08 NEST Relaciones TypeORm

- En esta sección vamos a ver como el cliente a la hora de crear va a ser capaz de mandarme los url de mis imágenes
- Voy a almacenar estas imágenes en una carpeta del filesystme de mis servidor que no va a ser pública
- Apunto a requerir autenticación: no va a ver esta imagen si no está autenticado o si no es el usuario
- Estas imágenes las voy a estar almacenando en una nueva tabla
- Les voy a asignar un id único
- Más adelante se hará un servicio para servir estas imágenes que podamos autenticar y validar
- Cuando envío la actualización, las imágenes que hayan sido guardadas previamente serán eliminadas

---

ProductImage Entity

- No voy a crear un CRUD completo para las imágenes de los productos. No quiero que funcione así
- En src/products/entitys creo product-image.entity
- No digo que la url puede ser null, por lo que tiene que venir

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

@Entity()
export class ProductImage{

    @PrimaryGeneratedColumn() //va a tener un número autoincremental como id
    id: number

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

- Debo indicar que existe esta entidad, aunque tenga el synchronize en true, por ahora solo he creado un archivo con una clase
- 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({
  controllers: [ProductsController],
  providers: [ProductsService],
  imports: [
    TypeOrmModule.forFeature([
```

```
      Product, ProductImage
    ])
  ]
})
export class ProductsModule {}
```

- Ahora debería aparecer en TablePlus
- Como importo las dos entidades desde el mismo lugar me creo un archivo de barril
- La forma fácil es hacer los imports y cambiar la palabra import por export
- index.ts

```
export { ProductImage } from "./product-image.entity";
export { Product } from "./product.entity";
```

- Puedo importarlas así en el módulo

```
import { Product, ProductImage } from './entities';
```

- Cómo hacemos una relación de uno a muchos?
- Necesito una nueva columna en product-image que me diga el id del producto al cual pertenece

---

OneToMany y ManyToOne

- Tengo que decirle a mi entity Product que ahora va a tener las imágenes
- Y a la entity imágenes que va a tener un id producto
- Pongo que las images son opcionales y su relación es OneToMany (lo importo de typeORM) ya que un producto puede tener varias imágenes
  - El callback va a regresar un ProductImage
  - productImage (con minúscula), todavía no aparece .product (pero si .id y .url)
  - Le coloco en las opciones el cascade en true. Esto va ayudar a que si hago una elminación, elimine las imagenes asociadas al producto
  - Usualmente no se borraría, si no que estaría activo inactivo
- product.entity

```
import {Entity, PrimaryGeneratedColumn, Column, BeforeInsert, BeforeUpdate,
OneToMany} from 'typeorm'
import { ProductImage } from './product-image.entity'

@Entity()
export class Product {
    @PrimaryGeneratedColumn('uuid')
    id: string

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

```
    title: string

    @Column('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

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

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

    @BeforeInsert()
    checkSlugInsert(){
        if(!this.slug){
            this.slug = this.title
```

```
        }

        this.slug = this.slug
        .toLowerCase()
        .replaceAll(' ', '_')
        .replaceAll("'", "")
    }

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

}
```

- En ProductImage voy a tener un product de tipo Product
- Muchas imágenes pueden tener un único producto, por lo que la relación es ManyToOne
- Muchos registros de mi tabla de ProductImage pueden ter un único producto
- El primer callback devuelve algo de TipoProduct
- El segundo callback le paso el product y lo asocio con product.image
- En opciones

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

@Entity()
export class ProductImage{

    @PrimaryGeneratedColumn() //va a tener un número autoincremental como id
    id: number

    @Column('text')
    url: string

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

Crear imághenes de producto

- Debo añadir images al create-product.dto para que no me de error

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

export class CreateProductDto {

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

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

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

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

    @IsInt()
    @IsPositive()
    @IsOptional()
    stock?: number

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

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

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

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

- Salta un error en consola:

```
    Types of property 'images' are incompatible.
      Type 'string[]' is not assignable to type 'DeepPartial<ProductImage>'.

     const product = await this.productRepository.preload({
                                                          ~
```

```
    id,
  ~~~~~~~~~
    ...updateProductDto
  ~~~~~~~~~~~~~~~~~~~~~~~
  })
```

- Si voy al productService veo que tengo un problema en el create y en el update
- Temporalmente voy a añadir como un arreglo vacío de imágenes en el create y el update.
- Este error es porque debo asegurarme de que las imágenes son una instancia de ProductImage
- Porqué? Porque ahí tienen el productId, el url, el id y no simplemente son strings como le estoy mandando en el body
- Entonces tengo que hacer algún tipo de conversión
- product.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';
import { PaginationDto } from 'src/common/dto/pagination.dto';
import { validate as isUUID } from 'uuid';


@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, images:
[]}) //añado como arreglo vacío images

      await this.productRepository.save(product)

      return product

    } catch (error) {

      this.handleDBExceptions(error)
    }

  }
```

```
  async findAll(paginationDto:PaginationDto) {

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

    return await this.productRepository.find({
      take: limit,
      skip: offset
      //TODO: relaciones
    })
  }

  async findOne(term: string) {

    let product: Product

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

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

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

  async update(id: string, updateProductDto: UpdateProductDto) {
    const product = await this.productRepository.preload({
      id,
      ...updateProductDto,
      images: []      //añado como arreglo vacío images
    })

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

    try {
      await this.productRepository.save(product)
      return product
    } catch (error) {
      this.handleDBExceptions(error)
    }
  }

  async remove(id: string) {

    const product = await this.findOne(id)

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

```
  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')
  }
}
```

- Extraigo las imágenes con desestructuración del createProductDto, le asigno un arreglo vacío por defecto por si no viene ninguna
- Extraigo el resto de propiedades con el operador **...**
- Para generar las instancias de las imágenes voy a necesitar inyectar el repositorio
- Hago un map, que devuelve un arreglo. Debo indicarle que la url es la imagen
- TypeORM va ainferir por mi, dice:
    - Ah! estás queriendo crear instancias de ProductImage que se encuentran dentro del producto
    - Cuando grabe este nuevo producto, typeORM va a saber que el id que le asigne a este producto va a ser el que le asigne a cada una de las imágenes

```
  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)
    }

  }
```

- Ahora apunto al endpoint y en el body le mando las url de las fotos

> POST http://localhost:3000/api/products/

```
{
  "title": "Migue's shoes",
  "sizes": ["SM", "M", "L"],
  "gender": "men",
  "price": 300,
  "tags":["shoes", "Migue"],
  "images": [
    "http://image1.jpg",
    "http://image2.jpg"
    ]
}
```

- Ahora las imágenes tienen una url y un id
- Me gustaría que estas imágenes que tienen una url y una id me las devuelva solo como un arreglo de url para el frontend
- Lo hago en el return, usando las images que estoy recibiendo para retornarlas tal cual

```
  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, images}

    } catch (error) {

      this.handleDBExceptions(error)
    }

  }
```

Aplanar las imágenes

- En product.entity tengo la relación pero **no tengo la columna en la db**
- Con lo que en la tabla de productos no me aparecen las imágenes relacionadas
- En el findAll del servicio establezco la relación con **relations**

```
async findAll(paginationDto:PaginationDto) {
```

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

  return await this.productRepository.find({
    take: limit,
    skip: offset,
    relations: {
      images: true
    }
  })
}
```

- Quiero aplanar las imágenes, que solo me devuelva la url
- El id me puede servir para hacer la eliminación y el update pero de momento lo quiero así
- Yo se que el .find va a regresar un arreglo de Product, lo guardo en una constante

```
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(product =>({
    ...product,
    images: product.images.map(img=> img.url)
  }))
}
```

- Puedo mejorar el código

```
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)
  }))
}
```

- En el .findOneBy tengo el mismo problema, pero en este no puedo especificar la condición de relations
- En la documentación de typeORM, en Eager relations, veo que solo funcionan con el .find
- Cuando se usa el QueryBuilder están deshabilitadas y hay que usar el leftJoinAndSelect
- En el OneToMany puedo especificar el eager en true
- product.entity

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

- Ahora cuando busco por id me aparecen las imágenes
- Pero si busco por el slug no aparecen, porque estoy usando el QueryBuilder
- Uso el leftJoinAndSelect, debo especificar cual es la relación
- Puedo agregarle un alias a la tabla de producto, en este caso prod
- Debo indicarle el punto en el cual quiero hacer el leftJoin, me pide que le ponga un alias en caso de que quiera hacer otros joins, le pongo prodImages

```
async findOne(term: string) {

  let product: Product

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

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

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

- Podría usar el mismo método de devolver el ...product, images: images.map, etc

    - Pero no lo voy a manejar así, porque me interesa deviolver una instancia de mi entiad, y no algo que luzca como tal
- Voy a crear un nuevo método que me sirva para aplanarlo

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

Query Runner

- Vamos a trabajar la actualización de un producto
- Si actualizo un producto y no le paso las imágenes (y el producto tiene imágenes) aparece el arreglo de imágenes vacío y en la tabla de imágenes he perdido la referencia al producto
- Esto sucede porque tengo el **cascade en true**
- También porque cuando estoy haciendo el **update le estoy diciendo que images es un arreglo vacio**
- **Borro todas las imagenes de la db**
- Quiero que las imágenes que añado en el body sean las nuevas imágenes y borrar las anteriores
- Entonces, son dos cosas que quiero hacer: borrar las anteriores e insertar las nuevas
- Si una de las dos falla quiero revertir el proceso. Para ellos usaré el **Query Runner**
- Desestructuro del dto las imagenes por separado
- Debo verificar si vienen imágenes o no. Si vienen imágenes voy a tener que borrar las anteriores de una manera controlada
- El queryRunner tiene que conocer **la cadena de conexión** que estoy usando
- Para ello usaré la inyección de depndencias con elc **DataSoure** (lo importo de typeORM)
- Uso **createQueryRunner** (el createQueryBuilder serviría si lo creara desde cero sin ninguna entidad)
- Con el queryRunner voy a empezar a **definir una serie de procedimientos**

```
  constructor(
    @InjectRepository(Product)
    private readonly productRepository: Repository<Product>,

    @InjectRepository(ProductImage)
    private readonly productImageRepository: Repository<ProductImage>,

    private readonly dataSource: DataSource

    ){}

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()


    try {
      await this.productRepository.save(product)
      return product
    } catch (error) {
      this.handleDBExceptions(error)
    }
  }
```

Transacciones

- Vamos a usar el queryRunner para crear **una transacción**
- Son una serie de querys que pueden impactar la db, hasta que no le haga el commit no lo hará
- Hay que liberar el queryRunner porque si no mantiene esa conexión
- Tengo que evaluar si hay imágenes en el dto para borrar las existentes. Así es como quiero que funcione
- El soft delete mantiene la referencia al objeto, no lo borra. El delete si
- Como primer parámetro recibe la entity, y de segundo los criterios, en este caso en la columna de producto, el id que me están pasando para actualizar
- Entonces, voy a borrar todas las images cuya columna product sea el id
- Coloco product y no productId como aparece en la tabla porque la columna de productId es una relación y me lo permite
- Creo el queryRunner fuera del try para poder usar el rollback en el catch
- Para que me devuelva las imágenes cuando no vienen en el body del update uso findOnePlain
    - Podría hacerlo como la línea de código del else pero tendría que volver a formatear la salida de las url de las imágenes

```
  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}}) //con esto
borramos las imágenes anteriores

       product.images= images.map(image=> this.productImageRepository.create({url:
image}))

      }else{
          //product.images = await this.productImageRepository.findBy({product:
{id}}) puedo hacerlo así pero usaré findOnePlain
      }

      await queryRunner.manager.save(product)

      await queryRunner.commitTransaction()  //commit
      await queryRunner.release() //desconexión

      return this.findOnePlane( id )

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

      this.handleDBExceptions(error)
    }
  }
```

- Si hago un update sin imágenes respeta las imágenes que había.
- Si añado la misma imágen le asigna un nuevo id

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
- Hay varias formas de resolver este problema
  - Una de ellas es crear una transacción dónde primero borraría las imágenes y luego el producto
- También puedo decirle que al borrar un producto se borren las imágenes relacionadas
- Eliminar en cascada: cuando se afecta una tabla se afectan las demás relacionadas
- En product-image.entity no tengo definido que quiero que suceda en esta tabla si se borra el producto

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

@Entity()
export class ProductImage{

    @PrimaryGeneratedColumn() //va a tener un número autoincremental como id
    id: number
```

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

    @ManyToOne(
        ()=> Product,
        product => product.images,
        {onDelete: 'CASCADE'} //le indico que borre en cascada
    )
    product: Product
}
```

- Creo un método en el servicio para borrar todos los productos
- Creo un queryBuilder y le pongo el alias de product
- Borro con query.delete en el where no le pongo nada para que borre todo, y lo ejecuto

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

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

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

Product Seed

- Copio del gist de Herrera la data

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

- Mi objetivo es apuntar a un endpoint que borre todos los registros anteriores e inserte estos

> nest g res seed --no-spec

- Borro dtos, entitys, todos los endpoints del controlador menos el GET, lo mismo en el servicio
- Nombro el GET como executeSeed (lo mismo en el servicio)
- controller

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

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

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

}
```

- service

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


@Injectable()
export class SeedService {

  async runSeed() {
    return `SEED EXECUTED`;
  }

}
```

- Para borrar los productos necesito acceso al servicio para usar el metodo que creé para borrar todos los productos
- Para ello exporto el servicio en el módulo de Products con exports
- Puedo exportar también el TypeOrmModule si quisiera trabajar con los repositorios

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


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

- E importo el módulo en el módulo de seed con imports

```
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 {}
```

- Creo un método privado en el servicio para encapsular la eliminación de los productos y lo ejecuto dentro del servicio

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


@Injectable()
export class SeedService {

  constructor(
    private readonly productsService: ProductsService
  ){}

  async runSeed() {

    this.insertNewProducts()

    return `SEED EXECUTED`;
  }

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

}
```

- Esto es lo necesario para la eliminación

Insertar de forma masiva

- En seed creo un directorio llamado data con data-seed.ts con la data a insertar
- data-seed.ts

```
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[];
}


export const initialData: SeedData = {
    products: [
        {
            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'
        },
        {
            description: "The Men's Quilted Shirt Jacket features a uniquely fit, quilted design for warmth and mobility in cold weather seasons. With an overall street-smart aesthetic, the jacket features subtle silicone injected Tesla logos below the back collar and on the right sleeve, as well as custom matte metal zipper pulls. Made from 87% nylon and 13% polyurethane.",
            images: [
                '1740507-00-A_0_2000.jpg',
                '1740507-00-A_1.jpg',
            ],
```

```
            stock: 5,
            price: 200,
            sizes: ['XS','S','M','XL','XXL'],
            slug: "men_quilted_shirt_jacket",
            type: 'shirts',
            tags: ['jacket'],
            title: "Men's Quilted Shirt Jacket",
            gender: 'men'
        }
     (etc..)
    ]
}
```

- Se podría hacer un insertMany pero tendríamos que insertar el modelo, hacer el repositorio, pero esto es algo que se va a ejecutar una sola vez
- En lugar de eso voy a llamar al método create del productService pasándole algo que luzca como el dto y ejecute el mismo procedimiento
- Guardo los productos del initialData en products con initialData.products
- La interfaz SeedProduct luce muy parecida al dto de create-product.dto

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


@Injectable()
export class SeedService {

  constructor(
    private readonly productsService: ProductsService
  ){}

  async runSeed() {

    this.insertNewProducts()

    const products = initialData.products

    const insertPromises = []

    products.forEach(product=>{
      insertPromises.push(this.productsService.create(product)) //create devuelve
una promesa. Las inserto en el arreglo
    })

    await Promise.all(insertPromises) //Ejecuto todas las promesas

    //Si quisiera el resultado de cada una de esas promesas podría guardar el
Promise.all en una constante results

    //const results = await Promise.all(insertPromises)
```

```
    return `SEED EXECUTED`;
  }

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

}
```

- Añado el seed al README

```
# Teslo API

1. Clonar proyecto
2. ```npm install```
3. Clonar el archivo ```.env.template``` y renombrarlo a ```.env```
4. Cambiar las variables de entorno
5. Levantar la db
```
docker-compose up -d
```
6. Ejecutar SEED
```
localhost:3000/api/seed
```
7. Levantar ```npm run start:dev```
```

Renombrar tablas

- Las tablas, en lugar de llamarse product y product_image deberían llamarse products y product_images
- Voy a las entidades, primero a Product
- Si abro llaves en el decorador Entity puedo ver que hay varias opciones
    - database, engine, name, orderBy, schema, synchronize, etc
    - En este caso me interesa el name

```
import {Entity, PrimaryGeneratedColumn, Column, BeforeInsert, BeforeUpdate,
OneToMany} from 'typeorm'
import { ProductImage } from './product-image.entity'

@Entity({name: 'products'})
export class Product {
    @PrimaryGeneratedColumn('uuid')
    id: string

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

```
    title: string

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

- Ahora aparecen las tablas products, product, product_image y product_images
- Las borro todas y hago el seed