
})
export class AppModule {}
```

Entity

- Creo la entidad

- pokemon/entities/pokemon.entity.ts

```
import { Prop, Schema, SchemaFactory } from "@nestjs/mongoose";
import { Document } from "mongoose";

@Schema()
export class Pokemon extends Document {

    @Prop({
        unique: true,
        index: true
    })
    name: string

    @Prop({
        unique: true,
        index: true
    })
    no: number
}

export const PokemonSchema = SchemaFactory.createForClass(Pokemon)
```

- Conecto la entidad con la DB
- pokemon.module.ts

```
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],
})
export class PokemonModule {}
```

POST- Recibir y validar data

- Para el dto instalo

> npm i class-validator class-transformer

- El dto create-pokemon

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

export class CreatePokemonDto {

    @IsString()
    @MinLength(3)
    name: string

    @IsNumber()
    @IsPositive()
    @Min(1)
    no: number
}
```

- Para que se hagan las validaciones hago la configuración en el main

```
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
    })
  )

  await app.listen(process.env.PORT ?? 3000);
}
bootstrap();
```

Inyectar Modelo en el servicio

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

```
@Injectable()
export class PokemonService {


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


  async create(createPokemonDto: CreatePokemonDto) {
    createPokemonDto.name = createPokemonDto.name.toLowerCase()
    const pokemon = await this.pokemonModel.create(createPokemonDto)
    return pokemon
  }
}
```

- Con ThunderClient, método POST, en la url http://localhost:3000/api/v2/pokemon, en el body agrego

```
{
  "name": "Bulbasur",
  "no": 1
}
```

- Me retorna

```
{
  "name": "bulbasur",
  "no": 1,
  "_id": "683420eb0a4e5b0c3dfd363e",
  "__v": 0
}
```

- En el método create puedo usar un try catch
- pokemon.service.ts

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

    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")
    }
  }
```

- Puedo alterar el valor de respuesta con el decorador @HttpCode en el controller y usar HttpStatus (o el número directamente, 200)

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

findOne

- Hay 3 identificadores: el nombre, el número y el id de mongo
- Hay que validar que sea un id de mongo válido con isValidObjectId de mongoose
- Uso el id como string para hacer las validaciones y lo parseo a numero si es necesario
- En el controller

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

- En el servicio

```
 async findOne(id: string) {
    let pokemon: Pokemon | null = null

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

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

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

    if(!pokemon) throw new NotFoundException("Pokemon not found")
```

```
      return pokemon
  }
```

- De esta manera puedo buscar por número, id o nombre

Actualizar Pokemon

- Si intento actualizar el número de un pokemon que ya existe con otro nombre me devuelve error 11000 (de valor duplicado)
- pokemon.service.ts

```
  async update(id: string, updatePokemonDto: UpdatePokemonDto) {
    let pokemon = await this.findOne(id)

    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) //para debuggear

        throw new InternalServerErrorException("Can't create pokemon - Check
server logs")
    }
  }
```

Método del servicio para manejar errores

- pokemon.service.ts

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

- Lo uso en el catch
- pokemon.service.ts

```
async update(id: string, updatePokemonDto: UpdatePokemonDto) {
    let pokemon = await this.findOne(id)

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

    try {
      await pokemon.updateOne(updatePokemonDto)
      return {...pokemon.toJSON(), ...updatePokemonDto}
    } catch (error) {
        this.handleExceptions(error)
    }
  }
```

Eliminar un Pokemon

- pokemon.service.ts

```
async remove(id: string) {
const pokemon = await this.findOne(id)
await pokemon.deleteOne()
}
```

- Quiero implementar la lógica para que solo se pueda borrar con el id de mongo

Implementar la lógica para tener que usar un id de Mongo para borrar un Pokemon

- Creo el módulo de common, uso el CLI

```
nest g mo common
```

- Uso el CLI para crear el custom Pipe de Mongo. El CLI me añade Pipe al final

```
nest g pi common/pipes/parseMongoId
```

- Esto me crea el esqueleto del pipe
- Coloco el pipe en el controlador @Delete
- pokemon.controller.ts

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

- Hago un console.log del value y la metadata en el Pipe
- parse-mongo-id-pipe

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

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

- Como id en la url le paso un 1 me devuelve esto (habiendo comentado el código del delete en el servicio)

```
{
  value: '1',
  metadata: { metatype: [Function: String], type: 'param', data: 'id' }
}
```

- Uso el isValidObjectId para hacer la validación
- parse-mongo-id.pipe

```
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 MongoID`)
    }

    return value
  }
}
```

- pokemon.service.ts

```
async remove(id: string) {
    const pokemon = await this.pokemonModel.findByIdAndDelete(id)
    return pokemon
  }
```

- Hecho así, si envío un ID válido pero no lo encuentra me devolverá un 200 igual
- Uso deletedCount en la desestructuración de deleteOne para validar

```
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
}
```

SEED

- Creo la API del SEED con los entry points

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

- Instalo axios

```
npm i axios
```

- Me traigo 500 pokemons de pokeapi que insertaré en la DB

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

- 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;
  }
}
```

- El controller

```
import { Controller, Get, Post, Body, Patch, Param, Delete } from
'@nestjs/common';
import { SeedService } from './seed.service';
import { CreateSeedDto } from './dto/create-seed.dto';
import { UpdateSeedDto } from './dto/update-seed.dto';

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

 @Get()
 executeSEED(){
  return this.seedService.executeSEED()
 }
}
```

- Saco la interfaz del resultado con PasteJSONAsCode
- seed/interfaces/poke-response.interface.ts

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

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

- Si añado el tipo a la petición get de Axios tengo el tipado
- El número de id del pokemon está en la penúltima posición de la url
- seed.service.ts

```
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");
```

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

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

    return data;
  }
}
```

Insertar Pokemons

- Inyecto el modelo en el SeedService
- Para ello debo exportarlo del 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]
})
export class PokemonModule {}
```

- Debo importar el módulo de Pokemon en seed.module.ts

```
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 {}
```

- Inyecto el módulo en el SeedService e inserto los pokemons en la DB

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

@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=10");

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

    return 'Seed executed';
  }
}
```

- Si tuviera que hacer miles de inserciones demoraría mucho tiempo

Insertar múltiples registros simultáneamente

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

@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=10");

    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';
  }
}
```

Crear un Custom provider (Patrón Adaptador)

- Creo la carpeta common/**interfaces**/http-adapter.interface.ts

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

- Creo la carpeta common/**adapters**/axios.adapter.ts

```
import axios, { AxiosInstance } from "axios"
import { HttpAdapter } from "../interfaces/http-adapter.interface"


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**
- common.module.ts

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

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

- Importo el common.module en el módulo de seed

```
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 {}
```

- Inyecto el AxiosAdapter en el seed.service.ts

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

@Injectable()
export class SeedService {
```

```
  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=10");

    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';
  }
}
```

Paginación

- Para obtener 5 pokemons lo haría así

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

- Obtengo los query parameters con el decorador **@Query** en el pokemon.controller

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

- common/dtos/pagination.dto

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

export class PaginationDto{

    @IsNumber()
    @IsPositive()
    @IsOptional()
    @Min(1)
    limit?: number


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

- Para transformar a número el string del query uso transform en el ValidationPipe del main

```
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,
      transform: true,
      transformOptions:{
        enableImplicitConversion: true
      }
    })
  )

  await app.listen(process.env.PORT ?? 3000);
}
bootstrap();
```

- Desestructuro del dto en el servicio

```
async findAll(paginationDto: PaginationDto) {
    const {limit=10, offset=0}= paginationDto;

    return await this.pokemonModel.find()
      .limit(limit)
```

```
      .skip(offset)
      .sort({
        no:1 //ordeno la columna no de manera ascendente
      })
  }
```

Variables de entorno

- Añado ConfigModule en app.module
- Instalo

```
npm i @nestjs/config
```

```
import { Module } from '@nestjs/common';
import { ServeStaticModule } from '@nestjs/serve-static';
import { join } from 'path';
import { PokemonModule } from './pokemon/pokemon.module';
import { MongooseModule } from '@nestjs/mongoose';
import { CommonModule } from './common/common.module';
import { SeedModule } from './seed/seed.module';
import { ConfigModule } from '@nestjs/config';


@Module({
  imports: [
    ConfigModule.forRoot(),
    ServeStaticModule.forRoot({
    rootPath: join(__dirname, '..', 'public')
  }),
  MongooseModule.forRoot('mongodb://localhost:27017/nest-pokemon'),
  PokemonModule,
  CommonModule,
  SeedModule],
  controllers: [],
  providers: [],
})
export class AppModule {}
```

- Para usar las variables solo tengo que escribir process.env.VARIABLE
- Creo app.config.ts

```
export const EnvConfiguration=()=>({
    environment: process.env.NODE_ENV || 'dev',
    mongodb: process.env.MONGODB,
    port: process.env.PORT || 3001,
    defaultLimit: process.env.DEFAULT_LIMIT || 5
})
```

.env

```
NODE_ENV=dev
MONGODB=mongodb://localhost:27017/nest-pokemon
PORT=3000
DEFAULT_LIMIT=5
```

- Le digo a ConfigModule que lo cargue

```
import { Module } from '@nestjs/common';
import { ServeStaticModule } from '@nestjs/serve-static';
import { join } from 'path';
import { PokemonModule } from './pokemon/pokemon.module';
import { MongooseModule } from '@nestjs/mongoose';
import { CommonModule } from './common/common.module';
import { SeedModule } from './seed/seed.module';
import { ConfigModule } from '@nestjs/config';
import { EnvConfiguration } from './app.config';


@Module({
  imports: [
    ConfigModule.forRoot({
      load: [EnvConfiguration]
    }),
    ServeStaticModule.forRoot({
    rootPath: join(__dirname, '..', 'public')
  }),
  MongooseModule.forRoot('mongodb://localhost:27017/nest-pokemon'),
  PokemonModule,
  CommonModule,
  SeedModule],
  controllers: [],
  providers: [],
})
export class AppModule {}
```

- Inyecto el ConfigService de @nestjs/config en el pokemon.service

```
import { ConfigService } from '@nestjs/config';


@Injectable()
export class PokemonService {


  constructor(
    @InjectModel(Pokemon.name)
```

```
    private readonly pokemonModel: Model<Pokemon>,
    private readonly configService: ConfigService
  ){}

  {...code}
}
```

- Debo importar el ConfigModule en 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';
import { ConfigModule } from '@nestjs/config';

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

- Puedo inicializar la variable de entorno en el servicio

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


@Injectable()
export class PokemonService {

  private defaultLimit: number | undefined;
```

```
  constructor(
    @InjectModel(Pokemon.name)
    private readonly pokemonModel: Model<Pokemon>,
    private readonly configService: ConfigService
  ){
    this.defaultLimit = this.configService.get<number>('defaultLimit')
  }

  async findAll(paginationDto: PaginationDto) {
    const {limit=this.defaultLimit, offset=0}= paginationDto;
    return await this.pokemonModel.find()
      .limit(limit!)
      .skip(offset)
      .sort({
        no:1
      })
  }

  {...code}
}
```

Joi

- Instalo Joi

```
npm i joi
```

- Creo el ValidationSchema joi-validation.schema.ts

```
import * as Joi from 'joi'


export const joiValidationSchema = Joi.object({
    MONGODB: Joi.required(),
    PORT: Joi.number().default(3001),
    DEFAULT_LIMIT: Joi.number().default(5)
})
```

- Lo agrego a app.module

```
import { Module } from '@nestjs/common';
import { ServeStaticModule } from '@nestjs/serve-static';
import { join } from 'path';
import { PokemonModule } from './pokemon/pokemon.module';
import { MongooseModule } from '@nestjs/mongoose';
import { CommonModule } from './common/common.module';
import { SeedModule } from './seed/seed.module';
import { ConfigModule } from '@nestjs/config';
```

```
import { EnvConfiguration } from './app.config';
import { joiValidationSchema } from './joi-validation.schema';


@Module({
  imports: [
    ConfigModule.forRoot({
      load: [EnvConfiguration],
      validationSchema: joiValidationSchema
    }),
    ServeStaticModule.forRoot({
    rootPath: join(__dirname, '..', 'public')
  }),
  MongooseModule.forRoot('mongodb://localhost:27017/nest-pokemon'),
  PokemonModule,
  CommonModule,
  SeedModule],
  controllers: [],
  providers: [],
})
export class AppModule {}
```

- Parseo a número las variables de entorno, le agrego ! para que no dé error

```
export const EnvConfiguration=()=>({
    environment: process.env.NODE_ENV || 'dev',
    mongodb: process.env.MONGODB,
    port: +process.env.PORT! || 3001,
    defaultLimit: +process.env.DEFAULT_LIMIT! || 5
})
```

Dockerizar app

- Borro la db de Docker
- Creo el Dockerfile en la raíz

```
# Etapa 1: Instalación de dependencias
FROM node:22-alpine AS deps
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm ci

# Etapa 2: Construcción del proyecto
FROM node:22-alpine AS builder
WORKDIR /app
COPY . .
COPY --from=deps /app/node_modules ./node_modules
RUN npm run build
```

```
# Etapa 3: Runner
FROM node:22-alpine AS runner
WORKDIR /usr/src/app
COPY package.json package-lock.json ./
RUN npm ci --omit=dev
COPY --from=builder /app/dist ./dist
CMD ["node", "dist/main"]
```

- .dockerignore

```
dist/
node_modules/
.gitignore
.git/
mongo/
```

- docker-compose.prod.yaml

```
services:
  pokedexapp:
    depends_on:
      - db
    build:
      context: .
      dockerfile: Dockerfile
    image: pokedex-dockerfile
    container_name: pokedexapp
    restart: always
    ports:
      - "${PORT}:${PORT}"
    environment:
      MONGODB: ${MONGODB}
      PORT: "${PORT}"
      DEAFULT_LIMIT: ${DEFAULT_LIMIT}
  db:
    image: mongo:5
    container_name: mongo-poke
    restart: always
    ports:
      - 27017:27017
    environment:
      MONGO_DATABASE: nest-pokemon
    volumes:
      - ./mongo:/data/db
```

- Creo el .env.prod con el string de conexión apuntando al container y el puerto del docker-compose

```
NODE_ENV=prod
MONGODB=mongodb://mongo-poke:27017/nest-pokemon
PORT=3000
DEFAULT_LIMIT=5
```

- Para construir la imagen

```
docker-compose -f docker-compose.prod.yaml --env-file .env.prod up --build
```

- Para usar la imagen

```
docker-compose -f docker-compose.prod.yaml --env-file .env.prod up
```

- **NOTA**: Recuerda usar las variables de entorno! Por ejemplo en app.module con el string de conexión

```
import { Module } from '@nestjs/common';
import { ServeStaticModule } from '@nestjs/serve-static';
import { join } from 'path';
import { PokemonModule } from './pokemon/pokemon.module';
import { MongooseModule } from '@nestjs/mongoose';
import { CommonModule } from './common/common.module';
import { SeedModule } from './seed/seed.module';
import { ConfigModule } from '@nestjs/config';
import { EnvConfiguration } from './app.config';
import { joiValidationSchema } from './joi-validation.schema';


@Module({
  imports: [
    ConfigModule.forRoot({
      load: [EnvConfiguration],
      validationSchema: joiValidationSchema
    }),
    ServeStaticModule.forRoot({
    rootPath: join(__dirname, '..', 'public')
  }),
  MongooseModule.forRoot(process.env.MONGODB!), //AQUI!
  PokemonModule,
  CommonModule,
  SeedModule],
  controllers: [],
  providers: [],
})
export class AppModule {}
```

- Está exponiendo el puerto 3000, por lo que el endpoint seguirá siendo http://localhost:3000/api/v2/pokemon

---

NEST HERRERA - TESLOSHOP

Docker Postgres

- Creo el docker-compose.yaml

```
services:
  db:
    image: postgres:14.3
    restart: always
    ports:
      - "5432:5432"
    environment:
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_DB: ${DB_NAME}
    container_name: teslodb
    volumes:
      - ./postgres:/var/lib/postgresql/data
```

- El README.md

```
# Teslo API

1. Configurar variables de entorno
```

DB_NAME= DB_PASSWORD=

```
2. Levantar la db

> docker-compose up -d
```

Conectar Postgres con Nest

- Instalo

```
npm i @nestjs/typeorm typeorm pg
```

- Instalo

```
npm i @nestjs/config
```

- Configuro las variables de entorno con **ConfigModule.forRoot** en app.module

```
import { Module } from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';
import { TypeOrmModule } from '@nestjs/typeorm';

@Module({
  imports: [
    ConfigModule.forRoot(),
    TypeOrmModule.forRoot({
      type: 'postgres',
      host: process.env.DB_HOST,
      port: +process.env.DB_PORT!,
      database: process.env.DB_NAME,
      username: process.env.DB_USERNAME,
      password: process.env.DB_PASSWORD,
      autoLoadEntities: true,
      synchronize: true
    })

  ],
  controllers: [],
  providers: [],
})
export class AppModule {}
```

- El .env queda así

```
DB_NAME=TesloDB
DB_PASSWORD=123456
DB_HOST=localhost
DB_USERNAME=postgres
DB_PORT=5432
```

TypeORM Entity Product

- Creo la API de Products

> nest g res products --no-spec

- Primero crearemos la entidad sin relaciones

```
import { Column, Entity, PrimaryGeneratedColumn } from "typeorm";

@Entity()
export class Product {

    @PrimaryGeneratedColumn('uuid')
    id: string
```

```
    @Column({
        type: 'text',
        unique: true
    })
    title: string

    @Column({
        type: 'float',
        default: 0
    })
    price: number

    @Column({
        type: 'text',
        nullable: true
    })
    description: string

    @Column({
        type: 'text',
        unique: true
    })
    slug: string

    @Column({
        type: 'int',
        default: 0
    })
    stock: number

    @Column({
        type: 'text',
        array: true
    })
    sizes: string[]

    @Column({
        type: 'text'
    })
    gender: string
}
```

- La añado al products.module

```
import { Module } from '@nestjs/common';
import { ProductsService } from './products.service';
import { ProductsController } from './products.controller';
import { TypeOrmModule } from '@nestjs/typeorm';
import { Product } from './entities/product.entity';

@Module({
  imports:[
```

```
    TypeOrmModule.forFeature([Product])
  ],
  controllers: [ProductsController],
  providers: [ProductsService],
})
export class ProductsModule {}
```

setGlobalPrefix

- En el main

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

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

  app.setGlobalPrefix('api/v1')

  await app.listen(process.env.PORT ?? 3000);
}
bootstrap();
```

create-product.dto

- Instalo

```
npm i class-validator class-transformer
```

- Añado el **useGlobalPipes** en el main

```
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/v1')

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

```
  )

  await app.listen(process.env.PORT ?? 3000);
}
bootstrap();
```

- En el create-product.dto

```
import { IsArray, IsIn, IsNumber, IsOptional, IsString, MinLength } from "class-validator"

export class CreateProductDto {
    @IsString()
    @MinLength(1)
    title: string

    @IsNumber()
    @IsOptional()
    price?: number

    @IsString()
    @IsOptional()
    description?: string

    @IsString()
    @IsOptional()
    slug?: string

    @IsString()
    @IsOptional()
    stock?: number

    @IsString({each: true})
    @IsArray()
    sizes: string[]

    @IsIn(['men', 'women', 'kid', 'unisex'])
    gender: string
}
```

Insertar usando TYpeORM

- Hago uso de **@InjectRepository**

```
import { Injectable } from '@nestjs/common';
import { CreateProductDto } from './dto/create-product.dto';
import { UpdateProductDto } from './dto/update-product.dto';
import { InjectRepository } from '@nestjs/typeorm';
import { Product } from './entities/product.entity';
```

```
import { Repository } from 'typeorm';

@Injectable()
export class ProductsService {


  constructor(
    @InjectRepository(Product)
    private readonly productRepository: Repository<Product>
  ){}

{...code}
}
```

- Hago la inserción dentro de un try catch
- products.service.ts

```
async create(createProductDto: CreateProductDto) {
    try {
      const product = this.productRepository.create(createProductDto)
      await this.productRepository.save(product)
      return product
    } catch (error) {
      console.log(error)
      throw new InternalServerErrorException('No se ha podido crear el producto.
Consultar logs')
    }
  }
```

- En POSTMAN creo un objeto como este

```
"title": "Migue's trousers",
"sizes": ["SM", "M", "L"],
"gender": "men",
"slug": "migues_trousers"
"price": 199.99
```

- Me devuelve

```
{
  "id": "9e077035-8bcf-4556-9e48-383b60bc2bbb",
  "title": "Migue's trousers",
  "price": 199.99,
  "description": null,
  "slug": "migues_trousers",
  "stock": 0,
  "sizes": [
```

```
    "SM",
    "M",
    "L"
  ],
  "gender": "men"
}
```

Manejo de errores (Logger)

- Usaré **el logger de @nestjs/common**
- products.service.ts

```
@Injectable()
export class ProductsService {

  private readonly logger = new Logger('ProductsService')

  constructor(
    @InjectRepository(Product)
    private readonly productRepository: Repository<Product>
  ){}


  async create(createProductDto: CreateProductDto) {
    try {
      const product = this.productRepository.create(createProductDto)
      await this.productRepository.save(product)
      return product
    } catch (error) {

      this.logger.error(error) //Lo uso aquí
      throw new InternalServerErrorException('No se ha podido crear el producto.
Consultar logs')
    }
  }
}
```

- Si intento insertar con el mismo nombre obtengo un error 23505
- Puedo ser más específico en el catch
- products.service

```
async create(createProductDto: CreateProductDto) {
    try {
      const product = this.productRepository.create(createProductDto)
      await this.productRepository.save(product)
      return product
    } catch (error) {
      if(error.code === '23505'){
        throw new BadRequestException(error.detail)
```

```
      }
      this.logger.error(error)
      throw new InternalServerErrorException('Unexpected error. Check server
logs')
    }
  }
```

BeforeInsert y BeforeUpdate

- Si no mando el slug **me da error** porque es requerido ya que en la entity **no tiene el nullable** pero en el dto está como **opcional**
- Lo genero yo basado en el título
- products.service

```
  async create(createProductDto: CreateProductDto) {
    try {

      if(!createProductDto.slug){
        createProductDto.slug = createProductDto.title.toLowerCase().replaceAll('
','_').replaceAll("'", "")
      }else{
        createProductDto.slug = createProductDto.slug.toLowerCase().replaceAll('
','_').replaceAll("'", "")
      }

      const product = this.productRepository.create(createProductDto)
      await this.productRepository.save(product)
      return product

    } catch (error) {
      if(error.code === '23505'){
        throw new BadRequestException(error.detail)
      }
      this.logger.error(error)
      throw new InternalServerErrorException('Unexpected error. Check server
logs')
    }
  }
```

- Puedo usar **@BeforeInsert** para hacer este procedimiento antes de la inserción en la entidad
- products.entity.ts

```
import { BeforeInsert, Column, Entity, PrimaryGeneratedColumn } from "typeorm";

@Entity()
export class Product {
```

```
    @PrimaryGeneratedColumn('uuid')
    id: string

    @Column({
        type: 'text',
        unique: true
    })
    title: string

    @Column({
        type: 'float',
        default: 0
    })
    price: number

    @Column({
        type: 'text',
        nullable: true
    })
    description: string

    @Column({
        type: 'text',
        unique: true
    })
    slug: string

    @Column({
        type: 'int',
        default: 0
    })
    stock: number

    @Column({
        type: 'text',
        array: true
    })
    sizes: string[]

    @Column({
        type: 'text'
    })
    gender: string

    @BeforeInsert()
    checkSlugInsert(){
        if(!this.slug){
            this.slug = this.title //si no viene el slug guardo el titulo en slug
        }

        this.slug = this.slug
        .toLowerCase()
        .replaceAll(" ", "_")
```

```
            .replaceAll("'", "")
        }
    }
```

Get y Delete

- Uso **ParseUUIDPipe** en el controller en los métodos findOne y remove

```
import { Controller, Get, Post, Body, Patch, Param, Delete, ParseUUIDPipe } from
'@nestjs/common';
import { ProductsService } from './products.service';
import { CreateProductDto } from './dto/create-product.dto';
import { UpdateProductDto } from './dto/update-product.dto';

@Controller('products')
export class ProductsController {
  constructor(private readonly productsService: ProductsService) {}

  @Post()
  create(@Body() createProductDto: CreateProductDto) {
    return this.productsService.create(createProductDto);
  }

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

  @Get(':id')
  findOne(@Param('id', ParseUUIDPipe) id: string) {
    return this.productsService.findOne(id);
  }

  @Patch(':id')
  update(@Param('id') id: string, @Body() updateProductDto: UpdateProductDto) {
    return this.productsService.update(+id, updateProductDto);
  }

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

- Creo un método privado para los errores, lo uso en el catch de create
- products.service

```
import { BadRequestException, Injectable, InternalServerErrorException, Logger,
NotFoundException } from '@nestjs/common';
```

```
import { CreateProductDto } from './dto/create-product.dto';
import { UpdateProductDto } from './dto/update-product.dto';
import { InjectRepository } from '@nestjs/typeorm';
import { Product } from './entities/product.entity';
import { Repository } from 'typeorm';

@Injectable()
export class ProductsService {

  private readonly logger = new Logger('ProductsService')

  constructor(
    @InjectRepository(Product)
    private readonly productRepository: Repository<Product>
  ){}


  async create(createProductDto: CreateProductDto) {
    try {

      if(!createProductDto.slug){
        createProductDto.slug = createProductDto.title.toLowerCase().replaceAll('
','_').replaceAll("'", "")
      }else{
        createProductDto.slug = createProductDto.slug.toLowerCase().replaceAll('
','_').replaceAll("'", "")
      }

      const product = this.productRepository.create(createProductDto)
      await this.productRepository.save(product)
      return product

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

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

  async findOne(id: string) {
    const product = await this.productRepository.findOneBy({id})
    if(!product) throw new NotFoundException('Product not found')

    return product
  }

  update(id: number, updateProductDto: UpdateProductDto) {
    return `This action updates a #${id} product`;
  }

  async remove(id: string) {
    const product = await this.findOne(id)
```

```
    await this.productRepository.delete(id)

  }
  //creo un método para los errores
  private handleDBExceptions(error:any){
    if(error.code === '23505'){
        throw new BadRequestException(error.detail)
      }
      this.logger.error(error)
      throw new InternalServerErrorException('Unexpected error. Check server
logs')
  }
}
```

Paginación en TypeORM

- Creo el módulo common
- Dentro el pagination.dto.ts

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

export class PaginationDto{

    @IsOptional()
    @IsPositive() //con IsPositive no hace falta el IsNumber
    @Type(()=> Number)
    limit?: number

    @IsOptional()
    @Min(0)
    @Type(()=> Number)
    offset?: number
}
```

- Hago uso del dto en el controller

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

- Hago la paginación en el servicio

```
async findAll(paginationDto: PaginationDto) {
    const {limit=10, offset=0} = paginationDto
```

```
    const products = await this.productRepository.find({
      take: limit,
      skip: offset
    })
  }
```

Buscar por slug, título o UUID

- Instalo uuid y los types @types/uuid
- Importo validate de uuid y lo renombro a isUUID en products.service

```
import { validate as isUUID } from 'uuid';
```

- Cambio id por term que es más adecuado
- products.service.ts

```
async findOne(term: string) {

  let product: Product | null = null;

  if(isUUID(term)){
    product = await this.productRepository.findOneBy({id: term})
  }else{
    product = await this.productRepository.findOneBy({slug: term})
  }

  if(!product) throw new NotFoundException('Product not found')
  return product
}
```

QueryBuilder

- Quito el ParseUUIDPipe del controller
- Los : significa que son **parámetros**
- Solo me interesa uno de los dos, por eso uso **getOne**
- QueryBuilder es case sensitive, para evitarlo puedo usar UPPER y luego pasar todo a mayúsculas

```
async findOne(term: string) {

  let product: Product | null = null;

  if(isUUID(term)){
    product = await this.productRepository.findOneBy({id: term})
  }else{
    const queryBuilder = this.productRepository.createQueryBuilder('product')
```

```
    product = await queryBuilder.where(`UPPER(product.title) = :title or
product.slug = :slug`, {
      title: term.toUpperCase(),
      slug: term.toLowerCase()
    }).getOne()
  }

  if(!product) throw new NotFoundException('Product not found')
  return product
}

{...code}
```

Update

- Cuando solo hay una tabla implicada el update es sencillo
- products.controller.ts

```
@Patch(':id')
  update(@Param('id', ParseUUIDPipe) id: string, @Body() updateProductDto:
UpdateProductDto) {
    return this.productsService.update(id, updateProductDto);
  }
```

- products.service.ts

```
async update(id: string, updateProductDto: UpdateProductDto) {
    if(isUUID(id)){
      const product = await this.productRepository.preload({
        id,
        ...updateProductDto
      })
      if(!product) throw new NotFoundException('Product not found')
      try {
        await this.productRepository.save(product)
        return product

      } catch (error) {
        this.handleDBExceptions(error)
      }
    }
  }
```

- Si le paso un título que ya existe me va a devolver un InternalServerError
- Los slugs los tengo que validar, deben cumplir las condiciones que establecí
- Para ello usaré **@BeforeUpdate**
- product.entity.ts

```
@BeforeUpdate()
    checkSlugUpdate(){
        this.slug= this.slug
        .toLowerCase()
        .replaceAll(" ", "_")
        .replaceAll(" ", "")
    }
```

Tags

- Puedo usar tags para mejorar las búsquedas
- Como tengo el synchronize en true puedo añadirlo directamente en la entity
- product.entity.ts

```
@Column({
  type: 'text',
  array: true,
  default: []
  })
  tags: string[]
```

- En el dto lo pongo como opcional (como por defecto le pongo un arreglo vacío puedo)

```
@IsString({each: true})
@IsOptional()
@IsArray()
tags?: string[]
```

Relaciones TypeORM

- Creo product-image.entity

```
import { Column, Entity, PrimaryGeneratedColumn } from "typeorm";

@Entity()
export class ProductImage{
    @PrimaryGeneratedColumn()
    id: number

    @Column('text')
    url: string
}
```

- Indico que existe esta entidad en products.module

```
import { Module } from '@nestjs/common';
import { ProductsService } from './products.service';
import { ProductsController } from './products.controller';
import { TypeOrmModule } from '@nestjs/typeorm';
import { Product } from './entities/product.entity';
import { ProductImage } from './entities/product-image.entity';

@Module({
  imports:[
    TypeOrmModule.forFeature([Product, ProductImage])
  ],
  controllers: [ProductsController],
  providers: [ProductsService],
})
export class ProductsModule {}
```

OneToMany y ManyToOne

- Un producto puede tener varias imágenes, por eso OneToMany
- product.entity.ts

```
@OneToMany(
  ()=> ProductImage,
  productImage=> productImage.product, //todavía no he creado el campo product
  {cascade: true} //si hago una eliminación, elimina las imágenes asociadas al
producto
    )
  images?: ProductImage[]
```

- Muchas imágenes pueden tener un único producto, por eso es ManyToOne
- product-image.entity.ts

```
@ManyToOne(
    ()=>Product,
    product=> product.images
)
product: Product
```

- Salta un error: Types of property images are incompatible, Type 'string[] is not assignable to type 'DeepPartialProductImage'
- Por ello añado en el create y el update un arreglo vacío de imágenes
- Para crear las imágenes voy a necesitar inyectar el repositorio. Hago un .map que devuelve un arreglo
- products.service.ts

```
@Injectable()
export class ProductsService {

  private readonly logger = new Logger('ProductsService')

  constructor(
    @InjectRepository(Product)
    private readonly productRepository: Repository<Product>,
    @InjectRepository(ProductImage)
    private readonly productImageRepository: Repository<ProductImage>
  ){}


  async create(createProductDto: CreateProductDto) {
    try {
      const {images=[], ...productDetails}= createProductDto
      const product = this.productRepository.create({
        ...productDetails,
        images: images.map(image=>
this.productImageRepository.create({url:image}))
      })

      await this.productRepository.save(product)
      return product

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

 async update(id: string, updateProductDto: UpdateProductDto) {
    if(isUUID(id)){
      const product = await this.productRepository.preload({
        id,
        ...updateProductDto,
        images: []
      })
      if(!product) throw new NotFoundException('Product not found')
      try {
        await this.productRepository.save(product)
        return product

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


}
```

- Añado images al dto!
- create-product.dto.ts

```
@IsOptional()
@IsString({each: true})
@IsArray()
images?: string[]
```

- Hago un POST con las imagenes con este body (borro los registros anteriores)

```
{
"title": "Migue's trousers",
"sizes": ["SM", "M", "L"],
"gender": "men",
"slug": "migues_trousers",
"price": 199.99,
"images": [
  "http://image1.jpg",
  "http://image2.jpg"
  ]
}
```

- Esto inserta las imagenes con el id de producto en la tabla product_image
- en la tabla products no vemos las imágenes, porque están relacionadas con la tabla product_image

Aplanar las imágenes

- Para que me muestre las imágenes en el método findAll establezco la relación con **relations**
- products.service.ts

```
async findAll(paginationDto: PaginationDto) {
    const {limit=10, offset=0} = paginationDto
    const products = await this.productRepository.find({
      take: limit,
      skip: offset,
      relations:{
        images: true
      }
    })

    return products.map(({images, ...rest})=>({
      ...rest,
      images: images.map(img=>img.url)
    }))
  }
```

- Debo poner el **eager** en true para que funcione relations
- Cuando se usa el QueryBuilder, eager no está disponible, se usa **leftJoinAndSelect**
- product.entity.ts

```
@OneToMany(
    ()=> ProductImage,
    productImage=> productImage.product,
    {cascade: true, eager: true}
)
images: ProductImage[]
```

- Ahora si busco por id con el find me aparecen las imágenes pero por slug no porque uso un QueryBuilder
- Uso leftJoinAndSelect, le pongo el alias prodImages

```
async findOne(term: string) {

  let product: Product | null = null;

  if(isUUID(term)){
    product = await this.productRepository.findOneBy({id: term})
  }else{
    const queryBuilder = this.productRepository.createQueryBuilder('product')

    product = await queryBuilder.where(`UPPER(product.title) = :title or
product.slug = :slug`, {
      title: term.toUpperCase(),
      slug: term.toLowerCase()
    })
    .leftJoinAndSelect('product.images', 'prodImages')
    .getOne()
  }
}
```

- No lo voy a manejar así porque me interesa devolver una instancia de mi entidad y no algo que luzca como tal
- Creo un método para aplanarlo

```
async findOnePlane(term:string){
    const {images=[], ...product} = await this.findOne(term)
    return{
      ...product,
      images: images.map(img=> img.url)
    }
  }
```

QueryRunner

- Si actualizo un producto y no le paso las imágenes aparece el arreglo vacío y pierdo la referencia
- Esto sucede por el cascade en true
- También porque al hacer el update indico que es un arreglo vacío
- Borro todas las imágenes de la db
- Quiero que las imágenes que añado en el body sean las nuevas imágenes
- Entonces son dos cosas: borrar las anteriores e insertar las nuevas
- Si una de las dos falla quiero revertir el proceso. Para ello usaré QueryRunner
- El QueryRunner tiene que conocer la cadena de conexión que estoy usando
- Para ello usaré inyección de dependencies con el DataSource (de TypeORM)
- products.service.ts

```
 async update(id: string, updateProductDto: UpdateProductDto) {
    const {images, ...toUpdate} = updateProductDto

    const product = await this.productRepository.preload({id, ...toUpdate})

    if(!product) throw new NotFoundException(`Product with id ${id} not found`)

      const queryRunner = this.dataSource.createQueryRunner()
      await queryRunner.connect()
      await queryRunner.startTransaction()

      try {
        if(images){
          await queryRunner.manager.delete(ProductImage, {product: {id}})
          product.images =
images?.map(image=>this.productImageRepository.create({url: image}))
        }

        await queryRunner.manager.save(product)
        await queryRunner.commitTransaction()
        await queryRunner.release()

        return this.findOnePlane(id)

      } catch (error) {
        await queryRunner.rollbackTransaction()
        await queryRunner.release()

        this.handleDBExceptions(error)
      }

  }
```

Eliminación en cascada

- Si quiero borrar un producto que tiene una imagen me da error

- Dice que borrar de la tabla producto viola la foreign key de la tabla product_image
- Se puede resolver de varias formas: una es crear una transacción donde borrar primero las imágenes y luego el producto
- También puedo decirle que al borrar un producto se borren las imágenes relacionadas (eliminación en cascada)
- product-image.entity.ts

```
import { Column, Entity, ManyToOne, PrimaryGeneratedColumn } from "typeorm";
import { Product } from "./product.entity";

@Entity()
export class ProductImage{
    @PrimaryGeneratedColumn()
    id: number

    @Column('text')
    url: string

    @ManyToOne(
        ()=>Product,
        product=> product.images,
        {onDelete: 'CASCADE'}
    )
    product: Product
}
```

- Creo un método en el servicio para borrar todos los productos
- products.service.ts

```
async deleteAllProducts(){
  const query = this.productRepository.createQueryBuilder('product')

  try{
    return await query
      .delete()
      .where({})
      .execute()

  } catch(error){
    this.handleDBExceptions(error)
  }
}
```

Product Seed

- Copio el gist de Herrera

> https://gist.github.com/Klerith/1fb1b9f758bb0c5b2253dfc94f09e1b6

- Tengo estas interfaces

```
interface SeedProduct {
    description: string;
    images: string[];
    stock: number;
    price: number;
    sizes: ValidSizes[];
    slug: string;
    tags: string[];
    title: string;
    type: ValidTypes;
    gender: 'men'|'women'|'kid'|'unisex'
}

type ValidSizes = 'XS'|'S'|'M'|'L'|'XL'|'XXL'|'XXXL';
type ValidTypes = 'shirts'|'pants'|'hoodies'|'hats';


interface SeedData {
    products: SeedProduct[];
}
```

- La data luce como un arreglo con objetos como este

```
  {
    description: "Introducing the Tesla Chill Collection. The Men’s Chill Crew
Neck Sweatshirt has a premium, heavyweight exterior and soft fleece interior for
comfort in any season. The sweatshirt features a subtle thermoplastic polyurethane
T logo on the chest and a Tesla wordmark below the back collar. Made from 60%
cotton and 40% recycled polyester.",
    images: [
        '1740176-00-A_0_2000.jpg',
        '1740176-00-A_1.jpg',
    ],
    stock: 7,
    price: 75,
    sizes: ['XS','S','M','L','XL','XXL'],
    slug: "mens_chill_crew_neck_sweatshirt",
    type: 'shirts',
    tags: ['sweatshirt'],
    title: "Men’s Chill Crew Neck Sweatshirt",
    gender: 'men'
   },
```

- Genero el módulo de SEED

```
nest g res seed
```

- Borro dtos, entitys, dejo solo el GET en el controller

```
import { Controller, Get, Post, Body, Patch, Param, Delete } from
'@nestjs/common';
import { SeedService } from './seed.service';

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


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

}
```

- En el servicio

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

@Injectable()
export class SeedService {

  runSeed(){

    return 'SEED EXECUTED'
  }
}
```

- Necesito acceder al servicio para usar el método para borrar todos los productos
- Lo exporto del ProductsModule

```
import { Module } from '@nestjs/common';
import { ProductsService } from './products.service';
import { ProductsController } from './products.controller';
import { TypeOrmModule } from '@nestjs/typeorm';
import { Product } from './entities/product.entity';
import { ProductImage } from './entities/product-image.entity';
import { TypeORMError } from 'typeorm';

@Module({
  imports:[
    TypeOrmModule.forFeature([Product, ProductImage])
  ],
  controllers: [ProductsController],
  providers: [ProductsService],
```

```
  exports: [ProductsService, TypeOrmModule]
})
export class ProductsModule {}
```

- Importo el ProductsModule

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

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

- Lo inyecto en el servicio

```
import { Injectable } from '@nestjs/common';
import { ProductsService } from 'src/products/products.service';
import { initialData } from './data/data';

@Injectable()
export class SeedService {

  constructor(
    private readonly productsService: ProductsService
  ){}

  async runSeed(){

    this.insertNewProducts()

    const products = initialData.products

  const insertPromises = products.map(product =>
this.productsService.create(product));

    await Promise.all(insertPromises)
    return 'SEED EXECUTED'
  }

  private async insertNewProducts(){
    await this.productsService.deleteAllProducts()
  }
}
```

- Creo el README.md

```
# Teslo API


1. Clonar proyecto

> npm i

2. Configurar variables de entorno

~~~env
DB_NAME=TesloDB
DB_PASSWORD=123456
DB_HOST=localhost
DB_USERNAME=postgres
DB_PORT=5432
```

3. Levantar la db

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

4. Ejecutar SEED

```
localhost:3000/api/seed
```

5. Levantar con **npm run start:dev**

```

```

Renombrar tablas

- Las tablas deberían llamarse products y product_images y no product y product_image
- Puedo usar el decorado **@Entity para renombrarlas**
- product.entity.ts

```
@Entity({name: 'products'})
export class Product {

    @PrimaryGeneratedColumn('uuid')
    id: string

{...code}
}
```

- product-images.entity.ts

```
@Entity({name: 'product_images'})
export class ProductImage{
    @PrimaryGeneratedColumn()
    id: number

{...code}
}
```

Carga de archivos

- A través de un POST con el UUID de la imagen en la url voy a mostrar la fotografía
- Instalo

```
npm i @types/multer
```

- Creo el módulo file

```
nest g res file --no-spec
```

- Solo un endpoint POST, sin dtos ni entities

```
import { Controller, Post } from '@nestjs/common';
import { FileService } from './file.service';


@Controller('files')
export class FileController {
  constructor(private readonly fileService: FileService) {}

  @Post('product')
  uploadProductFile(file: Express.Multer.File) {
    return this.fileService.create(file);
  }
}
```

- En POSTMAN, en Body, de tipo form-data, de key le pongo file y al lado puedes elegir el file a subir
- Para poder ver el archivo necesito el decorador **@UploadFile**
- Necesito saber el nombre de la llave para usar el **interceptor**
- Los interceptores interceptan las solicitudes y también pueden interceptar y mutar las respuestas
- Dentro de @UseInterceptors uso FileInterceptor de @nestjs/plattform-express
- Debo indicarle el nombre de la key que haya puesto

```
import { Controller, Post, UploadedFile, UseInterceptors } from '@nestjs/common';
import { FileService } from './file.service';
import { FileInterceptor } from '@nestjs/platform-express';
```

```
@Controller('files')
export class FileController {
  constructor(private readonly fileService: FileService) {}

  @Post('product')
  @UseInterceptors(FileInterceptor('file'))
  uploadProductFile(@UploadedFile() file: Express.Multer.File) {
    return this.fileService.create(file);
  }
}
```

- Por defecto Nest sube el archivo a una carpeta temporal
- No se recomienda guardar el archivo en el filesystem
- Se recomienda un servicio de terceros como **Cloudinary**

Validar archivos

- Usaré un filter, creo la carpeta file/**helpers**/fileFilter.helper.ts
- Para poder usarlo en el FileInterceptor debe cumplir unos requisitos: req, file y callback

```
export const fileFilter = (req: Express.Request, file: Express.Multer.File,
callback: Function)=>{
    if(!file) return callback(new Error('File is empty'), false)

    const fileExtension = file.mimetype.split('/')[1]
    const validExtensions= ['jpg','jpeg', 'png', 'gif']

    if(validExtensions.includes(fileExtension)){
        return callback(null, true)
    }

    callback(null, false)

}
```

- En el controller

```
import { BadRequestException, Controller, Post, UploadedFile, UseInterceptors }
from '@nestjs/common';
import { FileService } from './file.service';
import { FileInterceptor } from '@nestjs/platform-express';
import { fileFilter } from './helpers/fileFilter.helper';
import { diskStorage } from 'multer';


@Controller('files')
export class FileController {
  constructor(private readonly fileService: FileService) {}
```

```
  @Post('product')
  @UseInterceptors(FileInterceptor('file', {
    fileFilter: fileFilter,
    limits: {fileSize: 100000},
    storage: diskStorage({
      destination: 'static/products'
    })
  }))
  uploadProductFile(@UploadedFile() file: Express.Multer.File) {
   if(!file) throw new BadRequestException('Make sure that the file is an image')

    return{
      fileName: file.originalname
    }
  }
}
```

- Esto devuelve

```
{
    "fileName": "FLA2.jpeg"
}
```

- Ahora tengo en la raíz la carpeta .static/products y un archivo con nombre como adbb966fd108dfa519eb4d2d0da2d024

Renombrar el archivo subido

- Copio fileFilter y lo renombro a fileNamer
- Instalo uuid

> npm i uuid @types/uuid

- fileNamer.helper.ts

```
import {v4 as uuid} from 'uuid'

export const fileNamer = (req: Express.Request, file: Express.Multer.File,
callback: Function)=>{
    if(!file) return callback(new Error('File is empty'), false)

    const fileExtension = file.mimetype.split('/')[1]
    const fileName =  `${uuid()}.${fileExtension}`

    callback(null, fileName)

}
```

- Lo uso en el controller

```
@Controller('files')
export class FileController {
  constructor(private readonly fileService: FileService) {}

  @Post('product')
  @UseInterceptors(FileInterceptor('file', {
    fileFilter: fileFilter,
    limits: {fileSize: 100000000},
    storage: diskStorage({
      destination: 'static/products',
      filename: fileNamer
    })
  }))
  uploadProductFile(@UploadedFile() file: Express.Multer.File) {
   if(!file) throw new BadRequestException('Make sure that the file is an image')

    return{
      fileName: file.originalname
    }
  }
}
```

Subir archivos de manera controlada

- No puedo usar el filename para servir el archivo porque no lo sé. Solo estoy grabando el archivo en el filesystem
- Creo la constante secureURL, inyecto el ConfigService
- HOST_API=http://localhost:3000/api/v1
- file.controller.ts

```
import { BadRequestException, Controller, Get, Param, Post, Res, UploadedFile,
UseInterceptors } from '@nestjs/common';
import { FileService } from './file.service';
import { FileInterceptor } from '@nestjs/platform-express';
import { fileFilter } from './helpers/fileFilter.helper';
import { diskStorage } from 'multer';
import { fileNamer } from './helpers/fileNamer.helper';
import { Response } from 'express';
import { ConfigService } from '@nestjs/config';


@Controller('files')
export class FileController {
  constructor(private readonly fileService: FileService,
    private readonly configService: ConfigService
  ) {}
```

```
  @Post('product')
  @UseInterceptors(FileInterceptor('file', {
    fileFilter: fileFilter,
    limits: {fileSize: 100000000},
    storage: diskStorage({
      destination: 'static/products',
      filename: fileNamer
    })
  }))
  uploadProductFile(@UploadedFile() file: Express.Multer.File) {
   if(!file) throw new BadRequestException('Make sure that the file is an image')

    const secureURL=
`${this.configService.get('HOST_API')}/files/product/${file.filename}`
    return{
      secureURL
    }
  }
}
```

- Debo importar el módulo ConfigModule

```
import { Module } from '@nestjs/common';
import { FileService } from './file.service';
import { FileController } from './file.controller';
import { ConfigModule } from '@nestjs/config';

@Module({
  imports:[ConfigModule],
  controllers: [FileController],
  providers: [FileService],
})
export class FileModule {}
```

- Creo un método GET en el file.controller
- Hay que ir con cuidado con usar Res porque se salta ciertos interceptores y restricciones

```
@Get(':imageName')
  findProductImage(@Res() res: Response, @Param('imageName') imageName: string){
    const path = this.fileService.getStaticProductImage(imageName)
    res.sendFile(path)
  }
```

- En el file.service

```
import { BadRequestException, Injectable } from '@nestjs/common';
import { existsSync } from 'fs';
```

```
import { join } from 'path';


@Injectable()
export class FileService {

    getStaticProductImage(imageName: string){
        const path = join(__dirname, '../../static/products', imageName)
        if(!existsSync) throw new BadRequestException(`No product found with image
${imageName}`)

        return path
    }
}
```

- Ahora si hago un post a http://localhost:3000/api/v1/files/product subiendo una imagen me retorna la sevureURL
- La secureURL es algo asi:

```
{
    "secureURL": "http://localhost:3000/api/v1/files/product/3903855a-7b65-4be9-
89df-b771db388230.jpeg"
}
```

- Si apunto con un GET a esta URL me devuelve la imagen

Otras formas de servir archivos

- Usando ServeStaticModule en app.module

```
ServeStaticModule.forRoot([
  rootPath: join(__dirname, '..', 'public')
])
```

- Conviene crear un index.html en la carpeta public
- En el endpoint localhost:3000/assets/nombre_del_archivo puedo acceder a las imágenes
- De esta manera no puedo controlar quien accede a las imágenes
- Son recursos públicos, estáticos, que no van a cambiar
- **Copio los archivos descargados en la carpeta static/products**

Autenticación

- Crearemos decoradores personalizados
- Las rutas GET serán publicas, el resto requerirán autenticación

- Haremos modificaciones en el SEED para crear usuarios automáticamente

Entidad de usuarios

- Creo el módulo de auth con **nest g res --no-spec**
- users.entity.ts

```
import { IsEmail, IsString } from "class-validator";
import { Column, Entity, PrimaryGeneratedColumn } from "typeorm";

@Entity()
export class User {

    @PrimaryGeneratedColumn('uuid')
    id: string


    @Column({
        type: 'text'
    })
    fullName: string

    @Column('text',{
    select: false //para que no me lo devuelva en la peticion
    })
    password: string

    @Column({
        type: 'text',
        unique: true
    })
    email: string

    @Column({
        type: 'bool',
        default: true
    })
    isActive: boolean

    @Column({
        type: 'text',
        array: true,
        default: ['user']
    })
    roles: string[]
}
```

- Declaro la entity en AuthModule con **TypeOrmModule.forFeature([])**

```
import { Module } from '@nestjs/common';
import { AuthService } from './auth.service';
import { AuthController } from './auth.controller';
import { TypeORMError } from 'typeorm';
import { TypeOrmModule } from '@nestjs/typeorm';
import { User } from './entities/user.entity';

@Module({
  imports: [TypeOrmModule.forFeature([User])],
  controllers: [AuthController],
  providers: [AuthService],
  exports: [TypeOrmModule]
})
export class AuthModule {}
```

Crear Usuario

- Para crear usuario usaré el endpoint register

> http://localhost:3000/api/v1/auth/register

- auth.controller.ts

```
@Controller('auth')
export class AuthController {
  constructor(private readonly authService: AuthService) {}

  @Post('register')
  create(@Body() createUserDto: CreateUserDto) {
    return this.authService.create(createUserDto);
  }
  {...code}
}
```

- CreateUserDto

```
import { IsEmail, IsString, Matches, MaxLength, MinLength } from "class-validator"

export class CreateUserDto {

    @IsEmail()
    email: string

    @IsString()
    @MinLength(6)
    @MaxLength(50)
    @Matches(
        /(?:(?=.*\d)|(?=.*\W+))(?![.\n])(?=.*[A-Z])(?=.*[a-z]).*$/, {
```

```
            message: 'The password must have a uppercase. lowercase letter and a
number'
        }
    )
    password: string

    @IsString()
    @MinLength(1)
    fullName: string

}
```

- Para encriptar la contraseña uso

> npm i bcrypt @types/bcrypt

- Para que no retorne el password lo seteo a null
- auth.service.ts

```
import * as bcrypt from 'bcrypt'

@Injectable()
export class AuthService {

  constructor(
    @InjectRepository(User)
    private readonly userRepository: Repository<User>
  ){}

  async create(createUserDto: CreateUserDto) {

    try {
      const {password, ...userData} = createUserDto

    const user = this.userRepository.create({
      ...userData,
      password: bcrypt.hashSync(password, 12)
    })

    await this.userRepository.save(user)

     return {
      ...user,
      password: null
    }

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

    handleDbError(error: any): void{
```

```
    if(error.code === '23505'){
      throw new BadRequestException(error.detail)
    }

    console.log(error)
    throw new InternalServerErrorException('Check logs')
  }
}
```

Login

- Creo el dto login-user.dto

```
import { IsEmail, IsString, Matches, MaxLength, MinLength } from "class-validator"

export class LoginUserDto{

    @IsEmail()
    email: string

    @IsString()
    @MinLength(6)
    @MaxLength(50)
    @Matches(
        /(?:(?=.*\d)|(?=.*\W+))(?![.\n])(?=.*[A-Z])(?=.*[a-z]).*$/, {
            message: 'The password must have a uppercase. lowercase letter and a
number'
        }
    )
    password: string

}
```

- El endpoint en el controller
- auth.controller.ts

```
@Post('login')
loginUser(@Body() loginUserDto: LoginUserDto){
  return this.authService.loginUser(loginUserDto)
}
```

- En el findOneBy({email}) no me devuelve el password porque le puse el **select: false en la entidad**
- Pero lo necesito para validar el password del login, uso el **where** con **findOne**
- auth.service.ts

```

```

```
import { BadRequestException, Injectable, InternalServerErrorException,
UnauthorizedException } from '@nestjs/common';
import { CreateUserDto } from './dto/create-user.dto';
import { UpdateAuthDto } from './dto/update-auth.dto';
import { InjectRepository } from '@nestjs/typeorm';
import { User } from './entities/user.entity';
import { Repository } from 'typeorm';
import * as bcrypt from 'bcrypt'
import { LoginUserDto } from './dto/login-user.dto';

@Injectable()
export class AuthService {

  constructor(
    @InjectRepository(User)
    private readonly userRepository: Repository<User>
  ){}

  async create(createUserDto: CreateUserDto) {

    try {
      const {password, ...userData} = createUserDto

    const user = this.userRepository.create({
      ...userData,
      password: bcrypt.hashSync(password, 12)
    })

    await this.userRepository.save(user)

    return {
      ...user,
      password: null //elimino el password en el retorno
    }


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

  async loginUser(loginUserDto: LoginUserDto){
        const {email, password} = loginUserDto

        const user = await this.userRepository.findOne({
          where: {email},
          select: {email: true, password: true}
        })

        if(!user){
          throw new UnauthorizedException('Credentials are invalid')
        }
```

```
        if(!bcrypt.compareSync(password, user.password)){
          throw new UnauthorizedException('Password is not valid')
        }

        return {
          ...user,
          password: null
        }
  }

  {...code}
}
```

Passport

- Instalo

```
npm i @nestjs/passport passport @nestjs/jwt passport-jwt npm i -D @types/passport-jwt
```

- Uso **PassportModule** en **AuthModule**

```
import { Module } from '@nestjs/common';
import { AuthService } from './auth.service';
import { AuthController } from './auth.controller';
import { TypeORMError } from 'typeorm';
import { TypeOrmModule } from '@nestjs/typeorm';
import { User } from './entities/user.entity';
import { PassportModule } from '@nestjs/passport';
import { JwtModule } from '@nestjs/jwt';

@Module({
  imports: [
    TypeOrmModule.forFeature([User]),
    PassportModule.register({defaultStrategy: 'jwt'}),
    JwtModule.register({
      secret: process.env.JWT_SECRET,
      signOptions:{
        expiresIn: '2h'
      }
    })
],
  controllers: [AuthController],
  providers: [AuthService],
  exports: [TypeOrmModule]
})
export class AuthModule {}
```

- Sería mejor usar el módulo asíncrono para asegurarnos de que la variable de entorno esté cargada
- Para ello haré uso de **useFactory**
- auth.module.ts

```
import { Module } from '@nestjs/common';
import { AuthService } from './auth.service';
import { AuthController } from './auth.controller';
import { TypeORMError } from 'typeorm';
import { TypeOrmModule } from '@nestjs/typeorm';
import { User } from './entities/user.entity';
import { PassportModule } from '@nestjs/passport';
import { JwtModule } from '@nestjs/jwt';
import { ConfigModule, ConfigService } from '@nestjs/config';

@Module({
  imports: [
    TypeOrmModule.forFeature([User]),
    PassportModule.register({defaultStrategy: 'jwt'}),
    JwtModule.registerAsync({
    imports:[ConfigModule],
    inject: [ConfigService],
    useFactory: (configService: ConfigService)=>{
      return{
          secret: process.env.JWT_SECRET,
          signOptions:{
          expiresIn: '2h'
      }
    }
    }
  })
],
  controllers: [AuthController],
  providers: [AuthService],
  exports: [TypeOrmModule]
})
export class AuthModule {}
```

JWTStrategy

- Es recomendable guardar en el jwt algún campo indexado, añadir también en que momento fue creado y fecha de expiración
- Solo guardaré el correo en el jwt
- Todas las estrategias son providers, le coloco el decorador @Injectable y lo indico en el auth.module.ts
- Me pide el método validate que ejecutará automáticamente al definir jwt como strategy
- jwt.strategy.ts

```
import { PassportStrategy } from "@nestjs/passport";
import { InjectRepository } from "@nestjs/typeorm";
import { ExtractJwt, Strategy } from "passport-jwt";
import { Repository } from "typeorm";
import { User } from "../entities/user.entity";
import { ConfigService } from "@nestjs/config";
import { JwtPayloadInterface } from "../interfaces/jwt-payload.interface";
```

```
import { Injectable, UnauthorizedException } from "@nestjs/common";

@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy){

    constructor(
        @InjectRepository(User)
        private readonly userRepository: Repository<User>,
        private readonly configService: ConfigService
    ){

    super({
        secretOrKey: configService.get('JWT_SECRET')!,
        jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken()
    })
    }

    async validate(payload: JwtPayloadInterface): Promise<User>{
        const {email} = payload

        const user = await this.userRepository.findOneBy({email})

        if(!user) throw new UnauthorizedException('Token not valid')
        if(!user.isActive) throw new UnauthorizedException('User is not active')

        return user
    }
}
```

- auth.module

```
import { Module } from '@nestjs/common';
import { AuthService } from './auth.service';
import { AuthController } from './auth.controller';
import { TypeORMError } from 'typeorm';
import { TypeOrmModule } from '@nestjs/typeorm';
import { User } from './entities/user.entity';
import { PassportModule } from '@nestjs/passport';
import { JwtModule } from '@nestjs/jwt';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { JwtStrategy } from './strategies/jwt.strategy';

@Module({
  imports: [
    ConfigModule,
    TypeOrmModule.forFeature([User]),
    PassportModule.register({defaultStrategy: 'jwt'}),
    JwtModule.registerAsync({
    imports:[ConfigModule],
    inject: [ConfigService],
    useFactory: (configService: ConfigService)=>{
      return{
```

```
          secret: process.env.JWT_SECRET,
          signOptions:{
          expiresIn: '2h'
      }
    }
    }
  })
],
  controllers: [AuthController],
  providers: [AuthService, JwtStrategy],
  exports: [TypeOrmModule, JwtStrategy, PassportModule, JwtModule] //exporto!
})
export class AuthModule {}
```

- La interfaz, luego cambiaremos el email por el id

```
export interface JwtPayloadInterface{
    email: string
}
```

Generar Jwt

- Para generar el token necesito del **servicio de jwt de Nest del JwtModule**
- Uso .sign para crear el token
- auth.service.ts

```
import { BadRequestException, Injectable, InternalServerErrorException,
UnauthorizedException } from '@nestjs/common';
import { CreateUserDto } from './dto/create-user.dto';
import { UpdateAuthDto } from './dto/update-auth.dto';
import { InjectRepository } from '@nestjs/typeorm';
import { User } from './entities/user.entity';
import { Repository } from 'typeorm';
import * as bcrypt from 'bcrypt'
import { LoginUserDto } from './dto/login-user.dto';
import { JwtService } from '@nestjs/jwt';
import { JwtPayloadInterface } from './interfaces/jwt-payload.interface';

@Injectable()
export class AuthService {

  constructor(
    @InjectRepository(User)
    private readonly userRepository: Repository<User>,
    private readonly jwtService: JwtService
  ){}

  async create(createUserDto: CreateUserDto) {
```

```
    try {
      const {password, ...userData} = createUserDto

    const user = this.userRepository.create({
      ...userData,
      password: bcrypt.hashSync(password, 12)
    })

    await this.userRepository.save(user)

    return{
      ...user,
      password: null,
      token: this.getJwt({email: user.email}) //creo el token
    }


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

  async loginUser(loginUserDto: LoginUserDto){
        const {email, password} = loginUserDto

        const user = await this.userRepository.findOne({
          where: {email},
          select: {email: true, password: true}
        })

        if(!user){
          throw new UnauthorizedException('Credentials are invalid')
        }

        if(!bcrypt.compareSync(password, user.password)){
          throw new UnauthorizedException('Password is not valid')
        }

        return {
          ...user,
          password: null,
          token: this.getJwt({email: user.email}) //creo el token
        }
  }

  handleDbError(error: any): void{
    if(error.code === '23505'){
      throw new BadRequestException(error.detail)
    }

    console.log(error)
    throw new InternalServerErrorException('Check logs')
  }
```

```
  private getJwt(payload: JwtPayloadInterface){
    const token = this.jwtService.sign(payload) //uso sign para crear el token
    return token
  }
}
```

- Quiero guardar todo en minúsculas, lo hago usando **@BeforeInsert** y **@BeforeUpdate** (es el mismo código) en la entidad
- user.entity.ts

```
@BeforeInsert()
    checkFieldsBeforeInsert(){
        this.email = this.email.toLowerCase().trim()
    }

@BeforeUpdate()
checkFieldsBeforeUpdate(){
    this.checkFieldsBeforeInsert()
}
```

- Apuntando a auth/register mando un objeto como este en el body

```
{
  "email": "migue@gmail.com",
  "password":"1Migue",
  "fullName":"Miguel Pernas"
}
```

- Recibo esto de respuesta

```
{
  "id": "f16a2c17-dc8e-4077-a33c-dc0571d93b25",
  "fullName": "Miguel Pernas",
  "password": null,
  "email": "migue@gmail.com",
  "isActive": true,
  "roles": [
    "user"
  ],
  "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6Im1pZ3VlQGdtYWlsLmNvbSIsImlhdCI6
MTc0ODYyODcwNCwiZXhwIjoxNzQ4NjM1OTA0fQ.Ap-
0iQk7xHhkKvDhBG32x6Rrem4P7LW6HmrgOy5X3FI"
}
```

Private Route - General

- Creo mi primera ruta privada que su único objetivo es asegurar de que hay un jwt, que el usuario esté activo y que el token n haya expirado

```
@Get('private')
@UseGuards(AuthGuard())
  testingPrivateRoute(){
    return {
      ok: true
    }
  }
```

- Los **Guards** son usados para **permitir o prevenir el acceso a una ruta**
- **Es donde se debe autorizar una solicitud**
- Autenticación y autorización no son lo mismo
    - Autenticado es cuando el usuario está validado y autorizado es que tiene permiso para acceder
- Se usa **@UseGuards**, mças adelante haremos un usario especializado
- Uso AuthGuard de @nestjs/passport que usa la estrategia que yo definí por defecto
- Para probarlo en POSTMAN debo añadir el token proporcionado en el login en Auth/Bearer

Cambiar el email por el id en el payload

- Primero cambiamos la interfaz

```
export interface JwtPayloadInterface{
    id: string
}
```

- En el auth.service

```
@Injectable()
export class AuthService {

  constructor(
    @InjectRepository(User)
    private readonly userRepository: Repository<User>,
    private readonly jwtService: JwtService
  ){}

  async create(createUserDto: CreateUserDto) {

    try {
      const {password, ...userData} = createUserDto

    const user = this.userRepository.create({
      ...userData,
      password: bcrypt.hashSync(password, 12)
    })
```

```
    await this.userRepository.save(user)

    return{
      ...user,
      password: null,
      token: this.getJwt({id: user.id})
    }


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

  async loginUser(loginUserDto: LoginUserDto){
        const {email, password} = loginUserDto

        const user = await this.userRepository.findOne({
          where: {email},
          select: {email: true, password: true, id: true}
        })

        if(!user){
          throw new UnauthorizedException('Credentials are invalid')
        }

        if(!bcrypt.compareSync(password, user.password)){
          throw new UnauthorizedException('Password is not valid')
        }

        return {
          ...user,
          password: null,
          token: this.getJwt({id: user.id})
        }
  }
  {...code}
}
```

- Debo cambiarlo también la estrategia!

```
import { PassportStrategy } from "@nestjs/passport";
import { InjectRepository } from "@nestjs/typeorm";
import { ExtractJwt, Strategy } from "passport-jwt";
import { Repository } from "typeorm";
import { User } from "../entities/user.entity";
import { ConfigService } from "@nestjs/config";
import { JwtPayloadInterface } from "../interfaces/jwt-payload.interface";
import { Injectable, UnauthorizedException } from "@nestjs/common";

@Injectable()
```

```
export class JwtStrategy extends PassportStrategy(Strategy){

    constructor(
        @InjectRepository(User)
        private readonly userRepository: Repository<User>,
        private readonly configService: ConfigService
    ){

    super({
        secretOrKey: configService.get('JWT_SECRET')!,
        jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken()
    })
    }

  //la estrategia ejecuta este método automáticamente
    async validate(payload: JwtPayloadInterface): Promise<User>{
        const {id} = payload //aquí!

        const user = await this.userRepository.findOneBy({id}) //aqui!

        if(!user) throw new UnauthorizedException('Token not valid')
        if(!user.isActive) throw new UnauthorizedException('User is not active')

        return user
    }
}
```

- Si ahora vuelvo a apuntar a /auth/register con el token del login debería dejarme ver el ok: true

Custom Property Decorator - GetUser

- Puedo extraer el usuario del Guard
- Si se me olvidara que tengo implementado el Guard y quisiera extraer el usuario debería lanzar un error propio

> nest g d getUser

- Este decorador funciona de manera global, por clase y controlador, no por propiedad
- Para extraer el usuario usaré **@Request** de @nestjs/common

```
@Get('private')
  @UseGuards(AuthGuard())
  testingPrivateRoute(@Request() request: Express.Request){
    return {
      ok: true
    }
  }
```

- **request.user** me devuelve el usuario
- No es muy bonito y necesitaría algunas validaciones

- Usaremos el auth/decorators/get-user.decorator.ts
- Notar que la data y el ctx están dentro de un callback

```
import { createParamDecorator, ExecutionContext, InternalServerErrorException }
from '@nestjs/common';


export const GetUser = createParamDecorator((data, ctx: ExecutionContext)=>{
    const req = ctx.switchToHttp().getRequest()

    const user = req.user

    if(!user) throw new InternalServerErrorException('User not found')

    return user
})
```

- Quiero usar dos veces el @GetUser en el mismo endpoint
- Una sin pasarle ningún argumento que me devuelva el user completo
- Otra pasándole solo el email como parámetro a @GetUser
- Podría usar los Pipes para validar/transformar la data, pero no es el caso

```
@Get('private')
@UseGuards(AuthGuard())
testingPrivateRoute(
  @GetUser() user: User,
  @GetUser('email') email:string
){
  return {
    ok: true
  }
}
```

- En el decorador uso un ternario para devolver si no hay data el user y si no el user.propiedad.computada

```
import { createParamDecorator, ExecutionContext, InternalServerErrorException }
from '@nestjs/common';


export const GetUser = createParamDecorator((data, ctx: ExecutionContext)=>{
    const req = ctx.switchToHttp().getRequest()

    const user = req.user

    if(!user) throw new InternalServerErrorException('User not found')
```

```
    return (!data)? user: user[data]
})
```

- Creo un decorador que me devuelva **lo que yo quiera de @Request**, por ejemplo los headers
- getRawHeaders.decorator.ts

```
import { createParamDecorator, ExecutionContext } from "@nestjs/common";

export const GetRawHeaders = createParamDecorator((data, ctx: ExecutionContext)=>{
    const req = ctx.switchToHttp().getRequest()

    return req.rawHeaders
})
```

- En el auth.controller

```
@Get('private')
  @UseGuards(AuthGuard())
  testingPrivateRoute(
    @GetUser() user: User,
    @GetUser('email') email:string,
    @GetRawHeaders() rawHeaders: string[]
  ){
    return {
      ok: true,
      user,
      email,
      rawHeaders
    }
  }
```

Custom Guard y Custom Decorator

- Si quisiera validar el rol podría hacerlo en el controlador con user.roles.includes('admin')
- Pero voy a crear un Guard y un Custom Decorator para esta tarea
- Para ello creo un nuevo endpoint GET en el controller

```
@Get('private2')
@UseGuards(AuthGuard())
testingPrivateRoute2(@GetUser() user: User)
  {
  return {
    ok: true,
    user
```

```
  }
}
```

- Este GET necesita ciertos roles podría usar **@SetMetadata** para validarlos
- **Pero con esto no es suficiente debo crear un GUard para que los evalúe**
- Ejemplo @SetMetadata insuficiente

```
@Get('private2')
@UseGuards(AuthGuard())
@SetMetadata('roles', ['admin'])
testingPrivateRoute2(
  @GetUser() user: User)
  {
  return {
    ok: true,
    user
  }
}
```

- Creo el Guard

> nest g gu auth/guards/userRole --no-spec+

- Los Guards son async por defecto
- En este caso no lleva el paréntesis en el controlador porque AuthGuard ya devuelve la instancia
  - Los Guards personalizados no llevan paréntesis para usar la misma instancia
- Los Guards están dentro de la Exception Zone de Nest
- Para verificar los roles debo extraer la data del decorador @SetMetadata
- **Inyecto Reflector en el constructor**
- Lo uso para guardar en la variable .get('roles') lo que pone en @SetMetadata, el target es context.getHandler()
- El UserRoleGuard

```
import { BadRequestException, CanActivate, ExecutionContext, ForbiddenException,
Injectable } from '@nestjs/common';
import { Reflector } from '@nestjs/core';
import { Observable } from 'rxjs';
import { User } from 'src/auth/entities/user.entity';

@Injectable()
export class UserRoleGuard implements CanActivate {

  constructor(
    private readonly reflector: Reflector
  ){}

  canActivate(
    context: ExecutionContext,
```

```
  ): boolean | Promise<boolean> | Observable<boolean> {

    const validRoles: string[]= this.reflector.get('roles', context.getHandler())

    if(!ValidRoles) return true //que pase en el caso de que no haya roles
    if(validRoles.length===0) return true //lo mismo

    const req = context.switchToHttp().getRequest()
    const user = req.user as User

    if(!user) throw new BadRequestException('User not found')

    for(const role of user.roles){
      if(validRoles.includes(role)){
        return true
      }
    }

    throw new ForbiddenException(`User ${user.fullName} needs a valid role`)
  }
}
```

- En el auth.controller

```
@Get('private2')
@UseGuards(AuthGuard(), UserRoleGuard)
@SetMetadata('roles', ['admin'])
testingPrivateRoute2(
  @GetUser() user: User)
  {
  return {
    ok: true,
    user
  }
}
```

- Para implementar esta lógica debo usar el @SetMetadata
- Si se me olvidara, al extraer los validRoles de mi app reventaría. debería validarlo
- El arreglo de roles es muy volátil, me puedo equivocar
- **El @SetMetadata es poco común como decorador directamente**
- Mejor crear un **Custom Decorator**

Custom Decorator Role Protected

- Si no son decoradores de propiedades, perfectamente puedo usar el CLI
- Cómo me va a servir para establecer los roles que el usario debe de tener para acceder a la ruta el decorador está ligado al módulo de auth

> nest g d auth/decorators/RoleProtected --no-spec

- auth/decorators/role-protected.decorator.ts

```
import { SetMetadata } from '@nestjs/common';
import { ValidRoles } from '../interfaces/valid-roles.enum';

export const META_ROLES = 'roles'


export const RoleProtected = (...args: ValidRoles[]) => {
    return SetMetadata(META_ROLES, args)
};
```

- auth/interfaces/valid-roles.enum.ts

```
export enum ValidRoles{
    admin='admin',
    superUser = 'super-user',
    user='user'
}
```

- Uso @RoleProtected en el auth.controller

```
@Get('private2')
@UseGuards(AuthGuard(), UserRoleGuard)
@RoleProtected(ValidRoles.admin)
testingPrivateRoute2(
  @GetUser() user: User)
  {
  return {
    ok: true,
    user
  }
}
```

- Puedo pasarle valores separados por comas
- Es fácil que me olvide de implementar el AuthGuard, el RoleProtected
- **Crearemos un único decorador para hacer todo esto**

Composición de decoradores

- Se usa **applyDecorators** de @nestjs/common para hacer composición de decoradores
- Creo el auth.decorator.ts

```
import { applyDecorators, UseGuards } from "@nestjs/common";
import { ValidRoles } from "../interfaces/valid-roles.enum";
```

```
import { RoleProtected } from "./role-protected.decorator";
import { AuthGuard } from "@nestjs/passport";
import { UserRoleGuard } from "../guards/user-role.guard";



export function Auth(...roles: ValidRoles[]){
    return applyDecorators(
        RoleProtected(...roles),
        UseGuards(AuthGuard('jwt'), UserRoleGuard)
    )
}
```

- Lo uso en el auth.controller
- Recuerda que en el UserRoleGuard hay dos lineas que permiten pasar sin ValidRoles, por lo que puedo pasarle un rol o no para la autorización

```
@Get('private2')
@Auth(ValidRoles.admin)
testingPrivateRoute2(
  @GetUser() user: User)
  {
  return {
    ok: true,
    user
  }
}
```

Auth en otros módulos

- Quiero usar el decorador @Auth en mi seed.controller.ts
- @Auth está usando @AuthGuard que **está asociado a Passport** y **Passort es un módulo**
- Auth **pide el defaultStrategy**, por lo que **en el módulo de Auth exporto JwtStrategy y PassportModule**

```
import { Module } from '@nestjs/common';
import { AuthService } from './auth.service';
import { AuthController } from './auth.controller';
import { TypeORMError } from 'typeorm';
import { TypeOrmModule } from '@nestjs/typeorm';
import { User } from './entities/user.entity';
import { PassportModule } from '@nestjs/passport';
import { JwtModule } from '@nestjs/jwt';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { JwtStrategy } from './strategies/jwt.strategy';

@Module({
  imports: [
```

```
    ConfigModule,
    TypeOrmModule.forFeature([User]),
    PassportModule.register({defaultStrategy: 'jwt'}),
    JwtModule.registerAsync({
    imports:[ConfigModule],
    inject: [ConfigService],
    useFactory: (configService: ConfigService)=>{
      return{
          secret: process.env.JWT_SECRET,
          signOptions:{
          expiresIn: '2h'
      }
    }
    }
  })
],
  controllers: [AuthController],
  providers: [AuthService, JwtStrategy],
  exports: [TypeOrmModule, JwtStrategy, PassportModule, JwtModule]
})
export class AuthModule {}
```

- Importo AuthModule en el seed.module

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

- Ahora puedo usar el decorador en el seed.controller

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

  @Get()
  @Auth(ValidRoles.admin)
  executeSeed(){
    return this.seedService.runSeed();
  }
}
```

- Si quiero que todos los endpoints necesiten autorización puedo colocar el **@Auth a nivel de controlador**

Usuario que creó el producto

- Sería útil saber que usuario creó el producto
- Un usuario puede crear varios productos (**@OneTo Many**)
- Muchos productos pueden ser de un usuario (**@ManyToOne**)
- El OneToMany no va a crear ninguna nueva columna en user, pero en product si
- user.entity

```
@OneToMany(
    ()=> Product,
    (product)=> product.user
)
product: Product
```

- product.entity

```
@ManyToOne(
      ()=> User,
      (user)=> user.product,
      {eager: true} //para que lo muestre en las búsquedas
  )
user: User
```

Insertar userId en los productos

- En el **products.module** debo importar **auth.module**
- Solo los admin van a poder crear productos
- Uso el decorador **@GetUser** para obtener el usuario
- Hago lo mismo en el update
- products.controller.ts

```
@Post()
@Auth(ValidRoles.admin)
create(@Body() createProductDto: CreateProductDto,
        @GetUser() user: User) {
  return this.productsService.create(createProductDto, user);
}

@Patch(':id')
@Auth(ValidRoles.admin)
update(@Param('id', ParseUUIDPipe) id: string,
        @Body() updateProductDto: UpdateProductDto,
        @GetUser() user: User) {
  return this.productsService.update(id, updateProductDto, user);
}
```

- En el products.service.ts

```
async create(createProductDto: CreateProductDto, user: User) {
  try {
    const {images=[], ...productDetails}= createProductDto
    const product = this.productRepository.create({
      ...productDetails,
      images: images.map(image=> this.productImageRepository.create({url:image})),
      user
    })

    await this.productRepository.save(product)
    return product

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

async update(id: string, updateProductDto: UpdateProductDto, user: User) {
  const {images, ...toUpdate} = updateProductDto

  const product = await this.productRepository.preload({id, ...toUpdate})

  if(!product) throw new NotFoundException(`Product with id ${id} not found`)

    const queryRunner = this.dataSource.createQueryRunner()
    await queryRunner.connect()
    await queryRunner.startTransaction()

    try {

      if(images){
        await queryRunner.manager.delete(ProductImage, {product: {id}})
        product.images =
images?.map(image=>this.productImageRepository.create({url: image}))

      }

      product.user = user

      await queryRunner.manager.save(product)
      await queryRunner.commitTransaction()
      await queryRunner.release()

      return this.findOnePlane(id)

    } catch (error) {
      await queryRunner.rollbackTransaction()
      await queryRunner.release()

      this.handleDBExceptions(error)
    }

}
```

- Muteo estas dos lineas del SEED que me da error porque no le paso el user para poder crear un producto
- seed.service.ts

```
import { Injectable } from '@nestjs/common';
import { ProductsService } from 'src/products/products.service';
import { initialData } from './data/data';

@Injectable()
export class SeedService {

  constructor(
    private readonly productsService: ProductsService
  ){}

  async runSeed(){

    this.insertNewProducts()

    const products = initialData.products

  //const insertPromises = products.map(product =>
this.productsService.create(product));

  //await Promise.all(insertPromises)
    return 'SEED EXECUTED'
  }

  private async insertNewProducts(){
    await this.productsService.deleteAllProducts()
  }

}
```

- Como tengo el eager en true, en la respuesta me carga el usuario
- Falta arreglar lo del user en el SEED
- **NOTA**: si al crear el producto aparece un error Cannot read properties of undefined (reading 'challenge') es porque en la composición del decorador @AuthGuard **falta pasarle el 'jwt'**

SEED de usuarios, productos e imágenes

- Creo un método para purgar las tablas
- Si intento borrar primero los usuarios, estos están siendo utilizados por los productos
- Borro todos los productos con el servicio de productos
- Para los usuarios debo inyectar el servicio de usuarios
- Estoy exportando TypeORM dónde venía el usuario, **por eso no da error**
- Creo la interfaz seed-data.ts
- Añado los users

- seed-data.ts

```
import * as bcrypt form 'bcrypt'

interface SeedProduct {
    description: string;
    images: string[];
    stock: number;
    price: number;
    sizes: ValidSizes[];
    slug: string;
    tags: string[];
    title: string;
    type: ValidTypes;
    gender: 'men'|'women'|'kid'|'unisex'
}

export interface SeedUser{
    email: string,
    fullName: string,
    password: string,
    roles: string[]
}

type ValidSizes = 'XS'|'S'|'M'|'L'|'XL'|'XXL'|'XXXL';
type ValidTypes = 'shirts'|'pants'|'hoodies'|'hats';


interface SeedData {
    users: SeedUser[],
    products: SeedProduct[];
}


export const initialData: SeedData = {

    users:[
        {
            email: "test1@google.com",
            fullName: "Migue Sensei",
            password: bcrypt.hashSync("Abc123456", 10),
            roles: ['admin']
        },
        {
            email: "test2@google.com",
            fullName: "Anna Kunoichi",
            password: bcrypt.hashSync("Abc123456", 10),
            roles: ['user', 'super']
        }
    ],
    products:[...code]
}
```

- Ahora puedo usar los usuarios para insertarlos en el servicio
- Este firstUser que me retorna se lo paso a insertNewProducts y se lo paso al forEach, para que lo inserte en cada producto
- seed.service.ts

```
@Injectable()
export class SeedService {

  constructor(
    private readonly productsService: ProductsService,
    @InjectRepository(User)
    private readonly userRepository : Repository<User>
  ){}

  private async insertUsers(){
    const seedUsers = initialData.users
    const users: User[] = []

    seedUsers.forEach(user=>{
      users.push(this.userRepository.create(user))
    })

    const dbUsers = await this.userRepository.save(seedUsers)

    return dbUsers[0]
  }

  async runSeed(){

    this.deleteTables()
    const firstUser = await this.insertUsers()

    this.insertNewProducts(firstUser)

    const products = initialData.products
    return 'SEED EXECUTED'
  }

  private async deleteTables(){
    await this.productsService.deleteAllProducts()

    const queryBuilder = this.userRepository.createQueryBuilder()

    await queryBuilder
    .delete()
    .where({})
    .execute()
  }


private async insertNewProducts(user: User){
```

```
    await this.productsService.deleteAllProducts();
    const products = initialData.products;
    const insertPromises: Promise<Product | undefined>[] = [];

    products.forEach(product => {
      insertPromises.push(this.productsService.create(product, user));
    });

    await Promise.all(insertPromises);

      return 'SEED EXECUTED'
    }
  }
```

CheckAuthStatus

- Falta poder revalidar el token. Usar el token suministrado y generar un nuevo token basado en el anterior
- Si no hago esto, si el usuario refresca el navegador perderá la autenticación
- auth.controller.ts

```
@Get('check-auth')
  @Auth()
  checkAuthStatus(
    @GetUser() user: User
  ){
    return this.authService.checkAuthStatus(user)
  }
```

- En el auth.service

```
async checkAuthStatus(user: User){
  return {
    ...user,
    token: this.getJwt({id: user.id})
  }
}
```

- En la respuesta regreso un nuevo JWT y la info de name, fullName, por si le sirve al frontend
- El usuario tiene que estar activo

Documentación

```
npm i --save @nestjs/swagger
```

- main.ts

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

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

  app.setGlobalPrefix('api/v1')

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

  //SWAGGER
  const config = new DocumentBuilder()
  .setTitle('Teslo REST API')
  .setDescription('Teslo Shop')
  .setVersion('1.0')
  .build()

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

  await app.listen(process.env.PORT ?? 3000);
}
bootstrap();
```

Tags, ApiProperty, ApiResponse

- En el products.controller.ts

```
@ApiTags('Products')
@Controller('products')
export class ProductsController {
  {...code}
}
```

- Para indicar en el POST que tipo de data está esperando, si es obligatorio o no y que tipo de respuestas

```
@Post()
@ApiResponse({status: 201, description: 'Product was created', type: Product})
@ApiResponse({status: 400, description: 'Bad Request'})
@ApiResponse({status: 403, description: 'Forbidden. Token related'})
@Auth(ValidRoles.admin)
create(@Body() createProductDto: CreateProductDto,
        @GetUser() user: User) {
  return this.productsService.create(createProductDto, user);
}
```

- Debo ir a la entity y añadir **@ApiProperty**
- product.entity.ts

```
import { BeforeInsert, BeforeUpdate, Column, Entity, ManyToOne, OneToMany,
PrimaryGeneratedColumn } from "typeorm";
import { ProductImage } from "./product-image.entity";
import { User } from "src/auth/entities/user.entity";
import { ApiProperty } from "@nestjs/swagger";

@Entity({name: 'products'})
export class Product {

    @ApiProperty()
    @PrimaryGeneratedColumn('uuid')
    id: string

    @ApiProperty()
    @Column({
        type: 'text',
        unique: true
    })
    title: string

    @ApiProperty()
    @Column({
        type: 'float',
        default: 0
    })
    price: number

    @ApiProperty()
    @Column({
        type: 'text',
        nullable: true
    })
    description: string

    @ApiProperty()
    @Column({
        type: 'text',
        unique: true
```

```
    })
    slug: string

    @ApiProperty()
    @Column({
        type: 'int',
        default: 0
    })
    stock: number

    @ApiProperty()
    @Column({
        type: 'text',
        array: true
    })
    sizes: string[]

    @ApiProperty()
    @Column({
        type: 'text'
    })
    gender: string

    @ApiProperty()
    @Column({
        type: 'text',
        array: true,
        default: []
    })
    tags: string[]

    @ApiProperty()
    @OneToMany(
        ()=> ProductImage,
        productImage=> productImage.product,
        {cascade: true, eager: true}
    )
    images: ProductImage[]

    @ApiProperty()
    @ManyToOne(
        ()=> User,
        (user)=> user.product
    )
    user: User

    @BeforeInsert()
    checkSlugInsert(){
        if(!this.slug){
            this.slug = this.title //si no viene el slug guardo el titulo en slug
        }

        this.slug = this.slug
        .toLowerCase()
```

```
        .replaceAll(" ", "_")
        .replaceAll("'", "")
    }

    @BeforeUpdate()
    checkSlugUpdate(){
        this.slug= this.slug
        .toLowerCase()
        .replaceAll(" ", "_")
        .replaceAll(" ", "")
    }
}
```

Expandir ApiProperty

- product.entity.ts

```
@ApiProperty({
      example: '9a87a878s98-323ad4m4-3j3j3--Sa86ghs6rs', //uuid
      description: 'Product ID',
      uniqueItems: true
    })
    @PrimaryGeneratedColumn('uuid')
    id: string

@ApiProperty({
      example: 'T-Shirt Teslo',
      description: 'Product title',
      uniqueItems: true
    })
    @Column({
        type: 'text',
        unique: true
    })
    title: string
```

- También puedo añadir un **default en NULL o en 0, o con un arreglo vacío dentro de @ApiProperty**

Documentar DTOS

- Si los DTOS no fueran clases perderíamos la oportunidad de decorar las propiedades
- pagination.dto.ts

```
import { ApiProperty } from "@nestjs/swagger"
import { Type } from "class-transformer"
import { IsOptional, IsPositive, Min } from "class-validator"

export class PaginationDto{
```

```
    @ApiProperty({
        default: 10,
        description: 'How many rows do you need?'
    })
    @IsOptional()
    @IsPositive()
    @Type(()=> Number)
    limit?: number

    @ApiProperty({
        default: 0,
        description: 'How many rows do you want to skeep?'
    })
    @IsOptional()
    @Min(0)
    @Type(()=> Number)
    offset?: number
}
```

- Para tipar el create-product.dto sigo usando @ApiProperty
- Para tipar el update-prodcut dto **en lugar de importar PartialType de @nestjs/mapped-types lo importo de @nestjs/swagger**

NEST HERRERA - WEBSOCKETS

Introducción

- Crearemos una pequeña aplicación con Vite dónde introduciendo mi JsonWebToken me proporciona mi socket_id (identificación de la conexión) y nos conectaremos en tiempo real los dos backends
- Crearemos dos usuarios para las pruebas en la DB
- Los websockets no trabajan exactamente igual que la REST API
- Estamos sirviendo contenido estático en la carpeta public
- app.module.ts

```
{...imports}
import {ServeStaticModule} from '@nestjs/serve-static'
import { join } from 'path';


@Module({
  imports: [
    ConfigModule.forRoot(),
    TypeOrmModule.forRoot({
      type: 'postgres',
      host: process.env.DB_HOST,
      port: +process.env.DB_PORT!,
      database: process.env.DB_NAME,
      username: process.env.DB_USERNAME,
      password: process.env.DB_PASSWORD,
      autoLoadEntities: true,
      synchronize: true
    }),
    ProductsModule,
    CommonModule,
    SeedModule,
    FileModule,
    AuthModule,
    ServeStaticModule.forRoot({
      rootPath: join(__dirname, '..', 'public')
    })

  ],
  controllers: [],
  providers: [],
})
export class AppModule {}
```

- Conviene crear un index.html en public
- Guardo las imágenes en public/products
- Hay que habilitar el CORS para conectar el frontend
- Al estar el frontend dentro de la app en public no hay porqué habilitarlo

- Si tuviera que habilitarlo sería algo así
- main.ts

```
app.enableCors(options) //options es opcional
```

Websocket Gateways

- Usaremos socket.io
- Los websockets se implementan con necesidades específicas
- Permiten al servidor hablar de manera activa
    - Puede mandar la info tanto cliente-servidor como servidor cliente **sin que el cliente lo solicite**
    - Los dos están hablando y escuchándose
- El **@WebSocketGateway()** es **muy parecido al controlador**, hace una implementación que envuelve **socket.io o ws** (websocket)
- Funciona como un controlador, admite inyección de dependencias

> nest g res messagesWs --no-spec

- Elijo Websockets! No hacen falta los endpoints (aunque se puede hacer un CRUD con websockets)

> npm i @nestjs/websockets @nestjs/platform-socket.io

- Coloco el cors en true en el WebSocketGateway
- messages-ws.gateway.ts

```
import { WebSocketGateway} from '@nestjs/websockets';
import { MessagesWsService } from './messages-ws.service';


@WebSocketGateway({cors: true})
export class MessagesWsGateway {
  constructor(private readonly messagesWsService: MessagesWsService) {}

}
```

- Si hago una petición GET a

> localhost:3000/socket.io/socket.io.js

- Obtengo una respuesta muuuy larga de Socket.io

```
/*!
 * Socket.IO v4.8.1
 * (c) 2014-2024 Guillermo Rauch
 * Released under the MIT License.
 */
```

```
{...code}
```

- Este es el url que va a necesitar el cliente para conectarse

Server - Escuchar conexiones y desconexiones

- El servidor va a ser nuestra app de Nest
- El cliente va a ser una app web que se conecta al server
- La documentación de sockets está en socket.io
- El **namespace** podría verse como una sala de chat

```
@WebSocketGateway(80,{namespace: 'events'})
```

- Si no se especifica se apunta al namespace root '/'
- Cuando ingreso en el namespace tengo un nombre (que se puede repetir) y un id único
- Cada cliente se conecta a dos namespaces
    - El primero es como entrar en la casa, conectar con el server
    - También se conecta al namespace como una sala de chat que tiene el id de ese socket (único y volátil)
        - Cambia cada vez que el usuario se desconecta o se refresca el navegador
        - Me va a permitir mandarle un mensaje a esa persona si así lo deseo
- Cuando un cliente se conecta puedo reaccionar, quiero saber el id de ese cliente
- También quiero saber cuando un cliente se desconecta su id
- Para ello hay que **implementar 2 interfaces**
- Una vez indicadas me sitúo con el cursor encima de la clase y con **Ctrl .** las implemento
- Instalo socket.io para tener la interfaz de Socket

```
import { OnGatewayConnection, OnGatewayDisconnect, WebSocketGateway} from
'@nestjs/websockets';
import { MessagesWsService } from './messages-ws.service';
import { Socket } from 'socket.io';


@WebSocketGateway({cors: true})
export class MessagesWsGateway implements OnGatewayConnection,
OnGatewayDisconnect{

  constructor(private readonly messagesWsService: MessagesWsService) {}

handleConnection(client: Socket) {
    console.log('Cliente conectado', client.id); //este id es bien volátil
  }

  handleDisconnect(client: Socket) {
    console.log('Cliente desconectado', client.id)
```

```
  }
}
```

- Haremos una app puro Vanilla (sin framework)
- Hay cursos de React con socket.io

Cliente - Vite Vanilla Javascript

- Fuera de teslo-websockets, en la carpeta que lo alberga creo el proyecto con **npm create vite**
- Lo nombro ws-client, selecciono Vanilla - Typescript
- En ws-client hago un **npm i**
- Para correrlo

> npm run dev

- En el main está toda la mandanga
- Lo borro todo y dejo solo un div con un h1
- main.ts

```
import './style.css'

document.querySelector<HTMLDivElement>('#app')!.innerHTML = `
  <div>
    <h1>WebSocket Client</h1>

    <span><offline/span>
  </div>
`
```

- Necesito la url que mencioné anteriormente de socket.io para conectarme al servidor

> localhost:3000/socket.io/socket.io.js

- Creo un archivo dentro de src llamado socket-client.ts
- Para conectarme necesito instalar

> npm i socket.io-client

- socket.io-client debe hacer match o ser similar al socet.io de la respuesta del endpoint
- A este!

```
/*!
 * Socket.IO v4.8.1
 * (c) 2014-2024 Guillermo Rauch
 * Released under the MIT License.
 */
```

- Usualmente son las mismas
- Vamos con la conexión
- socket-client.ts

```
import {Manager} from 'socket.io-client'

export const connectToServer = ()=>{
    const manager = new Manager('localhost:3000/socket.io/socket.io.js')

    const socket = manager.socket('/') //conexión al namespace, esto es lo que
conecta al cliente

    //puedo usar un console.log para ver más info del socket
     console.log({socket})
}
```

- El cliente se conecta a través del Manager porque implementamos las dos interfaces en el Gateway
- Todos los clientes están conectados al servidor y es el servidor quien decide a quien le llega el mensaje
- No me conecto con otro cliente directamente
- Llamo a la función en el main.ts (estamos en el cliente)

```
import { connectToServer } from './socket-client'
import './style.css'

document.querySelector<HTMLDivElement>('#app')!.innerHTML = `
  <div>
    <h1>WebSocket Client</h1>

    <span>offline</span>

  </div>
`
connectToServer() //AQUI!
```

- Si voy al cliente (al navegador) y refresco varias veces la pantalla me indica Cliente conectado: "id"",Cliente desconectado: "id" en la consola del server que está corriendo
- Necesito saber los clientes conectados

Server - Mantener identificados los clientes

- Cuando el cliente se conecta voy a llamar a ConnectedClients con el id apuntando al socket (client)
- Creo un getter en el servicio para obtener el número de clientes conectados
- messages-ws.service.ts

```
import { Injectable } from '@nestjs/common';
import { Socket } from 'socket.io';

interface ConnectedClients{
    [id: string]: Socket
}


@Injectable()
export class MessagesWsService {
     private connectedClients: ConnectedClients = {}

     handleConnection(client: Socket){
        this.connectedClients[client.id] = client

        console.log({conectados: this.getConnectedClients()})
     }

     handleDisconnect(clientId: string){
        delete this.connectedClients[clientId]
     }

     getConnectedClients(): number{
        return Object.keys(this.connectedClients).length
     }

}
```

- En el Gateway llamo al servicio

```
import { OnGatewayConnection, OnGatewayDisconnect, WebSocketGateway} from
'@nestjs/websockets';
import { MessagesWsService } from './messages-ws.service';
import { Socket } from 'socket.io';


@WebSocketGateway({cors: true})
export class MessagesWsGateway implements OnGatewayConnection,
OnGatewayDisconnect{

  constructor(private readonly messagesWsService: MessagesWsService) {}

handleConnection(client: Socket) {
  this.messagesWsService.handleConnection(client)
}

  handleDisconnect(client: Socket) {
    this.messagesWsService.handleDisconnect(client.id)
  }
}
```

- Creo un getter en el servicio para obtener el número de clientes conectados

```
getConnectedClients(): number{
      return Object.keys(this.connectedClients).length
    }
```

- Si inicio el cliente, lo abro en el navegador y miro la consola del server corriendo aparece **{conectados: 1}**

Cliente - Detectar conexión y desconexión

- En el main.ts del cliente creo un id para el span, para podere acceder a él fácilmente

```
import { connectToServer } from './socket-client'
import './style.css'

document.querySelector<HTMLDivElement>('#app')!.innerHTML = `
  <div>
    <h1>WebSocket Client</h1>

    <span id="server-status">offline</span>

  </div>
`
connectToServer()
```

- Para saber si estoy conectado o no me valgo de unos eventos del mismo socket
  - Si quiero escuchar uso .on
  - Si quiero hablar con el servidor uso .emit
- En socket-client.ts (del cliente)

```
import {Manager, Socket} from 'socket.io-client'

export const connectToServer = ()=>{
    const manager = new Manager('localhost:3000/socket.io/socket.io.js')

    const socket = manager.socket('/') //conexión al namespace

    addListeners(socket)
}


const addListeners = (socket: Socket) =>{
    const serverStatusLabel = document.querySelector('#server-status')!

    socket.on('connect', ()=>{
```

```
        serverStatusLabel.innerHTML = 'connected'
    })

    socket.on('disconnect', ()=>{
        serverStatusLabel.innerHTML = 'disconnected'

    })

}
```

Cliente - Clientes conectados

- Añado un order list en el cliente para mostrar todos los clientes conectados
- En el main.ts

```
import { connectToServer } from './socket-client'
import './style.css'

document.querySelector<HTMLDivElement>('#app')!.innerHTML = `
  <div>
    <h1>WebSocket Client</h1>

    <span id="server-status">offline</span>

    <ul id='clients-ul'>
      <li> ID_CLIENTE </li>
    </ul>

  </div>
`
connectToServer()
```

- En el server, cuando un cliente se conecta quiero mandar una notificación a todos los clientes, que un nuevo cliente se conectó
- Para eso necesito la instancia del WebSocketServer, la obtengo con el decorador @WebSocketServer
- Este WebSocketServer tiene la info de todos los clientes conectados
- Uso .emit, le paso el nombre del evento (que puede ser cualquiera) y el Payload que puede ser un objeto, un boolean, puede ser cualquier cosa
- También queremos saber cuando algún cliente se desconecta
- message-ws.gateway.ts

```
import { OnGatewayConnection, OnGatewayDisconnect, WebSocketGateway,
WebSocketServer} from '@nestjs/websockets';
import { MessagesWsService } from './messages-ws.service';
import { Server, Socket } from 'socket.io';
```

```
@WebSocketGateway({cors: true})
export class MessagesWsGateway implements OnGatewayConnection,
OnGatewayDisconnect{

  @WebSocketServer() wss: Server

 constructor(private readonly messagesWsService: MessagesWsService) {}

handleConnection(client: Socket) {
  this.messagesWsService.handleConnection(client)

  this.wss.emit('clients-updated',this.messagesWsService.getConnectedClients() )
}

  handleDisconnect(client: Socket) {
    this.messagesWsService.handleDisconnect(client.id)
    this.wss.emit('clients-updated',this.messagesWsService.getConnectedClients() )
  }
}
```

- Solo quiero los ids de los clientes conectados, basado en el método que creeé en el messages-ws.service
- messages-ws.service.ts

```
getConnectedClients(): string[]{
      return Object.keys(this.connectedClients)
    }
```

- En los listeners del cliente (en socket-client.ts)ç

```
const addListeners = (socket: Socket) =>{
    const serverStatusLabel = document.querySelector('#server-status')!

    socket.on('connect', ()=>{
        serverStatusLabel.innerHTML = 'connected'
    })

    socket.on('disconnect', ()=>{
        serverStatusLabel.innerHTML = 'disconnected'

    })

    socket.on('clients-updated', (clients: string[])=>{
        console.log({clients})
    })
}
```

- En la consola del navegador veo Array(n) con los ids de los clientes conectados

- En el socket-client.ts obtengo el ul y le paso los ids conectados

```
import {Manager, Socket} from 'socket.io-client'

export const connectToServer = ()=>{
    const manager = new Manager('localhost:3000/socket.io/socket.io.js')

    const socket = manager.socket('/') //conexión al namespace

    addListeners(socket)
}


const addListeners = (socket: Socket) =>{
    const serverStatusLabel = document.querySelector('#server-status')!
    const clientsUL= document.querySelector('#clients-ul')!

    socket.on('connect', ()=>{
        serverStatusLabel.innerHTML = 'connected'
    })

    socket.on('disconnect', ()=>{
        serverStatusLabel.innerHTML = 'disconnected'

    })

    socket.on('clients-updated', (clients: string[])=>{
        let clientsHTML=''

        clients.forEach(clientId=>{
            clientsHTML += `
            <li>${clientId}</li>`
        })

        clientsUL.innerHTML = clientsHTML
    })

}
```

- Ahora en pantalla aparecen los ids conectados

Emitir Cliente - Escuchar Servidor (chat)

- Quiero enviar un mensaje desde el cliente y que lo escuchen los demás
- Creo un formulario en el HTML del main.ts del cliente

```
import { connectToServer } from './socket-client'
import './style.css'

document.querySelector<HTMLDivElement>('#app')!.innerHTML = `
```

```
  <div>
    <h1>WebSocket Client</h1>

    <span id="server-status">offline</span>

    <ul id='clients-ul'>
      <li> ID_CLIENTE </li>
    </ul>

    <form id="message-form">
      <input placeholder="message" id="message-input" />
    </form>

  </div>
`
connectToServer()
```

- En socket-client.ts capturo el form y el message a través del id

```
import {Manager, Socket} from 'socket.io-client'

export const connectToServer = ()=>{
    const manager = new Manager('localhost:3000/socket.io/socket.io.js')

    const socket = manager.socket('/') //conexión al namespace

    addListeners(socket)
}


const addListeners = (socket: Socket) =>{
    const serverStatusLabel = document.querySelector('#server-status')!
    const clientsUL= document.querySelector('#clients-ul')!
    const messageForm = document.querySelector<HTMLFormElement>('#message-form')!
    const messageInput = document.querySelector<HTMLInputElement>('#message-
input')!

    socket.on('connect', ()=>{
        serverStatusLabel.innerHTML = 'connected'
    })

    socket.on('disconnect', ()=>{
        serverStatusLabel.innerHTML = 'disconnected'

    })

    socket.on('clients-updated', (clients: string[])=>{
        let clientsHTML=''

        clients.forEach(clientId=>{
            clientsHTML += `
            <li>${clientId}</li>`
```

```
        })

        clientsUL.innerHTML = clientsHTML
    })

    //voy a estar escuchando el evento submit del formulario (cuando alguien le de
al intro)
    messageForm.addEventListener('submit', (event)=>{
        event.preventDefault()
        if(messageInput.value.trim().length <= 0) return

        console.log({id: "YO!", message: messageInput.value})//este objeto es el
que quiero mandarle al servidor
    })
}
```

- Agrego unos estilos al input en el styles.css

```
input{
  padding: 10px 20px;
}
```

- Emito para el server el evento message-from-client

```
messageForm.addEventListener('submit', (event)=>{
    event.preventDefault()
    if(messageInput.value.trim().length <= 0) return

    socket.emit('message-from-client', {
        id: 'YO!',
        message: messageInput.value
    })

    messageInput.value =''
})
```

- Hay que poner el server a escuchar a este evento
- Nest ofrece una forma genial con el decorador **@SubscribeMessage**. Estoy en el server, en el messages-ws.gateway
- Usando este decorador siempre voy a tener disponible el cliente y el payload
- messages-ws.gateway.ts

```
 @SubscribeMessage('message-from-client')
  onMessageFromClient(client: Socket, payload: any){

  }
```

- Creo un dto messages-ws/dtos/new-message.dto.ts. Pongamos que solo me interesa el message, descarto el id

```
import { IsString, MinLength } from "class-validator";

export class NewMessageDto{
    @IsString()
    @MinLength(1)
    message: string
}
```

- Lo uso en el gateway, ahora puedo tipar el payload
- messages-ws.gateway.ts

```
@SubscribeMessage('message-from-client')
  onMessageFromClient(client: Socket, payload: NewMessageDto){
    console.log(client.id, payload)
  }
```

- Me devuelve esto por consola (del server)

```
730gRc8H8VUqzROgAAAB { id: 'YO!', message: 'oihoij' }
```

- El string del SubscribeMessage debe de ser único, si no ejecutará el primero que encuentre

Formas de emitir desde el servidor

- Los clientes están emitiendo pero no pasa nada
- Pongamos que queremos imprimir en pantalla los mensajes de los clientes conectados
- En el main.ts del cliente creo otro ul

```
import { connectToServer } from './socket-client'
import './style.css'

document.querySelector<HTMLDivElement>('#app')!.innerHTML = `
  <div>
    <h1>WebSocket Client</h1>

    <span id="server-status">offline</span>

    <ul id='clients-ul'>
      <li> ID_CLIENTE </li>
    </ul>

    <form id="message-form">
      <input placeholder="message" id="message-input" />
    </form>
```

```
    <h3>Messages</h3>
    <ul id="messages-ul"></ul>

  </div>
`
connectToServer()
```

- En el socket-client.ts capturo el message-ul y pongo a escuchar el server con el texto message-from-server

```
const addListeners = (socket: Socket) =>{
    const serverStatusLabel = document.querySelector('#server-status')!
    const clientsUL= document.querySelector('#clients-ul')!
    const messageForm = document.querySelector<HTMLFormElement>('#message-form')!
    const messageInput = document.querySelector<HTMLInputElement>('#message-
input')!
    const messageUl = document.querySelector<HTMLUListElement>('#messages-ul')!
//capturo el message-ul

    socket.on('connect', ()=>{
        serverStatusLabel.innerHTML = 'connected'
    })

    socket.on('disconnect', ()=>{
        serverStatusLabel.innerHTML = 'disconnected'

    })

    socket.on('clients-updated', (clients: string[])=>{
        let clientsHTML=''

        clients.forEach(clientId=>{
            clientsHTML += `
            <li>${clientId}</li>`
        })

        clientsUL.innerHTML = clientsHTML
    })

    //voy a estar escuchando el evento submit del formulario (cuando alguien le de
al intro)
    messageForm.addEventListener('submit', (event)=>{
        event.preventDefault()
        if(messageInput.value.trim().length <= 0) return

        socket.emit('message-from-client', {
            id: 'YO!',
            message: messageInput.value
        })
```

```
        messageInput.value =''
    })

    //Llamo al evento message-from-server
    socket.on('message-from-server', (payload: {fullName: string, message:
string})=>{
        console.log(payload)
    })
}
```

- Pongamos que solo quiero emitir a la persona que mandó el mensaje
- messages-ws.gateway.ts

```
@SubscribeMessage('message-from-client')
onMessageFromClient(client: Socket, payload: NewMessageDto){

  //emite únicamente al cliente que mandó el mensaje, no a todos
  client.emit('message-from-server', {
    fullName: 'Soy yo!',
    message: payload.message || 'no hay mensaje'
  })
}
```

- De esta manera solo el cliente que envió el mensaje recibe el mensaje que envió
- Si quiero emitir a todos menos al cliente que emite el mensaje (que es el client:Socket que recibe como parámetro el método)

```
@SubscribeMessage('message-from-client')
  onMessageFromClient(client: Socket, payload: NewMessageDto){

    //client.emit('message-from-server', {
      //fullName: 'Soy yo!',
      //message: payload.message || 'no hay mensaje'
    //})

    //Emitir a todos menos al cliente que emitió
    client.broadcast.emit('message-from-server', {
      fullName: 'Soy yo!',
      message: payload.message || 'no hay mensaje'
    })
  }
```

- Para mandar un mensaje inlcusive al que emitió el mensaje uso el WebSocketServer con this.wss.emit

```
import { OnGatewayConnection, OnGatewayDisconnect, SubscribeMessage,
WebSocketGateway, WebSocketServer} from '@nestjs/websockets';
import { MessagesWsService } from './messages-ws.service';
```

```
import { Server, Socket } from 'socket.io';
import { NewMessageDto } from './dtos/new-message.dto';


@WebSocketGateway({cors: true})
export class MessagesWsGateway implements OnGatewayConnection,
OnGatewayDisconnect{

  @WebSocketServer() wss: Server

 constructor(private readonly messagesWsService: MessagesWsService) {}

handleConnection(client: Socket) {
  this.messagesWsService.handleConnection(client)

  this.wss.emit('clients-updated',this.messagesWsService.getConnectedClients() )
}

 handleDisconnect(client: Socket) {
    this.messagesWsService.handleDisconnect(client.id)

    this.wss.emit('clients-updated', this.messagesWsService.getConnectedClients())
  }

  @SubscribeMessage('message-from-client')
  onMessageFromClient(client: Socket, payload: NewMessageDto){

    //client.emit('message-from-server', {
      //fullName: 'Soy yo!',
      //message: payload.message || 'no hay mensaje'
    //})

    //Emitir a todos menos al cliente que emitió
    //client.broadcast.emit('message-from-server', {
      //fullName: 'Soy yo!',
      //message: payload.message || 'no hay mensaje'
    //})

    //Mensaje a todos inlcuyendo al que emitió el mensaje
    this.wss.emit('message-from-server', {
      fullName: 'Soy yo!',
      message: payload.message || 'no hay mensaje'
    })
  }
}
```

- También tengo el this.wss.to al que le puedo pasar una sala (un ClientId)
- Con client.join() puedo añadir al usuario a una sala
- Todos los clientes están conectados a una sala que es su client.id
- Puedo usar el email como identificador de sala con client.join(user.email)
- Si quiero emitir a todos los clientes que estén en la sala de ventas

> this.wss.to('ventas').emit('lo que sea')

- Para mostrar en pantalla el mensaje voy al socket-client.ts

```
const addListeners = (socket: Socket) =>{
    const serverStatusLabel = document.querySelector('#server-status')!
    const clientsUL= document.querySelector('#clients-ul')!
    const messageForm = document.querySelector<HTMLFormElement>('#message-form')!
    const messageInput = document.querySelector<HTMLInputElement>('#message-
input')!
    const messageUl = document.querySelector<HTMLUListElement>('#messages-ul')!

    socket.on('connect', ()=>{
        serverStatusLabel.innerHTML = 'connected'
    })

    socket.on('disconnect', ()=>{
        serverStatusLabel.innerHTML = 'disconnected'

    })

    socket.on('clients-updated', (clients: string[])=>{
        let clientsHTML=''

        clients.forEach(clientId=>{
            clientsHTML += `
            <li>${clientId}</li>`
        })

        clientsUL.innerHTML = clientsHTML
    })

    //voy a estar escuchando el evento submit del formulario (cuando alguien le de
al intro)
    messageForm.addEventListener('submit', (event)=>{
        event.preventDefault()
        if(messageInput.value.trim().length <= 0) return

        socket.emit('message-from-client', {
            id: 'YO!',
            message: messageInput.value
        })

        messageInput.value =''
    })

    socket.on('message-from-server', (payload: {fullName: string, message:
string})=>{

        const newMessage = `
            <li>
                <strong>${payload.fullName}</strong>
                <span>${payload.message}</span>
```

```
            </li>
        `
        const li = document.createElement('li') //creo el elemento
        li.innerHTML = newMessage //relleno el elemento

        messageUl.append(li) //agrego el elemento
    })
}
```

Preparar cliente para enviar JWT

- Quiero usar mi sistema de autenticación con JWT para evaluar si el cliente tiene o no la autenticación que yo espero
- Si no la tiene no le voy a dejar conectarse a mi servicio de webSockets (ws)
- Usaremos el mismo jwt de una petición POST hacia su endpoint, regresa el jwt, lo alamacena en el localstorage/sessionStorage a la hora de conectarnos
- En lugar de manejarlo con el localStorage, para que no de problemas con las diferentes instancias de Chrome usaremos un inputen el HTML que va aser el usuario que se está autenticando ahi
- Creo un input y un botón
- main.ts

```
import { connectToServer } from './socket-client'
import './style.css'

document.querySelector<HTMLDivElement>('#app')!.innerHTML = `
  <div>
    <h2>WebSocket Client</h2>

    <input  id="jwt-token" placeholder="Json Web Token"/>
    <button id="btn-connect">Connect</button>

    <br/>
    <span id="server-status">offline</span>

    <ul id='clients-ul'>
      <li> ID_CLIENTE </li>
    </ul>

    <form id="message-form">
      <input placeholder="message" id="message-input" />
    </form>

    <h3>Messages</h3>
    <ul id="messages-ul"></ul>

  </div>
`
const jwtToken = document.querySelector<HTMLInputElement>('#jwt-token')!
const btnConnect = document.querySelector<HTMLButtonElement>('#btn-connect')!
```

```
btnConnect.addEventListener('click', ()=>{
  connectToServer()
})
```

- De esta manera estoy llamando a la conexión manualmente
- Ahora tomaré el jwt y validaré que venga antes de conectarme
- Se podría usar una expresión regular para asegurarme que es un jwt, ya que de esta manera solo evalúa si hay algo en el input
- main.ts

```
const jwtToken = document.querySelector<HTMLInputElement>('#jwt-token')!
const btnConnect = document.querySelector<HTMLButtonElement>('#btn-connect')!

btnConnect.addEventListener('click', ()=>{

  if(jwtToken.value.trim().length <= 0) return alert('Enter a valid JWT')
  connectToServer(jwtToken.value.trim())
})
```

- connectToServer ahora recibe el jwt
- Puedo añadir cierta info adicional al manager
- socket-client.ts

```
import {Manager, Socket} from 'socket.io-client'

export const connectToServer = (token: string)=>{
    const manager = new Manager('localhost:3000/socket.io/socket.io.js', {
        extraHeaders:{
            authentication: token
        }
    })

    const socket = manager.socket('/') //conexión al namespace

    addListeners(socket)
}

const addListeners = {...code}
```

- En el server añado un console.log con el cliente en handleConnection
- message-ws.gateway.ts

```
handleConnection(client: Socket) {
  console.log(client)
  this.messagesWsService.handleConnection(client)
```

```
    this.wss.emit('clients-updated',this.messagesWsService.getConnectedClients() )
  }
```

- Me devuelve un objeto enorme en la consola del server
- En la parte del handshake encontramos los headers el host, el tipo de conexión y tenemos el authentication, que es el extraHeader que yo añadí
- Si yo quisiera obtener el authentication y mostrarlo en consola haríamos algo así

```
handleConnection(client: Socket) {
  const token = client.handshake.headers.authentication as string
  console.log({token})
  this.messagesWsService.handleConnection(client)

  this.wss.emit('clients-updated',this.messagesWsService.getConnectedClients() )
}
```

Validar JWT del Handshake

- Hay que verificar que el jwt sea mio, que esté firmado por mi, etc
- Para crear el jwt en Nest usábamos el JwtService
- Necesitamos el .verify (.verifyAsync) del servicio
- Inyecto el JwtService en el Gateway (podría inyectarlo en el servicio)
- messages-ws.gateway.ts

```
@WebSocketGateway({cors: true})
export class MessagesWsGateway implements OnGatewayConnection,
OnGatewayDisconnect{

  @WebSocketServer() wss: Server

 constructor(
  private readonly messagesWsService: MessagesWsService,
  private readonly jwtService: JwtService
) {}

{...code}
}
```

- Debo importar el AuthModule (en el que exporto el JwtModule junto con el TypeOrmModule y otros) en el messages-ws.module
- Esto me servirá para usar la entidad de User pronto

```
import { Module } from '@nestjs/common';
import { MessagesWsService } from './messages-ws.service';
import { MessagesWsGateway } from './messages-ws.gateway';
import { JwtModule } from '@nestjs/jwt';
```

```
@Module({
  imports: [AuthModule],
  providers: [MessagesWsGateway, MessagesWsService],
})
export class MessagesWsModule {}
```

- En el gateway verifico el token en un try catch
- En lugar de usar HttpException se usa **throw new WsException()**
- En lugar de usar la excepción desconectaré al cliente directamente en el catch
- messages-ws.gateway.ts

```
handleConnection(client: Socket) {
  const token = client.handshake.headers.authentication as string

  let payload : JwtPayloadInterface = {id: ''}

  try {
    payload = this.jwtService.verify(token)

  } catch (error) {
    client.disconnect()
  }
  console.log({payload})

  this.messagesWsService.handleConnection(client)

  this.wss.emit('clients-updated',this.messagesWsService.getConnectedClients() )
}
```

- En la consola del server aparece esto

```
{
  payload: {
    id: 'bcb907c1-74e2-4e28-9300-5aa4cf580fd3',
    iat: 1748900331,
    exp: 1748907531
  }
}
```

Enlazar Socket con usuario

- Queremos mostrar en pantalla el usuario que escribe el mensaje
- Le paso el id del cliente que viene en el payload a **registerClient**

```
handleConnection(client: Socket) {
  const token = client.handshake.headers.authentication as string

  let payload : JwtPayloadInterface = {id: ''}

  try {
    payload = this.jwtService.verify(token)
    this.messagesWsService.handleConnection(client, payload.id)

  } catch (error) {
    client.disconnect()
  }



  this.wss.emit('clients-updated',this.messagesWsService.getConnectedClients() )
}
```

- **NOTA:** cambio los nombres de los métodos del servicio de messages-ws
    - handleConnection ahora se llama **registerClient**
    - handleDisconnect ahora se llama **removeClient**
- Para pasarle el usuario a connectedClients, puedo decir en la interfaz que el id del Socket apunte a un objeto con el socket y un user de tipo User
- messages-ws.service.ts

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Socket } from 'socket.io';
import { User } from 'src/auth/entities/user.entity';
import { Repository } from 'typeorm';

interface ConnectedClients{
    [id: string]:{
      socket: Socket,
      user: User
    }
}


@Injectable()
export class MessagesWsService {

   constructor(
      @InjectRepository(User)
      private readonly userRepository: Repository<User>
   ){}
```

```
      private connectedClients: ConnectedClients = {}

      async registerClient(client: Socket, userId: string ){

       const user = await this.userRepository.findOneBy({id: userId})

       if(!user) throw new Error('User not found')
       if(!user.isActive) throw new Error('User not active')

         this.connectedClients[client.id] = {
          socket: client,
          user
         }

      }

      removeClient(clientId: string){
         delete this.connectedClients[clientId]
      }

      getConnectedClients(): string[]{
         return Object.keys(this.connectedClients)
      }

}
```

- Debo colocar el async y await al handleConnection del gateway

```
async handleConnection(client: Socket) {
  const token = client.handshake.headers.authentication as string

  let payload : JwtPayloadInterface = {id: ''}

  try {
    payload = this.jwtService.verify(token)
    await this.messagesWsService.registerClient(client, payload.id)

  } catch (error) {
    client.disconnect()
  }

  this.wss.emit('clients-updated',this.messagesWsService.getConnectedClients() )
}
```

- Vamos a mostrar el usuario en cada mensaje que escriba
- El Soy yo! que se está mostrando está en el gateway, al final

```
import { OnGatewayConnection, OnGatewayDisconnect, SubscribeMessage,
```

```
WebSocketGateway, WebSocketServer} from '@nestjs/websockets';
import { MessagesWsService } from './messages-ws.service';
import { Server, Socket } from 'socket.io';
import { NewMessageDto } from './dtos/new-message.dto';
import { JwtService } from '@nestjs/jwt';
import { JwtPayloadInterface } from 'src/auth/interfaces/jwt-payload.interface';


@WebSocketGateway({cors: true})
export class MessagesWsGateway implements OnGatewayConnection,
OnGatewayDisconnect{

  @WebSocketServer() wss: Server

 constructor(
  private readonly messagesWsService: MessagesWsService,
  private readonly jwtService: JwtService
) {}

async handleConnection(client: Socket) {
  const token = client.handshake.headers.authentication as string

  let payload : JwtPayloadInterface = {id: ''}

  try {
    payload = this.jwtService.verify(token)
    await this.messagesWsService.registerClient(client, payload.id)

  } catch (error) {
    client.disconnect()
  }



  this.wss.emit('clients-updated',this.messagesWsService.getConnectedClients() )
}

 handleDisconnect(client: Socket) {
    this.messagesWsService.removeClient(client.id)

    this.wss.emit('clients-updated', this.messagesWsService.getConnectedClients())
  }

  @SubscribeMessage('message-from-client')
  onMessageFromClient(client: Socket, payload: NewMessageDto){

    //client.emit('message-from-server', {
      //fullName: 'Soy yo!',
      //message: payload.message || 'no hay mensaje'
    //})

    //Emitir a todos menos al cliente que emitió
    //client.broadcast.emit('message-from-server', {
      //fullName: 'Soy yo!',
```

```
      //message: payload.message || 'no hay mensaje'
    //})

    //Mensaje a todos incluyendo al que emitió el mensaje
    this.wss.emit('message-from-server', {
      fullName: 'Soy yo!', //AQUI!<----------------------------------------
      message: payload.message || 'no hay mensaje'
    })
  }
}
```

- Creo un método en el messages-ws.service

```
getUserFullNameBySocketId(socketId: string){
      return this.connectedClients[socketId].user.fullName
     }
```

- Uso el servicio en el @SubscribeMessage del messages-ws.gateway

```
@SubscribeMessage('message-from-client')
onMessageFromClient(client: Socket, payload: NewMessageDto){

  //Mensaje a todos incluyendo al que emitió el mensaje
  this.wss.emit('message-from-server', {
    fullName: this.messagesWsService.getUserFullNameBySocketId(client.id),
    message: payload.message || 'no hay mensaje'
  })
}
```

- No estamos cerrando la conexión, por lo que si me conecto varias veces con el mismo token aparecen varias conexiones
- Debo evaluar que si existe un usuario conectado no se vuelva a conectar

Desconectar usuarios duplicados

- Puede ser que yo quiera que el comportamiento sea que un mismo usuario pueda crear varias conexiones activas
- Pero vamos a hacer que solo un usuario pueda tener una conexión activa
- Podría usar booleanos como desktop y mobile para permitir conexión desde escritorio y movil

```
interface ConnectedClients{
    [id: string]:{
      socket: Socket,
      user: User,
      desktop: boolean,
      mobile: boolean
```

```
    }
}
```

- Pero vamos a hacer que un usuario solo pueda tener una conexión activa
- Creo un método en el messages-ws.service.ts

```
checkUserConnection(user: User){
    for(const clientId of Object.keys(this.connectedClients)){
        const connectedClient = this.connectedClients[clientId]

        if(connectedClient.user.id === user.id){
          //desconecto el socket anterior
          connectedClient.socket.disconnect()
          break;
        }
    }
    }
```

- Llamo a checkUserConnection antes de realizar la conexión con registerClient en el service

```
async registerClient(client: Socket, userId: string ){

      const user = await this.userRepository.findOneBy({id: userId})

      if(!user) throw new Error('User not found')
      if(!user.isActive) throw new Error('User not active')

         this.checkUserConnection(user)

        this.connectedClients[client.id] = {
         socket: client,
         user
        }

     }
```

- Ahora funciona pero en pantalla al intentar conectar con el mismo token solo tengo una conexión activa pero pone discconect
- Algo está mal en el front
- Cada vez que toco el botón estoy creando un nuevo socket y crea nuevos listeners pero todos los listeners anteriores siguen existiendo
- Estamos creando listeners por todos lados
- Puedo usar **socket.removeAllListeners()**
- Pero esto no resuelve el problema, el socket anterior sigue en memoria y no lo estoy limpiando
- Por eso declaro socket fuera de la función

- Cada vez que mandaba a llamar a connectToServer creaba nuevos listeners, y los anteriores no eran eliminados
- socket-client.ts

```
import {Manager, Socket} from 'socket.io-client'

let socket: Socket

export const connectToServer = (token: string)=>{
    const manager = new Manager('localhost:3000/socket.io/socket.io.js', {
        extraHeaders:{
            authentication: token
        }
    })

    socket?.removeAllListeners()
    socket = manager.socket('/') //conexión al namespace

    addListeners(socket)
}
```

- Ahora hace bien lo de que cuando me vuelvo a conectar aparece connected y el nuevo id pero no los mensajes que escribo
- Es porque cuando estamos emitiendo usa el Socket que encontró en el contexto de addListeners, que es el viejo socket
- Ya no hace falta que le pase el socket a addListeners porque usamos el socket que está de manera global

```
import {Manager, Socket} from 'socket.io-client'

let socket: Socket

export const connectToServer = (token: string)=>{
    const manager = new Manager('localhost:3000/socket.io/socket.io.js', {
        extraHeaders:{
            authentication: token
        }
    })

    socket?.removeAllListeners()
    socket = manager.socket('/') //conexión al namespace

    addListeners()
}


const addListeners = () =>{
    const serverStatusLabel = document.querySelector('#server-status')!
    const clientsUL= document.querySelector('#clients-ul')!
```

```
    const messageForm = document.querySelector<HTMLFormElement>('#message-form')!
    const messageInput = document.querySelector<HTMLInputElement>('#message-
input')!
    const messageUl = document.querySelector<HTMLUListElement>('#messages-ul')!

    socket.on('connect', ()=>{
        serverStatusLabel.innerHTML = 'connected'
    })

    socket.on('disconnect', ()=>{
        serverStatusLabel.innerHTML = 'disconnected'

    })

    socket.on('clients-updated', (clients: string[])=>{
        let clientsHTML=''

        clients.forEach(clientId=>{
            clientsHTML += `
            <li>${clientId}</li>`
        })

        clientsUL.innerHTML = clientsHTML
    })

    //voy a estar escuchando el evento submit del formulario (cuando alguien le de
al intro)
    messageForm.addEventListener('submit', (event)=>{
        event.preventDefault()
        if(messageInput.value.trim().length <= 0) return

        socket.emit('message-from-client', {
            id: 'YO!',
            message: messageInput.value
        })

        messageInput.value =''
    })

    socket.on('message-from-server', (payload: {fullName: string, message:
string})=>{
        const newMessage = `
            <li>
                <strong>${payload.fullName}</strong>
                <span>${payload.message}</span>
            </li>
        `
        const li = document.createElement('li')
        li.innerHTML = newMessage

        messageUl.append(li)
    })
}
```