1

’El perfilamiento del servidor, realizando el test 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’

Con un console.log en la ruta /info

```
[Shared libraries]:
  ticks  total  nonlib   name
 62497   95.0%          C:\Windows\SYSTEM32\ntdll.dll
  3031    4.6%          C:\Program Files\nodejs\node.exe
     2    0.0%          C:\Windows\System32\KERNEL32.DLL
     1    0.0%          C:\Windows\System32\WS2_32.dll
     1    0.0%          C:\Windows\System32\KERNELBASE.dll

[JavaScript]:
```

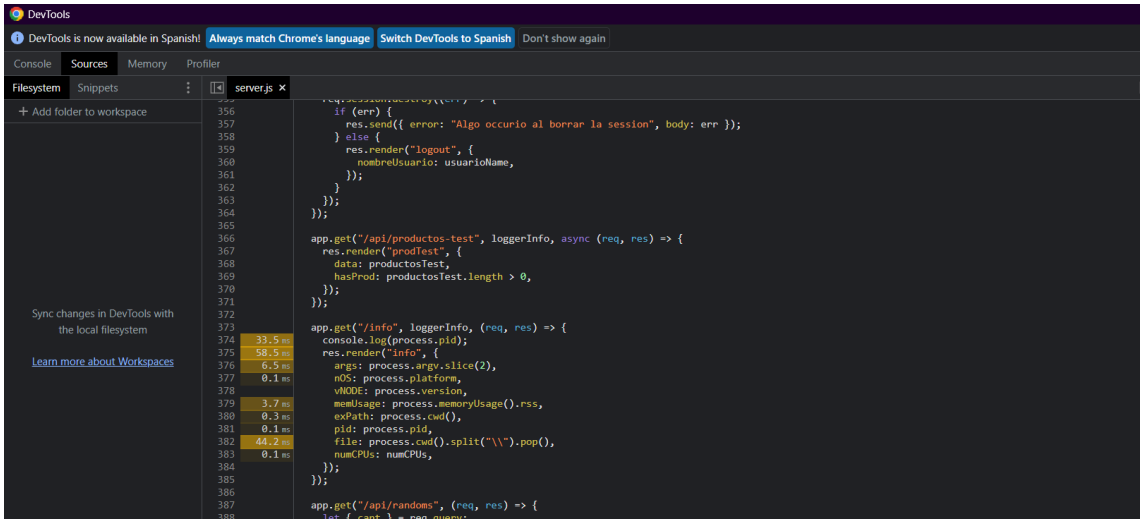
Sin un console.log en la ruta /info

```
[Shared libraries]:
  ticks  total  nonlib   name
 17491   92.0%          C:\Windows\SYSTEM32\ntdll.dll
  1424    7.5%          C:\Program Files\nodejs\node.exe
     3    0.0%          C:\Windows\System32\KERNELBASE.dll
     1    0.0%          C:\Windows\System32\KERNEL32.DLL
```

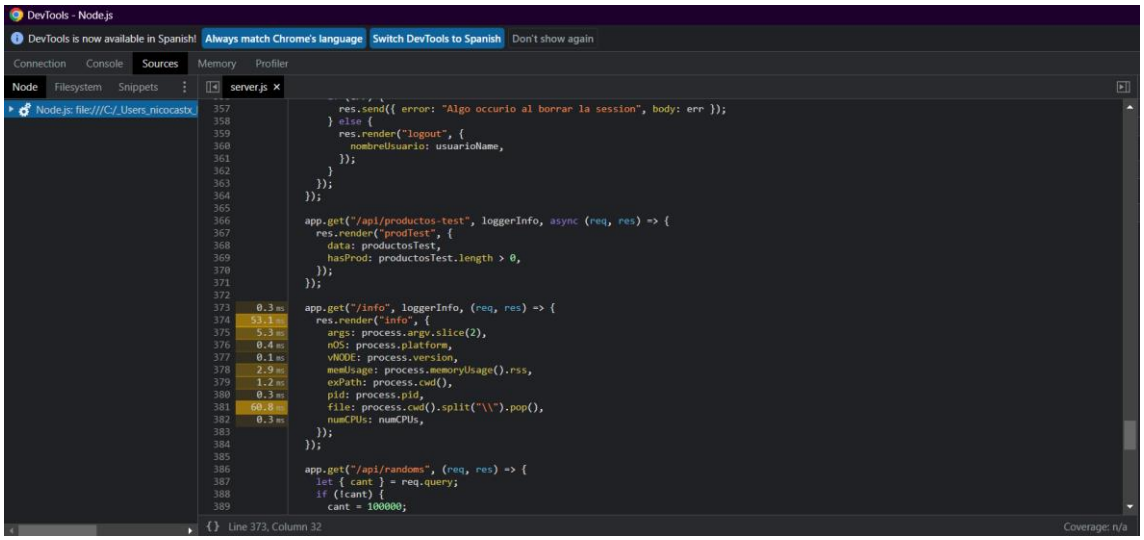
Cuando quitamos el console.log, la cantidad de ticks se reduce considerablemente en comparación a cuando no lo usamos

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.’

Con CLG



Sin CLG:



Podemos ver que uno de los procesos que mas me toman memoria al momento de cargar y por lo tanto, tardan mas tiempo en ejecutar son los procesos de redireccion y autenticación de usuario, los que me producen una tardanza significativa entre las peticiones al servidor

Otra de los procesos que me están ocupando mucho tiempo para ejecutar son los loggers, los cuales se ejecutan múltiples veces y el proceso en si de mostrar un resultado en consola ya es tardado en procesar

Reporte de autocannon:

No bloqueante

```
C:\Users\nicocastx\Desktop\cursoBackend\clasesBackend>npm test

> clasesbackend@1.0.0 test
> node benchmark.js

Autocannon corriendo
Running 20s test @ http://localhost:8080/
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	54 ms	80 ms	163 ms	181 ms	86.89 ms	28.73 ms	389 ms

```
C:\Users\nicocastx\Desktop\cursoBackend\clasesBackend>npm test
```

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	200	200	392	467	383.15	60.37	200
Bytes/Sec	316 kB	316 kB	620 kB	738 kB	605 kB	95.4 kB	316 kB

```
Req/Bytes counts sampled once per second.
# of samples: 20

8k requests in 20.16s, 12.1 MB read
```

Bloqueante

```
C:\Users\nicocastx\Desktop\cursoBackend\clasesBackend>npm test

> clasesbackend@1.0.0 test
> node benchmark.js

Autocannon corriendo
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	226 ms	312 ms	570 ms	905 ms	331.96 ms	102.4 ms	1028 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	99	99	304	377	300.15	64.95	99
Bytes/Sec	157 kB	157 kB	481 kB	596 kB	474 kB	103 kB	156 kB

```
Req/Bytes counts sampled once per second.
# of samples: 20

6k requests in 20.16s, 9.48 MB read
```

3

No bloqueante

Bloqueante

El diagrama de flama de mi servidor muestra que durante el lapso de tiempo de ejecución, podemos ver que muchos procesos que ejecutan al mismo tiempo, por lo que su altura en y del grafico aumenta considerablemente, sin embargo, ninguno de ellos es un proceso el cual ocupe mucho tiempo de ejecución, pero lo que mas se podría ver de optimizar es la distribución de y en el resto del tiempo de ejecución, esto se soluciona viendo de reducir la cantidad de funciones bloqueantes, y optimizarlas para que sean no bloqueantes

CONCLUSION:

Es importante intentar trabajar con funciones y procesos de manera asincrónica, de manera que ninguna trabe al tiempo de ejecución de la otra, esto es mucho mas notorio en proyectos muy grandes, ya que reduce la necesidad de llamados y tiempos de espera en los que no se hace nada