Gestión de **Logs** en **Plugcore**

Plug Soluciones TIC S.L.
info@plugcore.com


https://github.com/plugcore/plugcore


https://www.linkedin.com/company/plugcore


https://twitter.com/plugcoreit

Gestión de logs

1. **Introducción**
2. **Cómo usarlo**
3. **Campos de un log**
4. **Configuración**
5. **Transports**

Introducción

- Está basado en pino, que es una librería que sólo se encarga de crear los logs en formato JSON y sacarlos por stdout (Standard output, o lo que es lo mismo, por consola).

- La idea es que la gestión de los logs se haga a través de una aplicación que se ejecuta en paralelo a la nuestra, leerá ese stdout, y actuará en consecuencia, por ejemplo guardando en ficheros y/o en base de datos.

- Lo bueno de este sistema es que dado que la gestión de logs puede consumir muchos recursos, al hacerlo en una aplicación a parte, no ralentizará nuestra aplicación.

Cómo usarlo

- Lo único que tenemos que hacer es importarnos `Logger` en nuestro servicio:

```
@Service()
export class MyService {

    constructor(
        private log: Logger
    ) {
        this.log.info('Hi!');
    }

}
```

- En consola saldrá

```
{"level":30,"time":1586161456996,"pid":11152,"hostname":"DESKTOP-XXXXX","name":"main",
"msg":"My test log","v":1}
```

Campos de un log

- Como podemos observar se genera un JSON con esta propiedades
  - **level**: El nivel de criticidad del log, podemos ver los niveles por defecto de pino aquí
  - **time**: La fecha en la que se ejecutó el log, Epoch Time en milisegundos
  - **pid**: Identificador de proceso de nuestra aplicación
  - **hostname**: Nombre de la máquina en la que se está ejecutando
  - **name**: Variable que usamos para categorizar nuestros los, más adelante podemos ver cómo lo podemos modificar para usar nuestra propias categorías.
  - **msg**: Cuando haces un log simplemente de un string, se crea esta propiedad para poner el valor
  - **Resto de propiedades**: Cuando haces un log de un objeto, todas sus propiedades se agregan al objeto de log al mismo nivel.

Configuración

- Podemos modificar algunos aspectos de la generación de logs usando el sistema de configuración de plugcore, siguiendo la estructura de pino:

```
{
    "log": {
// Así estamos indicando que sólo aparezcan errores de tipo error o superiores
        "level": "error",
// Para que no aparezcan ni "pid" ni "hostname" en los objetos de log
        "base": null,
// Para que no aparezca la propiedad "time" en los logs
        "timestamp": false,
// Cambiar la key "msg" a "message" cuando estamos haciendo un log de un string
        "messageKey": "message",
// Cambiar la key "level" a "priority" de los logs
        "changeLevelName": "priority"
    }
}
```

Transports

- Finalmente, ahora que ya tenemos los logs gestionados así como necesitamos, ahora tenemos que ver qué hacemos con ellos.

- Para ello usaremos los “transports” de pino, se puede ver un listado aquí: https://getpino.io/#/docs/transports?id=known-transports.

- Ejemplo con https://github.com/pinojs/pino-pretty:
  - npm install --save-dev pino-pretty
  - En package.json/scripts:
    - "start:dev": "npx plug-cli start | pino-pretty"
  - Ejecutamos *npm run start:dev*

```
> npx plug start | npx pino-pretty -t -i pid,hostname,name

[2020-04-06 09:59:12.058 +0000] INFO : My test log
```

GRACIAS POR SU TIEMPO

Para más información
info@plugcore.com
germanml@plugcore.com
sergiolc@plugcore.com

plugcore.com