Desafío 14:

Logs, debug y profiling

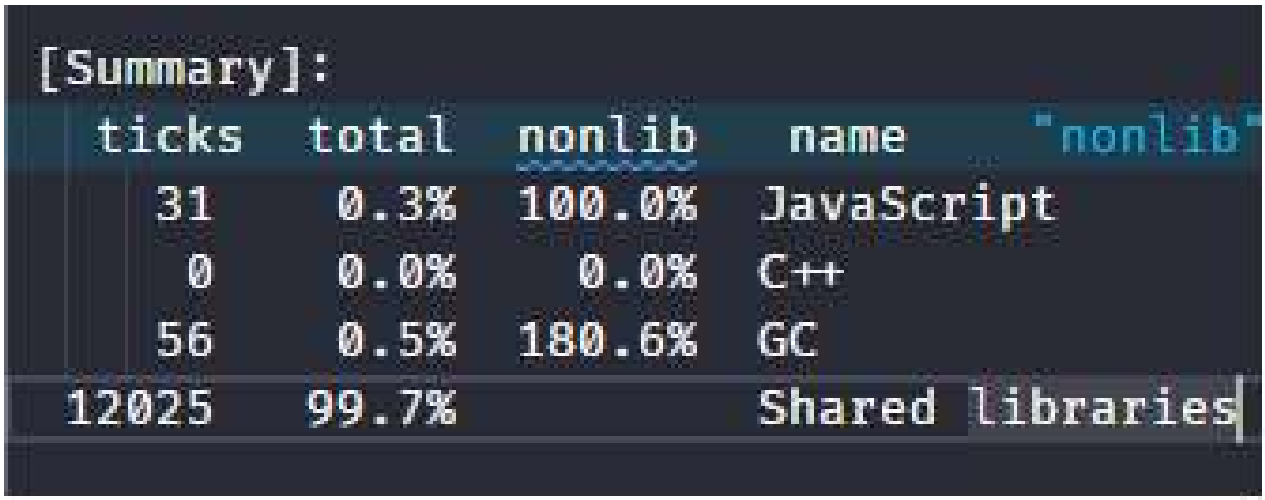
Loggers:

```
[LOGGER] - 22-06-22 16:45:59 - [warn] - Se esta intentando acceder a una ruta inexistente
[LOGGER] - 22-06-22 17:33:42 - [warn] - Se esta intentando acceder a una ruta inexistente
[LOGGER] - 22-06-22 17:39:25 - [warn] - Se esta intentando acceder a una ruta inexistente
```

Los loggers son herramientas que nos facilitan a la hora de llevar a producción nuestro proyecto, si es verdad que lo podemos usar en modo de desarrollo, pero se implementan mas en modo producción, ya que es la manera de poder dejar registrados los eventos que ocurren en nuestro servidor.

Destacándose los eventos del tipo INFO / WARN / ERROR, que por lo general son los mas usados.

Prof:



Con el archivo generado por la herramienta nativa de profiling de node, podemos ver, como luego de ejecutar un prof-process (que nos deja un archivo mas digerible). Analizando el resultado luego del procesamiento, el archivo generado en un archivo con console.log es mas lento en su generación que el que no lo tiene.

Autocannon:

```
# of samples: 50

5k requests in 50.32s, 14.6 MB read
PS C:\Users\marti\Documents\CoderHouse\Programacion Backend\14mo entregable> autocannon -d 20 -c 50 "http://localhost:8080/info"
Running 20s test @ http://localhost:8080/info
50 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	142 ms	572 ms	1408 ms	1806 ms	577.58 ms	332.65 ms	2239 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	10	10	94	108	84.05	25.3	10
Bytes/Sec	30.3 kB	30.3 kB	285 kB	327 kB	254 kB	76.6 kB	30.3 kB

Analizando la cantidad de conexiones que se realizan al estar en modo fork, imagino que esto hace saturar el hilo, por lo cual a medida que va aumentando el proceso, aumenta el tiempo de respuesta.

0X:

0x nos muestra que los procesos están mas cerca de estar optimizados, si bien no tiene grandes picos, esto se debe a ser un proceso no bloqueante.

Inspect:

```
1   const { isAuthenticated } = require('../middlewares/authMiddleware.js');
2   const os = require('os');
3   const processRouter = require('express').Router();
4
5   //process argv
6
7   const args = process.argv.length > 2 ? process.argv.slice(2).join(',') : 'Null';
8
9   const options = [
10    {
11      argument: args,
12      so: process.platform,
13      cores: os.cpus().length,
14      versionNode: process.version,
15      memory: Math.round(process.memoryUsage().rss) / 1024,
16      execPath: process.execPath,
17      processID: process.pid,
18      projectPath: process.cwd(),
19    },
20  ];
21
22  processRouter.get('/info', (req, res, next) => {
23    3.4 ms  res.render('info', { options });
24  });
25
26  //random operation
27
28  module.exports = processRouter;
29
```

Inspect es una herramienta que luego de generar profiling, nos permite ver donde hay latencias altas en nuestro código a la hora de hacer ejecuciones. Por ejemplo aca vemos como muestra que luego de hacer el get, tarda 3.4ms en realizar el render de EJS.

Conclusión:

Podemos concluir que estas herramientas nos dan una forma visual y mas palpable, de como podemos mejorar nuestra app en cuanto a rendimiento, viendo como resolver bloques de código que tengan problemas de optimización. Y poder reducir costos y mejorar los resultados de cara a producción.