Logs NEST

- Creo el proyecto con nest new logs-app
- Borro todos los archivos que no necesito como los test o el app.service y el app.controller (quito las dependencias)
- Instalo swagger y lo configuro
- Vamos a reutilizar el proyecto anterior de Cron
- Borro la carpeta src y la reemplazo por la de cron
- Necesito instalar el @nestjs/schedule y los types de cron
- Instalo Winston

> npm i winston

- Crearemos una instancia de logger por cada tipo de transport que queramos hacer, aunque transports sea un array
- No es como sale en la documentación

Creando nuestro módulo de logger de forma global

- Creo el módulo en src/modules con nest g res logger, aunque no necesito endpoints.
- Le diremos que el módulo es global. Una vez importe este módulo en el app.module no voy a tener que importarlo más
- Tiene lógica, ya que el módulo de logger es necesario en todo el resto de módulos
- Por ejemplo, la configuración de Mongo también puede ser global
- Tengo que exportar el servicio para que esté disponible

```
import { Global, Module } from '@nestjs/common';
import { LoggerService } from './logger.service';
import { LoggerController } from './logger.controller';

@Global()
@Module({
  controllers: [LoggerController],
  providers: [LoggerService],
  exports:[LoggerService]
})
export class LoggerModule {}
```

- Importo el módulo de Logger en el imports de app.module.
- Se recomienda poner los módulos globales primero en el array de imports

---

Creando el servicio

- Si digo: quiero los loggers de errores, o de info, o de warns, o todos.
- Tengo que crear los campos en el servicio

- En el método createLogger primero formateo el output (para que salga la fecha, la hora...)
- printf nos devuelve un objeto que vamos a llamar log
- Tiene tres propiedades (aunque timestamp no salga con el intellisense)
- Uso un template string para concatenarlas. Uso solo la primera letra en mayúscula del level

```
import { Injectable } from '@nestjs/common';
import { Logger, format } from 'winston';

@Injectable()
export class LoggerService {

    private loggerInfo: Logger
    private loggerError: Logger
    private loggerWarn: Logger
    private loggerAll: Logger


    constructor(){
    }

    createLogger(){
        const textFormat = format.printf((log)=>{
           return `${log.timestamp}- ${log.level.toUpperCase().charAt(0)}-
${log.message}`
        })
    }
}
```

- Formateo la fecha también. Le pongo año, mes, días, horas, minutos y segundos
- Creo el primer logger de info.
- Uso createLogger de Winston
- Uso .combine para combinar los dos formatos
- Transports es para indicar dónde quiero guardarlo

```
    createLogger(){
        const textFormat = format.printf((log)=>{
           return `${log.timestamp}- ${log.level.toUpperCase().charAt(0)}-
${log.message}`
        })

        const dateFormat = format.timestamp({
            format: 'YYYY-MM-DD HH:MM:SS'
        })

        this.loggerInfo = createLogger({
            level: 'info',
            format:format.combine(
                dateFormat,
                textFormat
```

```
            ),
            transports:[
                new transports.File({
                    filename: 'log/info/info.log'
                })
            ]

        })
    }
```

- El resto de loggers serían practicamente igual, solo cambio la ruta de ubicación en el transports con el nombre correspondientes
- Al loggerAll no le coloco level porque es en todos y lo guardo en dos ubicaciones
    - Uso un transport para el archivo log y otro transport para la consola

```
this.loggerAll = createLogger({
    format:format.combine(
        dateFormat,
        textFormat
    ),
    transports:[
        new transports.File({
            filename: 'log/all/all.log'
        }),
        new transports.Console()
    ]
})
```

Arrancando nuestro logger

- Hay que crear una instancia del logger en el main
- main.ts

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { LoggerService } from './modules/logger/logger.service';


async function bootstrap() {
  const app = await NestFactory.create(AppModule, {
    logger: new LoggerService()
  });
  await app.listen(3000);
}
bootstrap();
```

- Esto da error porque el LoggerService debe tener unos métodos concretos. Los creo
- logger.service

```
log(message: string){

}
error(message: string){

}
warn(message: string){

}
debug(message: string){

}
verbose(message: string){

}
```

- Una vez pongo estos métodos en el logger.service desaparece el error
- Uso los loggers que he creado y los coloco dentro de su método correspondiente

```
log(message: string){
    this.loggerInfo.info(message)
    this.loggerAll.info(message)
}
error(message: string){
    this.loggerError.error(message)
    this.loggerAll.error(message)
}
warn(message: string){
    this.loggerWarn.warn(message)
    this.loggerAll.warn(message)
}
```

- Debo llamar al método createLogger del servicio en el constructor

```
import { Injectable } from '@nestjs/common';
import { Logger, createLogger, format, transports } from 'winston';

@Injectable()
export class LoggerService {

    private loggerInfo: Logger
    private loggerError: Logger
    private loggerWarn: Logger
    private loggerAll: Logger
```

```
    constructor(){
        this.createLogger()
    }

    createLogger(){
        const textFormat = format.printf((log)=>{
           return `${log.timestamp}- ${log.level.toUpperCase().charAt(0)}-
${log.message}`
        })

        const dateFormat = format.timestamp({
            format: 'YYYY-MM-DD HH:MM:SS'
        })

        this.loggerInfo = createLogger({
            level: 'info',
            format:format.combine(
                dateFormat,
                textFormat
            ),
            transports:[
                new transports.File({
                    filename: 'log/info/info.log'
                })
            ]
        })
        this.loggerError = createLogger({
            level: 'error',
            format:format.combine(
                dateFormat,
                textFormat
            ),
            transports:[
                new transports.File({
                    filename: 'log/error/error.log'
                })
            ]
        })
        this.loggerWarn = createLogger({
            level: 'warn',
            format:format.combine(
                dateFormat,
                textFormat
            ),
            transports:[
                new transports.File({
                    filename: 'log/warn/warn.log'
                })
            ]
        })
        this.loggerAll = createLogger({
            format:format.combine(
                dateFormat,
```

```
                textFormat
            ),
            transports:[
                new transports.File({
                    filename: 'log/all/all.log'
                }),
                new transports.Console()
            ]
        })

    }

    log(message: string){
        this.loggerInfo.info(message)
        this.loggerAll.info(message)
    }
    error(message: string){
        this.loggerError.error(message)
        this.loggerAll.error(message)
    }
    warn(message: string){
        this.loggerWarn.warn(message)
        this.loggerAll.warn(message)
    }
    debug(message: string){

    }
    verbose(message: string){

    }
}
```

Usando LoggerService

- Cómo hicimos global el LoggerModule, si quiero usar el loggerService en el módulo de cron no tengo que importar nada en imports ni exports, solo el servicio como tal en las importaciones como cualquier otro paquete
- Inyecto el servicio
- Sustituimos los console.log de los métodos cron por los del logger

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { Cron, SchedulerRegistry } from '@nestjs/schedule';
import { CronJob } from 'cron';
import { LoggerService } from '../logger/logger.service';

@Injectable()
export class CronService {

  constructor(private readonly scheduleRegistry: SchedulerRegistry,
              private readonly loggerService: LoggerService){
```

```
}

@Cron(`*/10 * * * * *`, {
  name: 'cron1'

})
cron1(){
  this.loggerService.log("cron1: Acción cada 10 segundos")
}

@Cron(`*/30 * * * * *`, {
  name: 'cron2'

})
cron2(){
  this.loggerService.error("cron1: Acción cada 30 segundos")
}

@Cron(`* * * * *`, {
  name: 'cron3'

})
cron3(){
  this.loggerService.warn("cron1: Acción cada minuto")
}

desactivateCron(name:string){
  const job: CronJob = this.scheduleRegistry.getCronJob(name)

  if(!job){
    throw new NotFoundException("No se ha encontrado el cron")
  }else{
    job.stop()
    console.log(`El cron con el nombre: ${name} está desactivado`)
    return true
  }
}
activateCron(name:string){
  const job: CronJob = this.scheduleRegistry.getCronJob(name)

  if(!job){
    throw new NotFoundException("No se ha encontrado el cron")
  }else{
    job.start()
    console.log(`El cron con el nombre: ${name} está activado`)
    return true
  }
}

getNameCrons(){
  const names = []

  for(const name of this.scheduleRegistry.getCronJobs().keys()){
```

```
        names.push(name)
      }
      return names
    }

    desactivateAllCrons(){
      const names = this.getNameCrons()

      for(const name of names){
        this.desactivateCron(name)
      }
      return true
    }

    activateAllCrons(){
      const names = this.getNameCrons()

      for(const name of names){
        this.activateCron(name)
      }
      return true
    }
  }
```

- Pongo warn y error aleatoriamente solo con fines educativos

Instalar Winston rotate

- Esta dependencia separará los logs por días

```
npm i winston-daily-rotate-file
```

- Para rotar logs por días debo cambiar el transports.File del método createLogger
- Se importa distinto

```
import 'winston-daily-rotate-file';
```

```
this.loggerInfo = createLogger({
    level: 'info',
    format:format.combine(
        dateFormat,
        textFormat
    ),
    transports:[
        new transports.DailyRotateFile({
            filename: 'log/info/info-%DATE%.log', //le añado la fecha en el nombre
del archivo
            datePattern: 'YYY-MM-DD', //formateo la fecha
            maxFiles:'7d', //le digo que conserve los archivos un máximo de 7 días
```

```
            zippedArchive: true //que comprima los archivos
        })
    ]
})
```

- Si pongo maxFiles y uso el zippedArchive, los irá manteniendo, porque no borra los .zip
- Hago lo mismo con el resto de transports