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.’

```
41          //#region Funciones
42          //#endregion Funciones
43          //#region Middlewares
44   1.2 ms function isAuth(req, res, next) {
45  27.2 ms   if (req.isAuthenticated()) {
46              next();
47            } else {
48  82.4 ms     res.redirect("/login");
49            }
50   0.7 ms }
51
52   1.6 ms function loggerInfo(req, res, next) {
53   9.4 ms   const { url, method } = req;
54  10.0 ms   logger.info(`Ingresando a la ruta ${url} metodo ${method}`);
55   1.0 ms   next();
56          }
57
58          //#endregion Middlewares
59          //#region Manejo de sockets
```

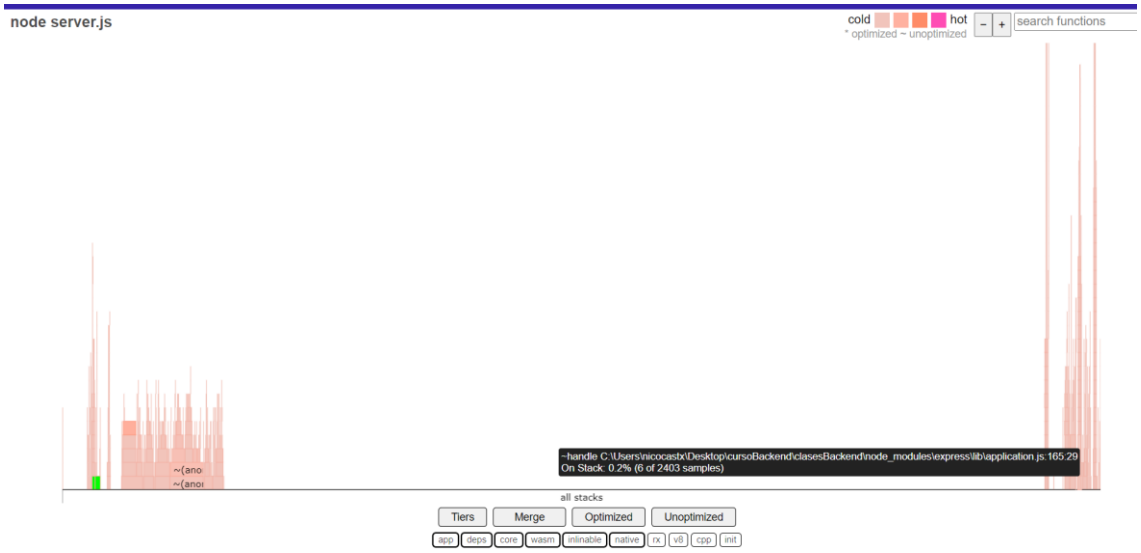
Podemos ver que uno de los procesos que mas me toman memoria al momento de cargar y por lo tanto, tardan mas tiempo en ejecutar son los procesos de redireccion y autenticación de usuario, los que me producen una tardanza significativa entre las peticiones al servidor

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:

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

> clasesbackend@1.0.0 test
> node benchmark.js

Req/Bytes counts sampled once per second.
Autocannon corriendo
Running 20s test @ http://localhost:8080/
100 connections


┌─────────┬──────┬──────┬───────┬──────┬──────────┬─────────┬────────┐
│ Stat    │ 2.5% │ 50%  │ 97.5% │ 99%  │ Avg      │ Stdev   │ Max    │
├─────────┼──────┼──────┼───────┼──────┼──────────┼─────────┼────────┤
│ Latency │ 185 ms │ 282 ms │ 471 ms │ 530 ms │ 282.67 ms │ 77.44 ms │ 586 ms │
└─────────┴──────┴──────┴───────┴──────┴──────────┴─────────┴────────┘

┌───────────┬─────┬──────┬──────┬──────┬──────┬──────┬──────┐
│ Stat      │ 1%  │ 2.5% │ 50%  │ 97.5% │ Avg  │ Stdev │ Min  │
├───────────┼─────┼──────┼──────┼──────┼──────┼──────┼──────┤
│ Req/Sec   │ 200 │ 200  │ 328  │ 496  │ 352.15 │ 86.9 │ 200  │
├───────────┼─────┼──────┼──────┼──────┼──────┼──────┼──────┤
│ Bytes/Sec │ 50 kB │ 50 kB │ 82 kB │ 124 kB │ 88 kB │ 21.7 kB │ 50 kB │
└───────────┴─────┴──────┴──────┴──────┴──────┴──────┴──────┘

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

0 2xx responses, 7043 non 2xx responses
7k requests in 20.11s, 1.76 MB read

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

3



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