Consigna:

Realizar el análisis completo de performance del servidor con el que venimos trabajando.

Vamos a trabajar sobre la ruta './info', en modo fork, agregando y extrayendo un console.log de la información colectada antes de devolverla al cliente. Además, desactivaremos el child_process de la ruta '/randoms'.

Para ambas condiciones (con o sin console.log) en la ruta '/info' OBTENER:

1. El perfilamiento del servidor, realizando la prueba con --prof de node.js. Analizar los resultados obtenidos luego de procesarlos con --prof-process.

   Utilizaremos como test de carga Artillery en línea de comandos, emulando 50 conexiones concurrentes con 20 request por cada una. Extraer un reporte con los resultados en archivo de texto.

   Luego utilizaremos Autocannon en línea de comandos, emulando 100 conexiones concurrentes realizadas en un tiempo de 20 segundos. Extraer un reporte con los resultados (puede ser un print screen de la consola).

2. El perfilamiento del servidor con el modo inspector de node.js --inspect. Revisar el tiempo de los procesos menos performantes sobre el archivo fuente de inspección.

3. El diagrama de flama con 0x, emulando la carga con Autocannon con los mismos parámetros anteriores.

Realizar un informe en formato pdf sobre las pruebas realizadas incluyendo los resultados de todos los test (texto e imágenes).

Resolución.

```
const argsv: any = process.argv;
const args = argsv.splice(2, argsv.length).join(" - ");
const memoria: any = Object.entries(process.memoryUsage());
const memoAux = memoria.map((memo: any) => `${memo[0]}: ${memo[1]}`);
const memoriaString = memoAux.join("  -  ");
const numCPUs = os.cpus().length

const datos = {
    argumentos: args,
    plataforma: process.platform,
    nodeVersion: process.version,
    memoriaUso: memoriaString,
    path: process.argv[1],
    pid: process.pid,
    carpeta: process.cwd(),
    numCPUs: numCPUs
};

app.get("/info", (_, res) => {

    //-------------------------
    //MODIFIFCAR SEGUN TEST
    //-------------------------
    console.log(datos);

    //-------------------------
    res.render("process", {
        datos,
        btnAction: "/home",
        info: true
    })
});
```

1.

```
E:\CODERHOUSE\4_backend\desafiosGitHub\nodeServer>node --prof ./dist/main.js 8080 423519862624165 de42abdb2f8e3917e10682d189668d1f
[2021-11-13T14:34:40.609] [INFO] info - Base de datos MongoDBAaS conectada!
[2021-11-13T14:34:40.662] [INFO] info - Servidor listo en el puerto 8080
```

Con console.log()

```
E:\CODERHOUSE\4_backend\desafiosGitHub\nodeServer>artillery quick -c 50 -n 20 "http://localhost:8080/info" > artillery_prof_fork_ccl.txt
```

```
E:\CODERHOUSE\4_backend\desafiosGitHub\nodeServer>node --prof-process node-ccl-punto1.log > node-ccl-punto1.txt
```

```
E:\CODERHOUSE\4_backend\desafiosGitHub\nodeServer>autocannon -d 20 -c 100 "http://localhost:8080/info"
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	340 ms	409 ms	698 ms	706 ms	436.85 ms	91.62 ms	761 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	0	0	0	285	49.85	94.86	100
Bytes/Sec	0 B	0 B	0 B	656 kB	115 kB	218 kB	230 kB

```
Req/Bytes counts sampled once per second.

47k requests in 20.14s, 2.3 MB read
46k errors (0 timeouts)
```

Sin console.log()

```
E:\CODERHOUSE\4_backend\desafiosGitHub\nodeServer>artillery quick -c 50 -n 20 "http://localhost:8080/info" > artillery_prof_fork_scl.txt
```

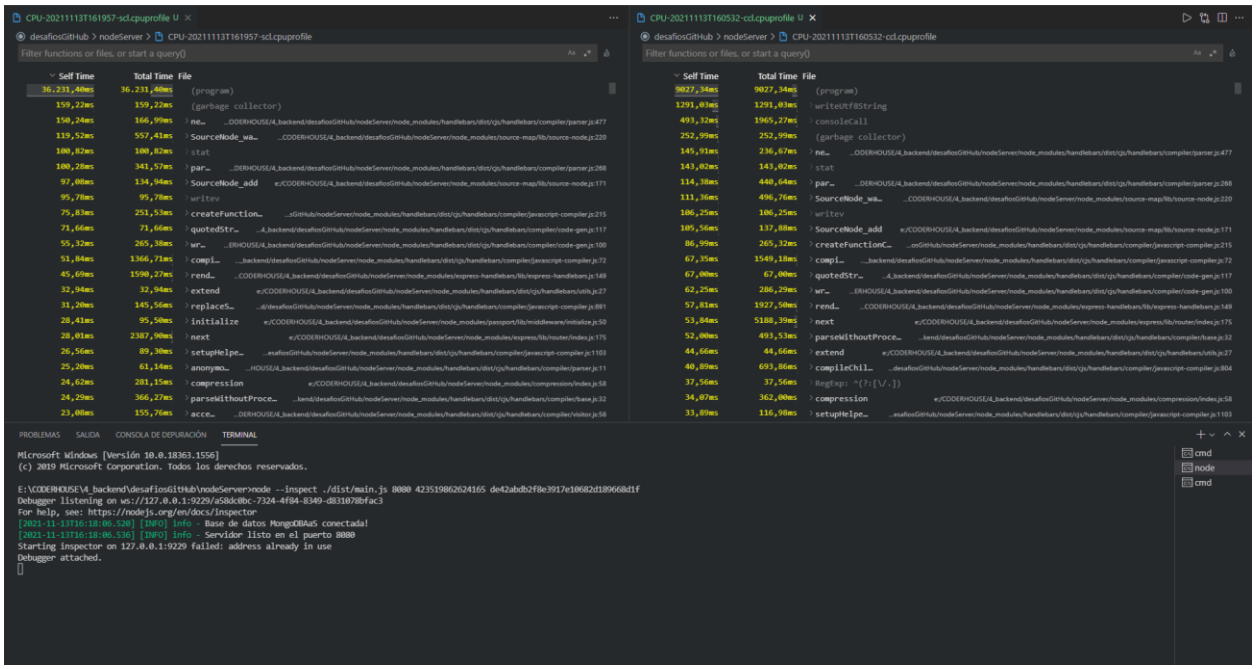
```
E:\CODERHOUSE\4_backend\desafiosGitHub\nodeServer>node --prof-process node-scl-punto1.log > node-scl-punto1.txt
```

```
E:\CODERHOUSE\4_backend\desafiosGitHub\nodeServer>autocannon -d 20 -c 100 "http://localhost:8080/info"
Running 20s test @ http://localhost:8080/info
100 connections

| Stat | 2.5% | 50% | 97.5% | 99% | Avg | Stdev | Max |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Latency | 173 ms | 207 ms | 441 ms | 479 ms | 222.51 ms | 54.43 ms | 504 ms |

| Stat | 1% | 2.5% | 50% | 97.5% | Avg | Stdev | Min |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Req/Sec | 0 | 0 | 0 | 528 | 172.4 | 220.19 | 200 |
| Bytes/Sec | 0 B | 0 B | 0 B | 1.22 MB | 397 kB | 507 kB | 461 kB |

Req/Bytes counts sampled once per second.

37k requests in 20.12s, 7.94 MB read
34k errors (0 timeouts)
```

2.

```
E:\CODERHOUSE\4_backend\desafiosGitHub\nodeServer>node --inspect ./dist/main.js 8080 423519862624165 de42abdb2f8e3917e10682d189668d1f
Debugger listening on ws://127.0.0.1:9229/dbdcd9bc-8033-4c46-bc80-e4179ce238af
For help, see: https://nodejs.org/en/docs/inspector
[2021-11-13T15:59:52.979] [INFO] info - Base de datos MongoDBAaS conectada!
[2021-11-13T15:59:53.003] [INFO] info - Servidor listo en el puerto 8080
Starting inspector on 127.0.0.1:9229 failed: address already in use
Debugger attached.
```

Sin console.log (izq.) – Con console.log (derecha)



3.

```
E:\CODERHOUSE\4 backend\desafiosGitHub\nodeServer>0x ./dist/main.js 8080 423519862624165 de42abdb2f8e3917e10682d189668d1f
Profiling[2021-11-14T12:51:01.449] [INFO] info - Base de datos MongoDBAaS conectada!
[2021-11-14T12:51:01.497] [INFO] info - Servidor listo en el puerto 8080
```

Con console.log()

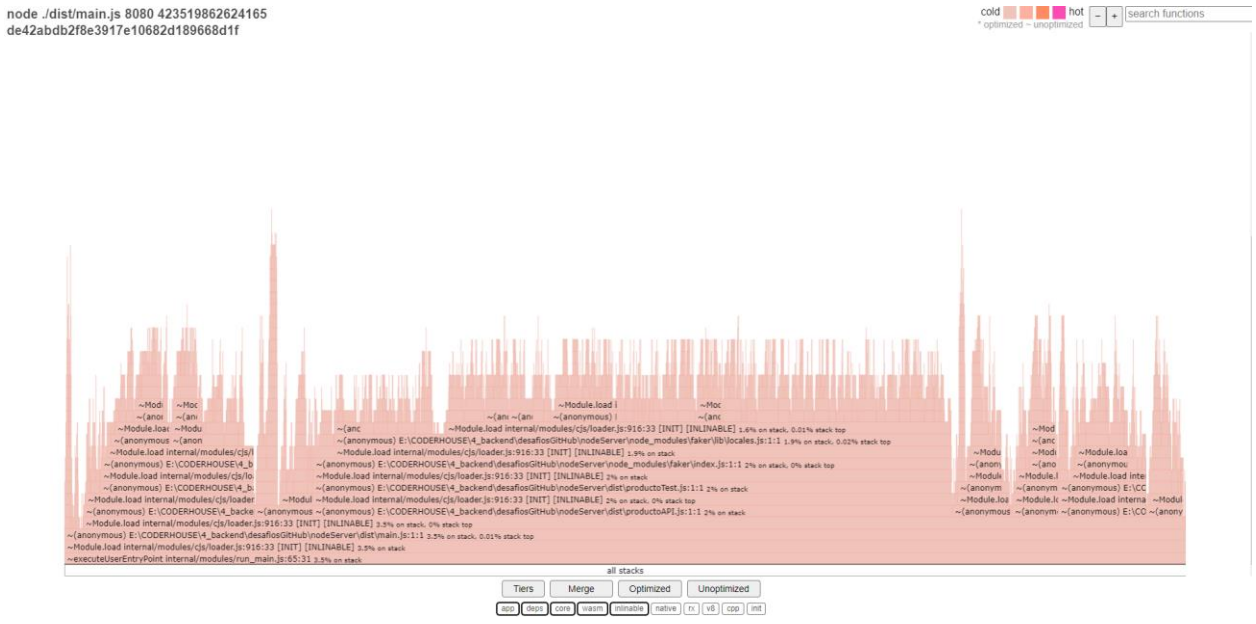
```
E:\CODERHOUSE\4_backend\desafiosGitHub\nodeServer>autocannon -d 20 -c 100 "http://localhost:8080/info"
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	225 ms	281 ms	790 ms	813 ms	322.82 ms	127.47 ms	868 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	0	0	0	388	85.05	140.28	100
Bytes/Sec	0 B	0 B	0 B	894 kB	196 kB	323 kB	230 kB

```
Req/Bytes counts sampled once per second.

39k requests in 20.29s, 3.92 MB read
38k errors (0 timeouts)
```

8184.0x



Sin console.log()

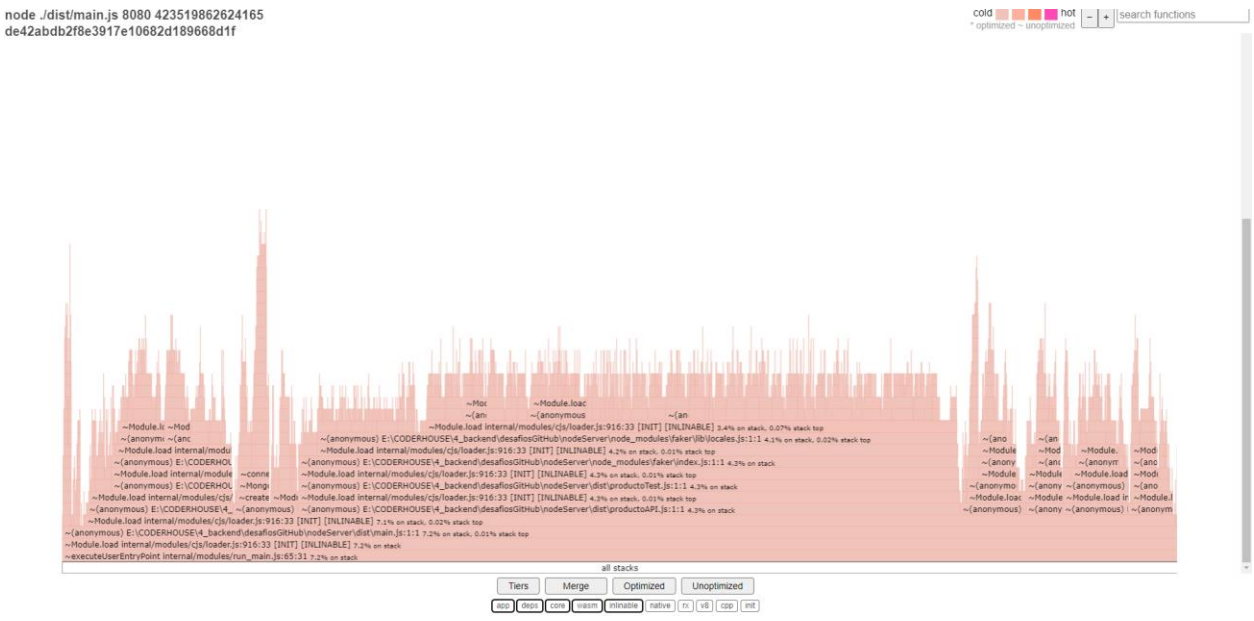
```
E:\CODERHOUSE\4_backend\desafiosGitHub\nodeServer>autocannon -d 20 -c 100 "http://localhost:8080/info"
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	161 ms	189 ms	500 ms	529 ms	204.32 ms	62.72 ms	562 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	0	0	0	594	164.5	234.29	200
Bytes/Sec	0 B	0 B	0 B	1.37 MB	379 kB	540 kB	461 kB

```
Req/Bytes counts sampled once per second.

40k requests in 20.14s, 7.58 MB read
37k errors (0 timeouts)
```

4156.0x



Conclusión

En todos los test se comprobó fehacientemente como el método Console.log() afecta el rendimiento de la aplicación. En particular el mejor método para determinar dicha diferencia fue el inspect de Chrome.