NEST 13

Subir un archivo al backend

- Instalación de los tipos de typescript. Multer ya está instalado

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

- La carga de archivos puedo necesitarla en varias aplicaciones, es algo general
- Creo un resource nuevo

```
nest g res files --no-spec
```

- Elimino todo lo que necesito: dto, entity
- En los servicios borro todos los métodos, en el controller también
- En el controller, llamo al endpoint product:

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

  @Post('product')
  uploadProductImage(){
    return "hola mundo"
  }
}
```

- Pruebo el endpoint con una petición Post

```
http://localhost:3000/api/files/product
```

- Le añado el tipado al archivo

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

- Voy a THunderClient y en Body, tipo de dato Form, activo la casilla de Files y añado un archivo con nombre File
- Doy al SEND y me devuelve un 201 created pero no hay nada en la respuesta. Le falta el decorador @UploadedFile!
- El UploadedFile necesita saber que el file es el archivo, para ello usaré un interceptor
- Los interceptores interceptan las solicitudes y tmabién pueden interceptar y mutar las respuestas
- Hay interceptores globales, por método, por controlador... en este caso por controlador

- Uso el FileInterceptor de express. tengo que especificar el nombre de la propiedad que estoy enviando, en este caso file

```
import { Controller, Post, 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') )
  uploadProductImage(
    @UploadedFile() file: Express.Multer.File){
    return file
  }
}
```

- Este archivo por defecto se sube a una carpeta temporal. Cuando no se use node lo limpiará por si solo
- Debo decirle que hacer con este archivo
- No es recomendable guardar el archivo en fileSystem por motivos de seguridad
- En un servidor interno pequeño no habría problema, pero si va a exponerse a internet es recomendable un servicio de terceros

Validar Archivos

- Por ahora quiero validar que no me suba un pdf, solo imágenes
- Lo voy a colocar en files ya que es algo que solo voy a usar en este módulo
- Creo una carpeta llamada helpers. La idea es tener funciones que específicamente trabajen en este módulo
- Creo el archivo fileFilter.helper.ts con la función de flecha fileFilter
- Esta función tiene que lucir para poder usarlo en el FileInterceptor
- Si escribo una coma dentro del FileInterceptor, después del primer argumento, abro llaves, pongo el fileFilter y coloco el cursor encima, veo lo que me está pidiendo para que funcione

```
  @Post('product')
  @UseInterceptors( FileInterceptor('file', {fileFilter}) ) //coloca el cursor
encima
  uploadProductImage(
    @UploadedFile() file: Express.Multer.File){
    return{
      filename: file.originalname
    }
  }
```

- Me pide la request, el archivo y un callback con el error y el acceptFile como boolean. Regresa void
- Tengo que hacer el fileFilter que se adapte a esta forma para poder usarlo en el interceptor
- Le pongo el error en null, y el accept en true
- fileFilter.helper.ts:

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

console.log(file)

callback(null, true)

}
```

- Mando la referencia a la función, no lo estoy ejecutando. El fileInterceptor ejecutará la función con los argumentos

```
import { Controller, Post, 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}) )
  uploadProductImage(
    @UploadedFile() file: Express.Multer.File){

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

- El console.log del fileFilter me devuelve en consola lo siguiente

```
{
  fieldname: 'file',
  originalname: 'ekaterina.jpg',
  encoding: '7bit',
  mimetype: 'image/jpeg'
}
```

- Ahora ya puedo hacer las evaluaciones
- Si el archivo no viene puedo llamar al callback con el error y el accept en 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', 'gif' ]

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

callback(null, false)

}
```

- Esto no va a lanzar el error de la excepción
- controller

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

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

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

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

Guardar imagen en fileSystem

- Creo las carpetas static/uploads en la raíz del proyecto. No le pongo public porque usualmente voy a tener archivos con ciertas restricciones de lectura
- Puedo establecerle limites de peso
- Hay otra propiedad que es el storage con cierta configuración

- Cuando pongo ./ en el path estoy diciendo la raíz del proyecto -files.controller:

```
import { BadRequestException, Controller, Post, UploadedFile, UseInterceptors }
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: 10000000},
    storage: diskStorage({
      destination: './static/uploads'
    })
  }) )
  uploadProductImage(
    @UploadedFile() file: Express.Multer.File){

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

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

- Al archivo le falta la extensión, necesita un nombre personalizado. en la proxima clase
- NOTA: Cuando se trabaja con carpetas vacías .gitkeep dentro de la carpeta se asegura de que los directorios sean preservados cuando se hacen commits

Renombrar el archivo subido

- Copio el archivo fileFilter y lo renombro a fileNamer
- Voy a necesitar la extensión.
- Como segundo argumento del callback va el nombre del archivo

```
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) // nombre del archivo

    }
```

- Lo añado al storage -files.controller:

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

  @Post('product')
  @UseInterceptors( FileInterceptor('file', {
    fileFilter: fileFilter,
    limits: {fileSize: 10000000},
    storage: diskStorage({
      destination: './static/uploads',
      filename: fileNamer
    })
  }) )
  uploadProductImage(
    @UploadedFile() file: Express.Multer.File){

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

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

- Ahora el objetivo es colocarle un identificador único a la imagen

> npm i 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) // nombre del archivo

    }
```

- Cómo poder regresar las imágenes? en la siguiente lección

Servir archivos de manera controlada

- Regreso el nombre de la imagen, lo guardo en una constante y la sirvo en el return
- Lo que debo regresar es la url para poder verla

```
import { BadRequestException, Controller, Post, UploadedFile, UseInterceptors }
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: 10000000},
    storage: diskStorage({
      destination: './static/uploads',
      filename: fileNamer
    })
  }) )
  uploadProductImage(
    @UploadedFile() file: Express.Multer.File){

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

      const secureUrl = `${file.filename}`

    return{
      secureUrl
    }
  }
}
```

- Haré una petición Get al nombre de la imagen

> http://localhost:3000/api/files/product/88b276ea-49e8-4b1f-964b-4fb62e78697f.jpeg

- Creo el endpoint en el controller y lo manejo con @Param
- files.controller:

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

    return imageName
  }
```

- Para la lógica voy al files.service y me creo un método
- Debo validar si la imagen existe y si existe mostrar el path completo
- Primero debo de especificar en que path me encuentro, para eso hay una función que se llama join de node
- Construyo el path con __dirname, me voy un directorio atrás y le concateno el nombre del archivo de imagen
- Si no existe existsSync (también de node, el path que yo creé)

```
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/uploads', imageName)

        if (!existsSync(path))
            throw new BadRequestException('No product found')

        return path
    }
}
```

- Uso el getStaticProductImage en la petición Get del controller
- files.controller

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

    const path = this.filesService.getStaticProductImage(imageName)

    return path
  }
```

- Esto me devuelve el path:

> C:\Users\castd\Documents\NEST\POSTGRESShop\teslo-shop\static\uploads\88b276ea-49e8-4b1f-964b-4fb62e78697f.jpeg

- Pero en lugar de regresar el path yo quiero regresar la imagen. Para eso haré uso de un decorador @Res
- Cuando se usa este decorador, se rompe la funcionalidad de nest, el return ya no sirve
- Manualmente voy a emitir mi respuesta
- Uso el res.sendFile(path) para enviar la imagen

```
import { BadRequestException, Controller, Get, Param, Post, Res, UploadedFile,
UseInterceptors } 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';


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

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

    const path = this.filesService.getStaticProductImage(imageName)

    res.sendFile( path )
  }

  @Post('product')
  @UseInterceptors( FileInterceptor('file', {
    fileFilter: fileFilter,
    limits: {fileSize: 10000000},
    storage: diskStorage({
      destination: './static/uploads',
      filename: fileNamer
    })
  }) )
  uploadProductImage(
    @UploadedFile() file: Express.Multer.File){

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

```
      const secureUrl = `${file.filename}`

    return{
      secureUrl
    }
  }
}
```

- Físicamente no se sabe cual es el path pero se muestra la imagen
- Ahora, ese path de la petición debería ser el secureUrl que regresé a la hora de cargar la imagen

> http://localhost:3000/api/files/product/88b276ea-49e8-4b1f-964b-4fb62e78697f.jpeg

- Para que cuando se cargue el archivo se pueda hacer clic en la url, y esa url sea la que yo cargue en imagenes de la DB

Retornar el secureUrl

- Para construir la url bien podría usar unavariable con localhost:3000 y colocarla entre back ticks
- Pero es una info muy volátil, al desplegar la aplicación no tendría el puerto, con lo cual voy a hacerlo con una variable de entorno
- Voy al archivo .env

> HOST_API=http://localhost:3000/api

- Necesito esa variable de entorno. Para ello voy a inyectar en el controller el ConfigService

```
import { BadRequestException, Controller, Get, Param, Post, Res, UploadedFile,
UseInterceptors } 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) {}

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

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

```
    res.sendFile( path )
  }

  @Post('product')
  @UseInterceptors( FileInterceptor('file', {
    fileFilter: fileFilter,
    limits: {fileSize: 10000000},
    storage: diskStorage({
      destination: './static/uploads',
      filename: fileNamer
    })
  }) )
  uploadProductImage(
    @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
    }
  }
}
```

- Esto da un error:

> Nest can't resolve dependencies of the FilesController (FilesService, ?). Please make sure that the argument ConfigService at index [1] is available in the FilesModule context

- Todos los módulos están encapsulados. Si quiero usar este servicio voy a tener que proporcionarlo en el módulo

```
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 un Post a este endpoint (con el archivo en el body) debería devolver algo así

> "secureUrl": "http://localhost:3000/api/files/product/ea2014b1-432c-4341-9ec9-01bd231b4d57.jpeg"

- Esta url ya es válida, si ctrl+clic veo la imagen en el navegador
- Hago un pequeño cambio en el main, cambiando el puerto por una variable de entorno y creando un logger para ver en consola un LOG mas elegante

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

async function main() {
  const app = await NestFactory.create(AppModule);
  const logger = new Logger('main')

  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}`)
}
main();
```

Otras formas de desplegar archivos

- Si se que son archivos estáticos y nunca van a cambiar no hace falta todo el rollo del controlador y la url
- Creo un nuevo directorio en la raíz llamado public. Ahí colocaré la carpeta products con las imágenes que hacen match con mi base de datos
- Mejor renombrar products porque podría ser una ruta tomada (public/products) en React
- Yo lo que quiero es decirle a NEST toma esta carpeta y sirvela como contenido estático
- Hay que importar en app.module ServeStaticModule

> npm i @nestjs/serve-static

- Coloco la configuración, importo join y creo el path
- 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
    }),
    ServeStaticModule.forRoot({
      rootPath: join(__dirname, '..', 'public')
    }),
    ProductsModule,
    CommonModule,
    SeedModule,
    FilesModule
  ],
  controllers: [],
  providers: [],
})
export class AppModule {}
```

- Para que no muestre este error al escribir en el navegador localhost:3000 hay que crear un index.html con cualquier cosa en public/
- Si ahora coloco el endpoint de localhost con el nombre de una foto me la muestra

> http://localhost:3000/products/1473809-00-A_1_2000.jpg

- Si las imagenes no van a cambiar y pueden ser vistas por tod@s esta puede ser una manera
- Ahora falta que cuando busco un producto en la base de datos, las imagenes sean las url

Colocar imágenes en el directorio estático

- Copio y pego las imagenes de public/products a static/uploads (borro public/)
- Si ahora hago un Get a

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

- Ahora puedo ver las imagenes