NEST 12

Product Image Entity

- Quiero crear una nueva tabla llamada ProductImage
- Cuando yo vaya a crear un nuevo producto quiero que esté el arreglo de imagenes con sus url's
- Cuando esté trabajando en la actualización de un producto, si me manda un nuevo arreglo de imagenes voy a borrar el arreglo anterior
- Así es como quiero que funcione
- Creo el archivo product-image.entity.ts en /src/entities
- Cada imagen con un identificador único, para que así si borro una imagen duplicada en diferentes productos no borre las dos (o las que sean)
- El url tiene que venir
- Importante, el decorador @Entity!!

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

@Entity()
export class ProductImage{

    @PrimaryGeneratedColumn()
    id: number

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

- Por el synchronize debería de haber creado la tabla automaticamente, pero yo tengo que decirle al ORM que tengo una nueva entidad
- Lo hago en el product.module.ts

```
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 {}
```

- Necesito una tercera columna que me diga a que producto (id) pertenece la fotografía
- Así cuando haga la petición a través de un query " tráeme todas las imagenes del producto (id)"

OneToMany y ManyToOne

- Voy a product.entity
- A producto voy a tener que decirle que tiene imágenes, y a imágenes habrá que decirle que tienen la id de un producto
- Tengo que decirle al @OneToMany cómo se va a conectar
- El productImage.product no existe todavía.
- El cascade: true me va a ayudar, por ejemplo en la eliminación de un producto, va a eliminar las imágenes que estén asociadas al producto
- Hay que definir que será un arreglo

```
import { ParseUUIDPipe } from "@nestjs/common";
import { ReplaySubject } from "rxjs";
import { BeforeInsert, BeforeUpdate, Column, Entity, OneToMany,
PrimaryGeneratedColumn } 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('int', {
        default: 0
    })
    stock: number

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

    @Column('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()
            .replace(' ','_')
            .replace("'", '')
    }

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

    }
}
```

- Ahora en product-image.entity:

- Un montón de imágenes puede tener un único producto. esa relación es @ManyToOne

```
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
    )
    product: Product
}
```

Crear Imágenes de Producto

- Voy a asegurarme de que las url's de images se inserten en la tabla con una petición POST en ThunderClient

> localhost:3000/api/products

- ene l body de la petición

```
{
  "title": "Juancho",
  "sizes": ["M", "L"],
  "gender": "men",
  "price": 123,
  "images": [
    "http://image1.jpg",
    "http://image1.jpg"
    ]
}
```

- Esto no lo acepta porque en mi dto no está especificado que acepte images

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

```
export class CreateProductDto {

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

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

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

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

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

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

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

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

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

}
```

- Esto da un error en la terminal. Dice:
  -Types of property 'images' are incompatible. Type 'string[]' is not assignable to type 'DeepPartial'.

```
 const product = await this.productRepository.preload({
                                                       ~
69       id: id,
```

70 ...updateProductDto

```
71     })
~~~~~
~~~

- Parece que el error está en el update-products.dto
- Vayamos al product.service
- Añado en el update el images: []
- En el create me marca otro error en el try en el momento de crear con el
createProductDto.
 - Abro llaves, uso el spread y le añado el images: []

~~~ts
import { BadRequestException, Injectable, InternalServerErrorException, Logger,
NotFoundException } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { PaginationDto } from 'src/common/dto/pagination.dto';
import { Repository } from 'typeorm';
import { CreateProductDto } from './dto/create-product.dto';
import { UpdateProductDto } from './dto/update-product.dto';
import { Product } from './entities/product.entity';
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: []
   })
   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)){
   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: id,
   ...updateProductDto,
   images: []
 })

 if(!product) throw new NotFoundException('This product does not exists')

 try {
   await this.productRepository.save(product)
   return product

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

 }
}

async remove(id: string) {
 const product= await this.productRepository.delete(id)
 if(!product) throw new NotFoundException('The product does not exists')
```

```
 return("Product deleted")
}

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')
}
}
~~~

- Un arreglo vacío en create es como que no tenga ninguna imagen creada
- Pero si voy a tener imágenes tengo que asegurarme de que esas imagenes tienen
que ser instancias de la entidad ProductImage
 - Porqué? Porque ahí tienen el id, la url, etc. y no son simples strings
- Puedo desestructurar las imágenes y asignarle un arreglo vacío por defecto, y
desestructurar el resto en productDetails
- products.service:

~~~ts
async create(createProductDto: CreateProductDto) {

 const {images=[], ...productDetails} = createProductDto;

 try {
   const product= this.productRepository.create({
     ...productDetails,
     images: []
   })
   await this.productRepository.save(product)

   return product

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

- Para crear las instancias voy a necesitar inyectar otro repositorio en el
constructor
- products.service:

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

 @InjectRepository(ProductImage)
 private readonly productImageRepository: Repository<ProductImage>
){}
~~~
```

- Hago un .map de las imagenes y uso el repositorio. Lo que le tengo que proporcionar es el url
- products.service

~~~ts
async create(createProductDto: CreateProductDto) {

 const {images=[], ...productDetails} = createProductDto;

 try {
   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)
 }
}
~~~

- También tengo que pasarle producto, pero como lo estoy creando dentro de la creación del producto TypeORM va a inferir por mi.
 - Va a decir "ah! estás queriendo crear instancias de product.images". Cuando yo grabe esto va a saber que el id que le asigne a este producto va a ser el id que va a tener cada una de esas imágenes
- Ahora si voy a ThunderClient y hago el Post lo crea
- Quiero manejar el return de otra manera
- products.service

~~~ts
async create(createProductDto: CreateProductDto) {

 const {images=[], ...productDetails} = createProductDto;

 try {
   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)
 }
}
~~~

- Si hago un get a todos los productos no devuelve las imágenes. Resolvámoslo

Aplanar las imágenes

- En productos no hay ninguna columna de imagenes, por lo que en la respuesta del Get no las muestra
- Tengo una relación, no una columna
- Voy al metodo de findAll del servicio y añado relations:

~~~ts
async findAll(paginationDto: PaginationDto) {

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

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

}
~~~
- Esto así funciona, pero yo quiero aplanar las imagenes
- Yo se que el find va a regresar una colección de producto
- Puedo hacer esto:
~~~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( product => ({
   ...product,
   images: product.images.map(image=> image.url)
 }))
}
~~~
- Ahora recibo solo las url en la respuesta
- En el findOne tengo el mismo problema con las imagenes, no las retorna. Pero tal y como está configurado no puedo usar el relations
- En la documentación es Eager Relations:
 - eager relations are loaded automatically each time you load entities from the db
- Cualquier find va a funcionar
- Hay ciertas restricciones. Si estoy usando el QueryBuilder eager está

````
deshabilitado  y hay que usar leftJoinAndSelect to load the relation
- Voy a la entidad de product y junto a cascade: true pongo el eager en true

~~~ts
import { ParseUUIDPipe } from "@nestjs/common";
import { ReplaySubject } from "rxjs";
import { BeforeInsert, BeforeUpdate, Column, Entity, OneToMany,
PrimaryGeneratedColumn } 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('int', {
     default: 0
 })
 stock: number

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

 @Column('text')
 gender: string

 @Column({
     type: 'text',
````

```
    array: true,
    default: []
})
tags: string[]

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

@BeforeInsert()
checkSlugInsert(){
    if ( !this.slug){
        this.slug= this.title
    }
    this.slug= this.slug
        .toLowerCase()
        .replace(' ','_')
        .replace("'", '')
}

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

}
}
~~~


- Si busco por el slug no aparecen las imagenes
- Si me fijo, en findOne estoy usando el QueryBuilder. Hay que usar
leftJoinAndSelect y especificar la relación
 - Hay que especificar cómo se van a  relacionar. La tabla
 - En este caso le quiero poner un alias a la tabla, en el QueryBuilder,cómo
estamos en productRepository, le pongo prod
 - me pide un alias para este join como segundo parámetro

~~~ts
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
}
~~~

- Ahora ya aparecen las imágenes
- De igual manera estaría bien aplanar las imágenes
- Puedo crear un método para hacerlo
- .service

~~~ts
async findOnePlain(term: string){
 const {images=[], ...rest} = await this.findOne(term)

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

}
~~~

- Ahora lo cambio en el controller
- .controller:

~~~ts
@Get(':term')
findOne(@Param('term') term: string) {
 return this.productsService.findOnePlain(term);
}
~~~

## Query Runner
-----
- Ahora se trabajará la actualización de un producto
- Si hago un PATCH a un id  con imágenes, y no mando imágenes, las imágenes las regresa en un arreglo vacío
- Si voy a TablePlus perdí la referencia ( productid ) de esas imágenes en la tabla de product_image, pero las imágenes siguen ahi
- Es porque tengo una actualización en cascada y estoy perdiendo la info
- Cuando hago el update, le estoy pasando el arreglo vacío, con lo cual este producto no tiene ninguna imagen

~~~ts
async update(id: string, updateProductDto: UpdateProductDto) {
```

```
 const product = await this.productRepository.preload({
   id: id,
   ...updateProductDto,
   images: []
 })

 if(!product) throw new NotFoundException('This product does not exists')

 try {
   await this.productRepository.save(product)
   return product

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

 }
}
~~~

- Puedo borrar todas las imágenes, ya que está establecida la relación muchos a uno
- Si intento borrar un producto que tiene imágenes va a rebentar porque hay un CONSTRAIN ahi puesto
- Hay que corregir estos problemas
- Si quiero agregar urls de imágenes desde el PATCH no las agrega. Lo que quiero es agregarlas y borrar todas las anteriores
- Son dos transacciones: la de eliminación que tiene que salir bien, y la de actualización.
- Si una de las dos falla debo poder hacer la inversión de la transacción y mandarle un mensaje al usuario indicándole qué sucedió
- El **QueryRunner** me va a permitir ejecutar X cantidad de instrucciones SQL y decirle "si todo sale bien, impacta la base de datos. Pero si no, puedo havcer el roll back y revertirlo todo"
- Todo va a empezar con el preload del producto
- Desestructuro las imágenes y el resto con el parámetro REST
- El queryRunnner es un objeto que entre otras cosas tiene que conocer la cadena de conexión con la base de datos
- Para ello debo inyectar el DataSource, lo importo del ORM
- Al inyectarlo de esta manera sabe cual es la cadena de conexión, sabe el usuario, tiene la misma configuración que mi repositorio
- Usaré ese DataSource para

~~~ts
import { BadRequestException, Inject, Injectable, InternalServerErrorException, Logger, NotFoundException } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { PaginationDto } from 'src/common/dto/pagination.dto';
import { DataSource, Repository } from 'typeorm';
import { CreateProductDto } from './dto/create-product.dto';
import { UpdateProductDto } from './dto/update-product.dto';
import { Product } from './entities/product.entity';
import {validate as isUUID} from 'uuid'
import { ProductImage } from './entities/product-image.entity';
```

```
@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>,


 private readonly dataSource: DataSource
){}

async create(createProductDto: CreateProductDto) {

 const {images=[], ...productDetails} = createProductDto;

 try {
   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)
 }
}

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(image=> image.url)
 }))
}

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
}

async findOnePlain(term: string){
 const {images=[], ...rest} = await this.findOne(term)

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

}

async update(id: string, updateProductDto: UpdateProductDto) {

const {images, ...toUpdate} = updateProductDto



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

 if(!product) throw new NotFoundException('This product does not exists')

 const queryRunner=this.dataSource.createQueryRunner()

 try {
   await this.productRepository.save(product)
   return product

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

 }
```

```
}

async remove(id: string) {
 const product= await this.productRepository.delete(id)
 if(!product) throw new NotFoundException('The product does not exists')

 return("Product deleted")
}

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')
}
}
~~~

- En el queryRunner necesita definir una serie de procedimientos
- No va a hacer los commits hasta que se ejecute el commit ( me comprometo a que esto es lo que se va a grabar en la db )
- Si no ejecuto el commit y pretendo hacer la actualización con el mismo nombre o el mismo slug de otro producto ni siquiera va a llegar ahí
- Puede ser que anteriormente cuando lance ese error ya se hicieran otras transacciones con la DB
- Pero si fallo con la modificación de alguna de esas transacciones voy a querer revertirlo y eso se conoce como un ROLL BACK
- El ROLL BACK es lo opuesto del commit

## Transacciones
-----
- Entiendase transacción cómo una serie de QUERYS que pueden impactar la DB
- Pueden insertar, eliminar, etc
- Debo decir: quiero hacer el commit de esa transacción, que puede implicar un montón de querys
- Debo asegurarme del commit o el ROLL BACK, y también liberar el queryRunner porque si no hace una de esas dos mantiene la conexión
- Le indico que conecte el queryRunner y que empiece la transacción
- product.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('This product does not exists')
```

```
 const queryRunner=this.dataSource.createQueryRunner()

 await queryRunner.connect()
 await queryRunner.startTransaction()

 try {



   await this.productRepository.save(product)
   return product

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

 }
}
~~~
```

- Si en la desestructuración de updateProductoDto vienen imñagenes quiere decir que hay que borrar las anteriores (de haberlas) porque así he decidido que sea, no es una norma
- Lo gestiono en el try
- Que borre, en la columna de producto del ProductImage, el que sea igual al id (del producto)
- Voy a borrar todas las ProductImages cuya columna product sea igual al id del producto
- Al ser una relación, aunque la columna se llame productId se puede poner de esta manera
- Luego creo una instancia de las nuevas imágenes y lo salvo con el queryRunner
- Puede ser que alguna de las dos transacciones falle y quiera revertir el cambio
- El queryRunner no está impactando en la DB todavía

```
~~~ts
async update(id: string, updateProductDto: UpdateProductDto) {

const {images, ...toUpdate} = updateProductDto



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

 if(!product) throw new NotFoundException('This product does not exists')

 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) //esto puede fallar

   //await this.productRepository.save(product)
   return product

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

 }
}
```

~~~

- Falta hacer el commit, decirle a la transacción si no has dado error hasta este punto, guarda los cambios
- Y luego como ya no necesito más el queryRunner uso el release
- Eso es con:

~~~ts
await queryRunner.commitTransaction()
await queryRunner.release()
~~~

- Lo coloco antes del return product
- Pero qué pasa cuando no hay imágenes nuevas? Tendría que cargar las imágenes de alguna manera
- Antes del this.handleDBExceptions (porque si no no se ejecuta) coloco el rollback
- Por eso creé el queryRunner fuera del try, para poder usar esto en el catch

~~~ts
async update(id: string, updateProductDto: UpdateProductDto) {

const {images, ...toUpdate} = updateProductDto



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

 if(!product) throw new NotFoundException('This product does not exists')

 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) //esto puede fallar

   await queryRunner.commitTransaction()
   await queryRunner.release()

   //await this.productRepository.save(product)
   return product

 } catch (error) {

   await queryRunner.rollbackTransaction()
   await queryRunner.release()

   this.handleDBExceptions(error)

 }
}
~~~

- Si no mando imágenes nuevas no tengo las imágenes en el PATCH. tengo que
cargarlas porque el preload no me carga mis relaciones
- Una manera de hacerlo sería cambiar el return product por

> return this.findOnePlain(id)

- Otra forma sería en el else con product.images:

~~~ts
async update(id: string, updateProductDto: UpdateProductDto) {

const {images, ...toUpdate} = updateProductDto



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

 if(!product) throw new NotFoundException('This product does not exists')

 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 }))

   }else{
     product.images = await this.productImageRepository.findBy({product: {id}})
   }

   await queryRunner.manager.save(product) //esto puede fallar

   await queryRunner.commitTransaction()
   await queryRunner.release()

   //await this.productRepository.save(product)
   return product

 } catch (error) {

   await queryRunner.rollbackTransaction()
   await queryRunner.release()

   this.handleDBExceptions(error)

 }
}
~~~

- De todas maneras, la consulta a la DB se hace igual, mejor reutilzar el código y
devolver las url planas
- Puede ser que yo tenga un mismo producto que comparta imágenes pero no quiero
que comparta la url, por eso lo manejo así

Eliminación en cascada

- Si intento borrar un producto que tiene imagen/es me da error 500

> Update or delete on table "product" violates foreign key constraint
"OIUJonuo8by7098B?)" on table "poduct_image"
- Se podría solucionar iniciando una transacción, borrar primero las imágenes y
luego borrar el producto
- También puedo decirle que cuando borre un producto se borren las imagenes
relacionadas en cascada
- En la entity no he especificado que quiero que suceda cuando se borren las
imñagenes
- 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 service para borrar toda la DB y la de imágenes también
- product.service:

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

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

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

- Voy a usar este deleteAllProducts cuando trabaje mi semilla. Es algo que sólo quiero para desarrollo

Product SEED

- Copio el seed que me ofrece el profesor; no lo copio aquí porque es muy largo.
- Si copio la interface, las constantes y el primer objeto

~~~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'
     }
 ]
~~~

- Voy a generar el seed

> nest g res seed --no-spec

- Borro los dtos, borro la entity, borro todo el controlador y dejo solo una
petición Get. Lo mismo en el seed.service
- seed.controller:

~~~ts
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();
}

}
~~~

- seed.service:

~~~ts
import { Injectable } from '@nestjs/common';


@Injectable()
export class SeedService {

async  runSeed() {
 return 'SEED EXECUTED'
}


}
~~~

- Compruebo en ThunderClient que funcione correctamente y ver el mensaje seed
executed
- Cómo hago para borrar los productos? Necesito acceso al servicio de productos.
Está en otro módulo
- Exporto el ProductsService. Es bien común exportar el TypeOrmModule
- products.module:

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

~~~

- Importo el módulo en seed.module

~~~ts
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 {}
~~~

- Ahora puedo inyectar el servicio
- Llamo al método para borrar toda la DB

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


@Injectable()
export class SeedService {

constructor(
 private readonly productsService: ProductsService
){}

async  runSeed() {
 await this.insertNewProducts()
 return 'SEED EXECUTED'
}

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

}
~~~

Insertar de forma masiva

- Creo una carpeta en /seed lamada data/ con un arcivo que se llama seed con la data
- Voy a centrarme en insertar la data
- Voy a mandar llamar el método de create

~~~ts

```
async create(createProductDto: CreateProductDto) {

 const {images=[], ...productDetails} = createProductDto;

 try {
   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)
 }
}
~~~

- Voy a tener que pasarle algo que luce muy parecido al Dto
- Guardo los productos en una variable
- El product del forEach es muy parecido al Dto, mirar la interface. Lo que dice
typescript es que debe de lucir como el dto
- this.productsService.create(product) es una promesa (si dejas el cursor encima
lo dice)
- La inserto en el arreglo de insertPromises
- Promise.all es de javascript
- esto está diciendo: espera a que todas estas promesas se resuelvan y cuando se
resuelvan sigue con la siguiente línea

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


@Injectable()
export class SeedService {

constructor(
 private readonly productsService: ProductsService
){}

async  runSeed() {
 await this.insertNewProducts()
 return 'SEED EXECUTED'
}

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

 const products = initialData.products

 const insertPromises = []
```

```
 products.forEach(product=>{
   insertPromises.push(this.productsService.create( product ))
 })
 await Promise.all( insertPromises)
}

}
~~~

- Le doy a la petición Get con el endpoint de seed y carga los productos con sus imágenes

Renombrar tablas

- Puedo renombrar las tablas desde entities

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

 @PrimaryGeneratedColumn('uuid')
 id: string
 (...)
~~~

- Lo mismo con product-image.entity (le cambio el nombre a "products_images")
- Ahora si voy a TablePlus tengo las nuevas tablas vacías y las antiguas con la data
- Las borro desde Table Plus
- Le doy al seed y salta un error 500, en consola dice: relation "products" does not exists
- Una solución es borrar el contenedor de docker y volvera levantar la imagen

> docker-compose up -d

- Subo el servidor y hago el seed. VOILÀ!
```