09 NEST Carga de archivos

- A través de un POST con el nombre de la imagen (UUID) en la url voy a mostrar la fotografía
- Cuando quiero subir una imagen, selecciono el archivo y me devuelve la url que voy a poder utilizar para el frontend

---

Subir un archivo al backend

- Instalo estos tipos

```
npm i -D @types/multer
```

- La carga de archivos es general, por lo que tendrá su propio módulo

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

- No voy a necesitar ni el dto ni la entity
- Borro todos los endpoints del controller y los métodos del servicio
- La carga de archivos se hará mediante una petición POST
- Le añado el endpoint product
- Recibirá el archivo de tipo **Express** (no hace falta importarlo)

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


@Controller('files')
export class FilesController {
  constructor(private readonly filesService: FilesService) {}

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

- Para enviar un archivo desde POSTMAN o ThunderClient, 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 un decorador, igual que necesito @Body o @Query
- En este caso es **@UploadedFile**
- Necesita saber el nombre de la llave, para esto vamos a usar un **interceptor**
- Los interceptores interceptan las solicitudes y también pueden interceptar y mutar las respuestas
- Dentro de **@UseInterceptors** uso **FileInterceptor** de nest/platform-express
- Debo indicarle el **nombre de la key** que le haya puesto, en este caso file

```
import { Controller, Get, Post, Body, Patch, Param, Delete, UploadedFile,
UseInterceptors } from '@nestjs/common';
import { FilesService } from './files.service';
import { FileInterceptor } from '@nestjs/platform-express';


@Controller('files')
export class FilesController {
  constructor(private readonly filesService: FilesService) {}

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

- Por defecto Nest sube el archivo a una carpeta temporal
- No se recomienda guardar el archivo en el filesystem por razones de seguridad
- Usar un servicio de terceros como Cloudinary

Validar archivos

- Cambio el valor del return (devolvía el file), para que me devuelva solo el nombre
- Quiero validar que no me suban un pdf, sólo imágenes
- Cómo esta validación es una tarea común se podría poner en el common
- Lo coloco en /files, ya que es algo que solo voy a usar en este módulo, creo la carpeta helpers
- Dentro creo el fileFilter.helper.ts
- Para poder usarlo en el FileInterceptor debo darle un aspecto característico
- Si coloco unas llaves después de 'file' dentro del FileInterceptor, y escribo fileFilter, la ayuda de Typescript me dice que tiene 3 argumentos
    - La request
    - El file
    - Callback, que tiene como argumentos el error y un boolean llamado acceptFile
- Esta función no regresa nada (void)
- Entonces, debo hacer la función fileFilter con estos 3 argumentos
- No hace falta que importe de express el request y demás porqué ya viene en Nest

```
export const fileFilter=(req: Express.Request, file: Express.Multer.File,
callback:Function )=>{

    console.log({file})

    callback(null, true)

}
```

- Se lo paso al FileInterceptor

```
import { Controller, Get, Post, Body, Patch, Param, Delete, UploadedFile,
UseInterceptors } from '@nestjs/common';
import { FilesService } from './files.service';
import { FileInterceptor } from '@nestjs/platform-express';
import { fileFilter } from './helpers/fileFilter.helper';


@Controller('files')
export class FilesController {
  constructor(private readonly filesService: FilesService) {}

  @Post('product')
  @UseInterceptors(FileInterceptor('file', {
    fileFilter: fileFilter //no lo estoy ejecutando, solo le paso la referencia
  }))
  uploadProductFile(@UploadedFile() file: Express.Multer.File){
    return {
      fileName: file.originalname
    }
  }
}
```

- Si hago el posteo veo en consola la info

```
{
  file: {
    fieldname: 'file',
    originalname: '01NEST_PRIMEROSPASOS.md',
    encoding: '7bit',
    mimetype: 'text/markdown'
  }
}
```

- Si el archivo no existe lanzo un error y le paso un false, que es el boolean diciendo que no aceptó el archivo
- Si el archivo existe, uso el split para dividirlo por / y me quedo con la segunda posición
- Creo un arreglo con las extensiones válidas
- Hago la comparación y devuelvo el calback con un true si el file es una imagen válida, si no con un false

```
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)

}
```

- Esto no va alanzar una excepción por parte de Nest, solo me va a validar si el archivo es permitido o no
- Creo un badRequestException si no viene el file

```
import { Controller, Get, Post, Body, Patch, Param, Delete, UploadedFile,
UseInterceptors, BadRequestException } from '@nestjs/common';
import { FilesService } from './files.service';
import { FileInterceptor } from '@nestjs/platform-express';
import { fileFilter } from './helpers/fileFilter.helper';


@Controller('files')
export class FilesController {
  constructor(private readonly filesService: FilesService) {}

  @Post('product')
  @UseInterceptors(FileInterceptor('file', {
    fileFilter: fileFilter
  }))
  uploadProductFile(@UploadedFile() file: Express.Multer.File){

    if(!file) throw new BadRequestException('Make sure that the file is an image')

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

- Podría poner la lista de extensiones en variables de entorno para poder expandir rapidamente la funcionalidad
- Ahora vamos a guardar fisicamente la imagen en el filesystem

---

Guardar imagen en filesystem

- Recuerdo que **no es recomendable guardar los archivos en el filesystem** en la vida real
- Lo recomendable es usar un servicio de terceros
- Yo podría crear la carpeta public/products y guardar las imágenes ahí
- Pero el problema es que **cualquier persona autenticada o no va a poder verlo**, porque es público
- En lugar de nombrarla public, la nombro static

- Dentro creo las carpetas uploads y products. Subiré los archivos a products
- Para subir el archivo voy al FileInterceptor
    - Hay muchas cosas que puedo establecer, como limits
    - En storage uso diskStorage de multer
    - Cuando uso ./ me refiero al root del proyecto

```
import { Controller, Get, Post, Body, Patch, Param, Delete, UploadedFile,
UseInterceptors, BadRequestException } from '@nestjs/common';
import { FilesService } from './files.service';
import { FileInterceptor } from '@nestjs/platform-express';
import { fileFilter } from './helpers/fileFilter.helper';
import { diskStorage } from 'multer';


@Controller('files')
export class FilesController {
  constructor(private readonly filesService: FilesService) {}

  @Post('product')
  @UseInterceptors(FileInterceptor('file', {
    fileFilter: fileFilter,
    limits: {fileSize: 10000},
    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
    }
  }
}
```

- Si voy a la carpeta uploads veo que tengo la imagen
- La guarda con un nombre estraño (único) sin la extensión
- Se suele usat **.gitkeep** para **dar seguimiento a directorios que pueden estar vacíos**, porque git por defecto **no lo hace**
- Vamos a renombrar la imagen que estamos subiendo

Renombrar el archivo subido

- Copio el archivo fileFilter y lo renombro a fileNamer
- En este punto ya debería tener el archivo, pero dejo la validación por si acaso
- Necesito la extensión del archivo, por lo que uso el split

```
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 = `HolaMundo.${fileExtension}`


    callback(null, fileName)

}
```

- Coloco en la propiedad fileName de diskStorage la función

```
import { Controller, Get, Post, Body, Patch, Param, Delete, UploadedFile,
UseInterceptors, BadRequestException } from '@nestjs/common';
import { FilesService } from './files.service';
import { FileInterceptor } from '@nestjs/platform-express';
import { fileFilter } from './helpers/fileFilter.helper';
import { diskStorage } from 'multer';
import { fileNamer } from './helpers/fileNamer.helper';


@Controller('files')
export class FilesController {
  constructor(private readonly filesService: FilesService) {}

  @Post('product')
  @UseInterceptors(FileInterceptor('file', {
    fileFilter: fileFilter,
    limits: {fileSize: 10000},
    storage: diskStorage({
      destination: './static/uploads',
      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
    }
  }
}
```

- Para colocarle un identificador único como nombre de la imágen (en lugar de usar HolaMUndo y que lo vaya reescribiendo) vamos a usar uuid. Instalo también los tipos

> npm i uuid @types/uuid

```
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)
}
```

- Si hago un console.log del file en el controlador veré toda la info
    - fieldname, originalname, encoding, mimetype, destination, filename, path, size
- Nadie desde afuera puede acceder al filesystem, ya que no está en la carpeta pública
- Vamos a ver como devolver la imagen.
- Hay toda una serie de validaciones, de autenticación que hay que hacer que no podría si estuvieran en una carpeta publica

---

Servir 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 en el uploadProductImage

```
uploadProductFile(@UploadedFile() file: Express.Multer.File){

  if(!file) throw new BadRequestException('Make sure that the file is an image')

  const secureURL = `${file.filename}`

  return {
    secureURL
  }
}
```

- El endpoint sería un GET a api/files/product/:nombre_imagen

> localhost:3000/api/files/product/93479347329847UUID.png

- Creo el endpoint GET en el controller
- Me aseguro de recibir la imagen

```
@Get('product/:imageName')
  findProductImage(@Param('imageName') imageName: string ){
    return imageName
  }
```

- Debo verificar que la imagen exista en /products, voy al servicio para escribir el código
- Para eso debo especificar el path en el que me encuentra, está la función **join del path de node**
- Subo dos escalones en la jerarquía con ../../
- Uso **existSync del fs de node**

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


@Injectable()
export class FilesService {


    getStaticProductImage(imageName: string){
        const path = join(__dirname, '../../static/products', imageName)

        if(!existsSync) throw new BadRequestException(`No product found with image
${imageName}`)

        return path
    }
}
```

- Podría usar un genérico para saber en qué carpeta buscar
- Uso el servicio en el controller

```
@Get('product/:imageName')
findProductImage(@Param('imageName') imageName: string ){

  const path = this.filesService.getStaticProductImage(imageName)

  return path
}
```

- Obtengo el path de mi computadora donde está el archivo en el controlador

- En lugar de regresa el path yo quiero regresar la imagen. Para eso haré uso de un nuevo decorador **@Res** de nest/common con el **Response de express**. Podría usar **Express.Response para no hacer la importación**
- En el momento que uso **@Res** rompo la funcionalidad de Nest, yo tomo el control de la respuesta manualmente
- Ahora puedo escribir mi respuesta como haría con Express

```
  @Get('product/:imageName')
  findProductImage(
    @Res() res: Response,
    @Param('imageName') imageName: string ){

    const path = this.filesService.getStaticProductImage(imageName)

     res.status(403).json({
      ok: false
     })
  }
```

- Hay que ir **con cuidado con usar Res** porque se salta ciertos interceptores y restricciones que usa Nest
- Uso sendFile para enviar el archivo que esté en el path

```
@Get('product/:imageName')
findProductImage(
  @Res() res: Response,
  @Param('imageName') imageName: string ){

  const path = this.filesService.getStaticProductImage(imageName)

    res.sendFile(path)
}
```

- De esta manera puedo usar la url de cloudinary o AWS, porque estoy ocultando donde está el archivo fisicamente
- Ahora, el secureURL debería ser este path
- Veamos como construir este url para que al subir el archivo quede listo para ser utilizado
- Hay más cosas que puedo hacer, como que cuando se eliminen verificar que las imágenes se han eliminado

---

Retornar el secureURL

- Puede ser que el puerto y la localización sean otros, con lo que es una url volátil
- La idea es que sea una variable de entorno

```
HOST_API=http://localhost:3000/api
PORT=3000
```

- Para usar las variables de entorno inyecto el servicio ConfigService en el controller

```
import { Controller, Get, Post, Param, Delete, UploadedFile, UseInterceptors,
BadRequestException, Res } from '@nestjs/common';
import { FilesService } from './files.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 FilesController {
  constructor(private readonly filesService: FilesService,
              private readonly configService: ConfigService
    ) {}

  @Post('product')
  @UseInterceptors(FileInterceptor('file', {
    fileFilter: fileFilter,
    limits: {fileSize: 10000},
    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
    }
  }

  @Get('product/:imageName')
  findProductImage(
    @Res() res: Response,
    @Param('imageName') imageName: string ){

    const path = this.filesService.getStaticProductImage(imageName)
```

```
      res.sendFile(path)
    }
  }
```

- Los módulos están encapsulados. Si quiero usar el servicio debo importar el módulo en files.module
- files.module

```
import { Module } from '@nestjs/common';
import { FilesService } from './files.service';
import { FilesController } from './files.controller';
import { ConfigModule } from '@nestjs/config';

@Module({
  controllers: [FilesController],
  providers: [FilesService],
  imports:[ConfigModule]
})
export class FilesModule {}
```

- Ahora cuando subo el archivo me responde con la secureUrl con el path donde se ubica el archivo

```
{
  "secureURL": "http://localhost:3000/api/files/product/23982de9-89be-4157-b1da-
cd8c629726b7.jpeg"
}
```

- Uso la variable de entorno PORT para el puerto del main
- Uso el logger para imprimir en consola el mensaje

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

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

  const logger = new Logger('bootstrap')

  app.setGlobalPrefix('api')

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

```
  await app.listen(process.env.PORT);
  logger.log(`App running on port ${process.env.PORT}`)
}
bootstrap();
```

- Si hago un GET de todos los productos y miro el arreglo de imágenes de los productos, estas imágenes no existen y no son urls

---

Otras formas de desplegar archivos

- Tengo un paquete comprimido con las imágenes que hacen match con la db
- Si yo sé que estos archivos no van a cambiar y siempre se van a servir de manera estática, accesible para todo el mundo, no hace falta hacer el Restful API
- Creo la carpeta public y copio dentro la carpeta products con todas las imágenes
- products no es el mejor nombre, ya que desde el front esa ruta podría estar tomada, le pongo assets
- Para servir contenido estático debo instalar @nestjs/serve-static y usar ServeStaticModule con el path

> @nestjs/serve-static

- En app.module

```
import { Module } from '@nestjs/common';
import {ConfigModule} from '@nestjs/config'
import { TypeOrmModule } from '@nestjs/typeorm';
import { ProductsModule } from './products/products.module';
import { CommonModule } from './common/common.module';
import { SeedModule } from './seed/seed.module';
import { FilesModule } from './files/files.module';
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,

    FilesModule,

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

- Conviene crear un index.html en la carpeta public, aunque sea de prueba, para que no de error en consola
- En el endpoint localhost:3000/assets/nombre_del_archivo.jpeg en el navegador me muestra la imagen
- De esta manera no puedo controlar quien accede a las imágenes.
- Son recursos públicos, estáticos que no van a cambiar
- Así como lo tengo en la db, no puedo acceder a las imágenes
- Habría que crear un endpoint que deduzca que la imagen de la db es la que tengo guardada y colocar el url completo o actualizar las imágenes
- Una solución viable es actualizar las imágenes de la db, se podría hacer mediante el seed
- Básicamente sería añadirles el **http://localhost:3000/api/assets** y concatenar el campo url

```
update product_images set url = 'http://localhost:3000/api/assets' || url
```

- Pero de esta manera estoy grabando mucha data de manera innecesaria, ya que repite http://localhost:3000/api/assets en cada imagen

---

Colocar imágenes en el directorio estático

- Copio las imágenes y las pego en la carpeta static/products
- De hecho no necesito el directorio public, era solo con fines educativos
- Las imágenes ya tienen una referencia en la db (coincide el nombre)
- Si apunto al endpoint, obtengo la imagen

> http://localhost:3000/api/files/product/1473809-00-A_1_2000.jpg