Subir Archivos NEST

- Creo el proyecto

```
nest new upload-files
```

- Instalo y configuro swagger

```
npm i @nestjs/swagger swagger-ui-express
```

- En el main añado la configuración

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

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

  const config = new DocumentBuilder()
    .setTitle('Upload files')
    .setDescription('Upload files API')
    .setVersion('1.0')
    .build();
  const document = SwaggerModule.createDocument(app, config);
  SwaggerModule.setup('swagger', app, document);

  await app.listen(3000);
}
bootstrap();
```

- Dejo solo el app.module y el main (el resto de archivos los borro y los quito del app.module quedando así)
- app.module

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


@Module({
  imports: [],
  controllers: [],
  providers: [],
})
export class AppModule {}
```

Creando el módulo

- Creo src/modules y en modules creo el RES con **nest g res upload-file**. Si me sobran endpoints los borro y ya está
- Añado el **UploadFileModule en imports de app.module**
- No habrá conexión con DB

MulterModule

- Necesito instalar los tipos de multer, que es el paquete que viene de forma nativa para subir ficheros

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

- En el upload-file.module importo el módulo de multer. Lo iremos cumplimentando y mejorando.

```
import { Module } from '@nestjs/common';
import { UploadFileService } from './upload-file.service';
import { UploadFileController } from './upload-file.controller';
import { MulterModule } from '@nestjs/platform-express';

@Module({
  imports:[
    MulterModule
  ],
  controllers: [UploadFileController],
  providers: [UploadFileService]
})
export class UploadFileModule {}
```

Subir un archivo

- Creo el endpoint POST para subir el archivo
- Uso el decorador **@UseInterceptors**, y con **FileInterceptor** le digo que vendrá el archivo 'file' (es lo que usaré en POSTMAN)
- Para recoger el archivo uso **@UploadedFile** del tipo Express.Multer.File

```
@Post('upload')
@UseInterceptors(FileInterceptor('file'))
uploadFile(@UploadedFile() file: Express.Multer.File) {
return this.uploadFileService.uploadFile(file);
}
```

- En el servicio, lo único que vamos a hacer es gestionar la respuesta. No hay que indicarle la acción de subirlo
- Le indico el nombre original del archivo y el filename que es el que yo le voy a asignar

```
uploadFile(file: Express.Multer.File) {
    if(file){
      const response = {
        originalName: file.originalname,
        filename: file.filename
      }
      return response
    }
    return null
  }
```

Subiendo archivo con POSTMAN/ThunderClient

- En Body/Form, en KEY le pongo el nombre file (como le puse en el FileInterceptor). Le digo que es de tipo File y selecciono el archivo a subir (cualquier .jpg)
- Así, sin configurar el MulterModule, tan solo me devuelve la respuesta, que es el originalName con el nombre original de la foto
- Configuremos el MulterModule
    - Le añado dest, de destino y le especifico una ruta. Si la carpeta no existe la creará
    - Ahora si me devuelve un filename como un id en la respuesta, sin extensión

```
@Module({
  imports:[
    MulterModule.register({
      dest: './upload'
    })
  ],
  controllers: [UploadFileController],
  providers: [UploadFileService]
})
export class UploadFileModule {}
```

- Para subir varios archivos a la vez creo otro endpoint POST en el controller
- En lugar de usar FileInterceptor usaré **FilesInterceptor** y **UploadedFiles**. Le indico que es de tipo array
- upload-file.controller

```
@Post('upload-files')
@UseInterceptors(FilesInterceptor('files'))
uploadFiles(@UploadedFiles() files: Express.Multer.File[]) {
return this.uploadFileService.uploadFiles(files);
}
```

- En el servicio uso un for of para recorrer el array de files
- Uso el método anterior para gestionar la respuesta
- En caso de que todo esté bien, hago un push al array de respuestas y lo retorno

```
uploadFiles(files: Express.Multer.File[]){
const responses = []

for(const file of files){
    const fileUpload = this.uploadFile(file)
    if(fileUpload){
    responses.push(fileUpload)
    }
}

return responses
}
```

- Ahora en ThunderClient, en Body/Form Files, en Key debo poner files (con s final, como puse en FilesInterceptor) y añado los archivos que quiera subir

---

Limitando el tamaño de los ficheros

- Le añado la propiedad limits y dentro del objeto en fileSize debo expresarlo en bytes
- Si quiero que no sobrepase los 2 Megas, lo multiplico **2 veces por 1024** (de byte pasa a KB, y de KB a MB)
- Si quiero solo subir imágenes uso **fileFilter** que es una función. Esta lleva la request (de Express), el file que subimos y el callback
- Uso una expresión regular. El $ dice que tiene que acabar en uno de estos tipos de archivos
- En caso de que no sea de alguno de estos tipos retorno el callback **con el que puedo lanzar una excepción**
- Si está bien le pongo **null** en el error, **y le digo true** (que está aceptado)

```
@Module({
  imports:[
    MulterModule.register({
      dest: './upload',
      limits:{
        fileSize: 2 * 1024 *1024
      },
      fileFilter: function(req, file, cb){
        if(!file.originalname.match(/\.(jpg|jpeg|png|gif)$/)){
            return cb(new ConflictException("Solo imágenes"), false)
        }
        return cb(null, true)
      }
    })
  ],
  controllers: [UploadFileController],
  providers: [UploadFileService]
})
export class UploadFileModule {}
```

Mejorando dónde almacenar archivos

- Uso la función diskStorage de storage
- Si quisiera filtrar los archivos por extension y guardarlos en diferentes carpetas aquí sería el sitio

```
@Module({
  imports:[
    MulterModule.register({
      limits:{
        fileSize: 2 * 1024 *1024
      },
      fileFilter: function(req, file, cb){
        if(!file.originalname.match(/\.(jpg|jpeg|png|gif)$/)){
            return cb(new ConflictException("Solo imágenes"), false)
        }
        return cb(null, true)
      },
      storage: diskStorage({
        destination: function(req,file,cb){
          cb(null, './upload')
        }
      })
    })
  ],
  controllers: [UploadFileController],
  providers: [UploadFileService]
})
export class UploadFileModule {}
```

Cambiar el nombre de los ficheros al subirlos

- Quiero añadir al archivo la fecha en la que se ha añadido
- Dentro del diskStorage uso otra función en filename
- Uso el split para dividir el archivo por el punto
- Con el slice le digo y la posición 0 al length - 1 le digo que me de todo desde el principio menos la útima parte después del punto
- Vuelvo a unir las partes (menos la extensión que ya no la tengo)
- Si tiene un mimetype hago el split separando por / y me quedo con lo que hay después del / (la extensión)
- Llamo al callback, le paso null en el error y le concateno el Date.now al nombre del archivo
- En caso de que no haya mimetype devuelveme lo mismo (con el Date.now) pero sin la extensión

```
import { ConflictException, Module } from '@nestjs/common';
import { UploadFileService } from './upload-file.service';
import { UploadFileController } from './upload-file.controller';
import { MulterModule } from '@nestjs/platform-express';
import { diskStorage } from 'multer';
```

```
@Module({
  imports:[
    MulterModule.register({
      limits:{
        fileSize: 2 * 1024 *1024
      },
      fileFilter: function(req, file, cb){
        if(!file.originalname.match(/\.(jpg|jpeg|png|gif)$/)){
            return cb(new ConflictException("Solo imágenes"), false)
        }
        return cb(null, true)
      },
      storage: diskStorage({
        destination: function(req,file,cb){
          cb(null, './upload')
        },
        filename: function(req,file,cb){
          let filenameParts = file.originalname.split('.')
          filenameParts = filenameParts.slice(0, filenameParts.length -1)
          const filename = filenameParts.join('.')

          if(file.mimetype){
            let ext = file.mimetype.split('/')[1]
            cb(null, filename + '-'+ Date.now()+ '.'+ ext)
          }else{
            cb(null, filename + '-'+ Date.now())
          }
        }
      })
    })
  ],
  controllers: [UploadFileController],
  providers: [UploadFileService]
})
export class UploadFileModule {}
```

- El mimetype se ve tipo **image/jpeg**. Puedes hacer un console.log para verlo

---

Descargar un archivo

- Para descargar creamos un GET en el controller
- Usare el response con **@Response** y el **@Body**

```
@Get('download')
download(@Response() res, @Body() body:any ) {
  console.log(res)
  console.log(body)
  return this.uploadFileService.download(res, body.filename);
}
```

- En el service coloco un return true para hacer las pruebas, me interesan los console.logs
- upload-file.service

```
download(res, filename: string){
  return true
}
```

- en el thunderClient en Body/x-www-form-urlencoded Files en la KEY añado filename y en el value el nombre del archivo a descargar
- res tiene una función llamada download donde le puedo indicar la url del fichero que quiero descargar

```
download(res, filename: string){

  return res.download('./upload/' + filename)

}
```

- Nos devuelve la imágen (si estuviera en un navegador lo abriria o lo guardaría)
- Para comprobar si existe o no podemos usar existsSync

```
download(res, filename: string){

    if(existsSync('./upload/'+ filename)){

      return res.download('./upload/' + filename)
    }

    return new NotFoundException("El fichero no existe")

  }
```

Documentando endpoints

- controller

```
import { Controller, Get, Post, Body, Patch, Param, Delete, UseInterceptors,
UploadedFile, UploadedFiles, Response } from '@nestjs/common';
import { UploadFileService } from './upload-file.service';
import { CreateUploadFileDto } from './dto/create-upload-file.dto';
import { UpdateUploadFileDto } from './dto/update-upload-file.dto';
import { FileInterceptor, FilesInterceptor } from '@nestjs/platform-express';
import { ApiOperation, ApiResponse } from '@nestjs/swagger';
```

```
@Controller('api/v1/upload-file')
export class UploadFileController {
  constructor(private readonly uploadFileService: UploadFileService) {}

  @Post('upload')
  @ApiOperation({
    description: "Sube un fichero"
  })
  @ApiResponse({
    status: 201,
    description: "Se ha subido correctamente"
  })
  @UseInterceptors(FileInterceptor('file'))
  uploadFile(@UploadedFile() file: Express.Multer.File) {
    return this.uploadFileService.uploadFile(file);
  }

  @Post('upload-files')
  @ApiOperation({
    description: "Sube varios ficheros"
  })
  @ApiResponse({
    status: 201,
    description: "Se han subido correctamente"
  })
  @UseInterceptors(FilesInterceptor('files'))
  uploadFiles(@UploadedFiles() files: Express.Multer.File[]) {
    return this.uploadFileService.uploadFiles(files);
  }

  @Get('download')
  @ApiOperation({
    description: "Descarga archivo"
  })
  @ApiResponse({
    status: 200,
    description: "Se ha descargado correctamente"
  })
  @ApiResponse({
    status: 409,
    description: "No existe el archivo"
  })
  download(@Response() res, @Body() body:any ) {
    console.log(body)
    console.log(res)
    return this.uploadFileService.download(res, body.filename);
  }

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

```
  @Patch(':id')
  update(@Param('id') id: string, @Body() updateUploadFileDto:
UpdateUploadFileDto) {
    return this.uploadFileService.update(+id, updateUploadFileDto);
  }

  @Delete(':id')
  remove(@Param('id') id: string) {
    return this.uploadFileService.remove(+id);
  }
}
```

- Falta poder seleccionar un fichero en la documentación de swagger

---

Subir archivos en swagger

- En el upload colocamos un nuevo decorador llamado **@ApiConsumes**
- Le indicamos que tipo de "body" va a captar
- En **@ApiBody** le indico que es de tipo objeto, con la propiedadd de nombre file (como puse en FileInterceptor)

```
@Post('upload')
  @ApiOperation({
    description: "Sube un fichero"
  })
  @ApiResponse({
    status: 201,
    description: "Se ha subido correctamente"
  })
  @ApiConsumes('multipart/form-data')
  @ApiBody({
    schema: {
      type: 'object',
      properties:{
        file: {
          type: 'string',
          format: 'binary'
        }
      }
    }
  })
  @UseInterceptors(FileInterceptor('file'))
  uploadFile(@UploadedFile() file: Express.Multer.File) {
    return this.uploadFileService.uploadFile(file);
  }
```

- Con varios archivos es un poco diferente. Sería la propiedad files y sería de tipo array

```
@Post('upload-files')
  @ApiOperation({
    description: "Sube varios ficheros"
  })
  @ApiResponse({
    status: 201,
    description: "Se han subido correctamente"
  })
  @ApiConsumes('multipart/form-data')
  @ApiBody({
    schema: {
      type: 'object',
      properties:{
        files: {
          type: 'array',
          items:{
            type: 'string',
            format: 'binary'
          }
        }
      }
    }
  })
  @UseInterceptors(FilesInterceptor('files'))
  uploadFiles(@UploadedFiles() files: Express.Multer.File[]) {
    return this.uploadFileService.uploadFiles(files);
  }
```

- el download la propiedad es filename, y es de tipo string

```
@Get('download')
  @ApiOperation({
    description: "Descarga archivo"
  })
  @ApiResponse({
    status: 200,
    description: "Se ha descargado correctamente"
  })
  @ApiResponse({
    status: 409,
    description: "No existe el archivo"
  })
  @ApiConsumes('multipart/form-data')
  @ApiBody({
    schema: {
      type: 'object',
      properties:{
        filename: {
          type: 'string',
        }
      }
```

```
    }
  })
  download(@Response() res, @Body() body:any ) {
    console.log(body)
    console.log(res)
    return this.uploadFileService.download(res, body.filename);
  }
```