COMANDO PARA EL ANÁLISIS DE RENDIMIENTO

```
(en la consola de powerSheel)
node -prof server.js
(en la console de Git Bash)
artillery quick --count 20 -n 50 "http://localhost:8080/info" > result bloq.txt
(en la consola de powerSheel)
```

node -prof server.js

(en la console de Git Bash)

artillery quick --count 20 -n 50 "http://localhost:8080/info/logger" > result_noblog.txt

DECODIFICAR ARCHIVOS LOG

node --prof-process bloq-v8.log > result prof bloq.txt node --prof-process nobloq-v8.log > result prof nobloq.txt

RESULTADO DEL ARCHIVO NO BLOQUEANTE

```
[Summary]:
 ticks total nonlib name
        0.3% 100.0% JavaScript
        0.0% 0.0% C++
        0.3% 108.3% GC
   13
       99.7%
                     Shared libraries
```

RESULTADO DEL ARCHIVO BLOQUEANTE

```
[Summary]:
 ticks total nonlib name
        0.1% 100.0% JavaScript
        0.0% 0.0% C++
        0.3% 320.0% GC
   16
       99.9%
                     Shared libraries
 5916
```

AHORA UTILIZAREMOS INSPECT

node --inspect profiling.js

LUEGO ABRIMOS CON CHROME LA SIGUIENTE DIRECCIÓN

chrome://inspect

```
artillery quick --count 20 -n 50 "http://localhost:8080/info" > result bloq.txt
artillery quick --count 20 -n 50 "http://localhost:8080/info/logger" > result_noblog.txt
```

```
Conexión Consola Profiler
                             Fuentes Memoria
                                                                                           - $
>>
               ■ node:internal/crypto/pbkdf2
                                              logger.js X
                                   chrow new Typechron( Musc be a number );
Node.js: file:
                132
                 133
                                 if (number < 0) {
                 134
                                   throw new RangeError('Must be >= 0');
                 135
                 136
                                 this.callStackSkipIndex = number;
                 137
                 138
                               log(level, ...args) {
  const logLevel = levels.getLevel(level);
                 139
                        0.1 ms
                 140
                     0.5 ms
                 141
                                 if (!logLevel) {
                                   if (configuration.validIdentifier(level) && args.length > 0) {
                 142
                 143
                                     // logLevel not found but of valid signature, WARN before fal:
                 144
                                     this.log(
                 145
                                       levels.WARN,
                 146
                                       'log4js:logger.log: valid log-level not found as first parar
                 147
                                       level
                 148
                 149
                                     this.log(levels.INFO, `[${level}]`, ...args);
                 150
                                   } else {
                 151
                                     // apart from fallback, allow .log(...args) to be synonym with
                 152
                                     this.log(levels.INFO, level, ...args);
                        0.1 ms
                 153
                 154
                        0.5 ms
                                 } else if (this.isLevelEnabled(logLevel)) {
                 155
                       1.6 ms
                                   this._log(logLevel, args);
                 156
                                 -}
                 157
                 158
                 159
                               isLevelEnabled(otherLevel) {
                                return this.level.isLessThanOrEqualTo(otherLevel);
                 160 1.7 ms
                 161
                 162
                 163
                               _log(level, data) {
                       1.4 ms
                 164
                                 debug(`sending log data (${level}) to appenders`);
                 165
                        1.7 ms
                                 const error = data.find((item) => item instanceof Error);
                 166
                                 let callStack;
                 167
                       0.6 ms
                                 if (this.useCallStack) {
                 168
                 169
                                     if (error) {
                 170
                                       callStack = this.parseCallStack(
                 171
                172
                                         this.callStackSkipIndex + baseCallStackSkip
                173
                                       );
                 174
               4
Cobertura: no disponible
II 🙃 🕆 🕆 📂 🛈
                                                   Alcance Seguimiento
▼ Puntos de interrunción
```

MODIFICAMOS EL PACKAGE. JSON

```
Debug
"scripts": {
    "test": "node benchmark.js",
    "start": "0x profiling.js"
},
```

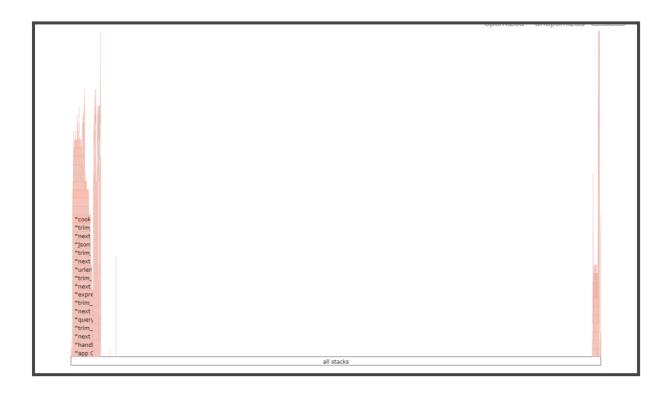
Y EJECUTAMOS LOS SIGUIENTES COMANDOS

(powershell)

npm start

(git bash)

npm test



En conclusión el uso de los logger al ser sincrónicos (no bloqueantes nos ayuda en el rendimiento de nuestra aplicación) si bien la diferencia no es muy grande la cantidad de procesos es menor (eso no sabemos por --prof-process) y tambien sabemos que el tiempo de ejecución es menor (gracias a lo que vemos en el diagrama flama con 0x)