

## Paso a paso para correr el profiling de la aplicación

Precondiciones: tener instalado artillery, 0x y autocannon de manera global

`npm i -g artillery`

`npm i -g autocannon`

`npm i -g 0x`

- `npm install`  
Este comando es para instalar todas las dependencias del proyecto
- `npm run build`  
Este comando es para buildear la aplicación que está hecha con typescript y lo transpila a javascript
- `npm run start:prof`  
Este comando corre un `"node --prof dist/index.js"`  
La opción `--prof` permite ejecutar una aplicación Node.js con el modo de profiling activado. Esto genera información de perfil en el archivo `"isolate-0x...-v8.log"` en el directorio actual que puede ser procesada y analizada para ayudar a identificar cuellos de botella en el rendimiento de la aplicación.
- `npm run artillery-without-console-log`  
Este comando corre `"artillery quick --count 50 -n 20 http://localhost:8080/api/info > artillery-without-console-log.txt"` para realizar un test con 50 conexiones concurrentes con 20 request por cada una y el resultado lo guarda en el archivo `artillery-without-console-log.txt`.
- El paso anterior genera un archivo `isolate-000002A39B685BE0-13828-v8.log` entonces lo que sigue hacer es procesarlo con el comando  
`node --prof-process isolate-000002A39B685BE0-13828-v8.log > prof-process-without-console-log.txt`
- Mato el proceso y repito a hacer los pasos anteriores pero con la version que loguea con `console.log`
  1. `npm run start:prof`
  2. `npm run artillery-with-console-log`
  3. Mato el proceso
  4. Me genera el archivo `isolate-0000015596EB9890-15868-v8.log` y lo proceso con  
`node --prof-process isolate-0000015596EB9890-15868-v8.log > prof-process-with-console-log.txt`

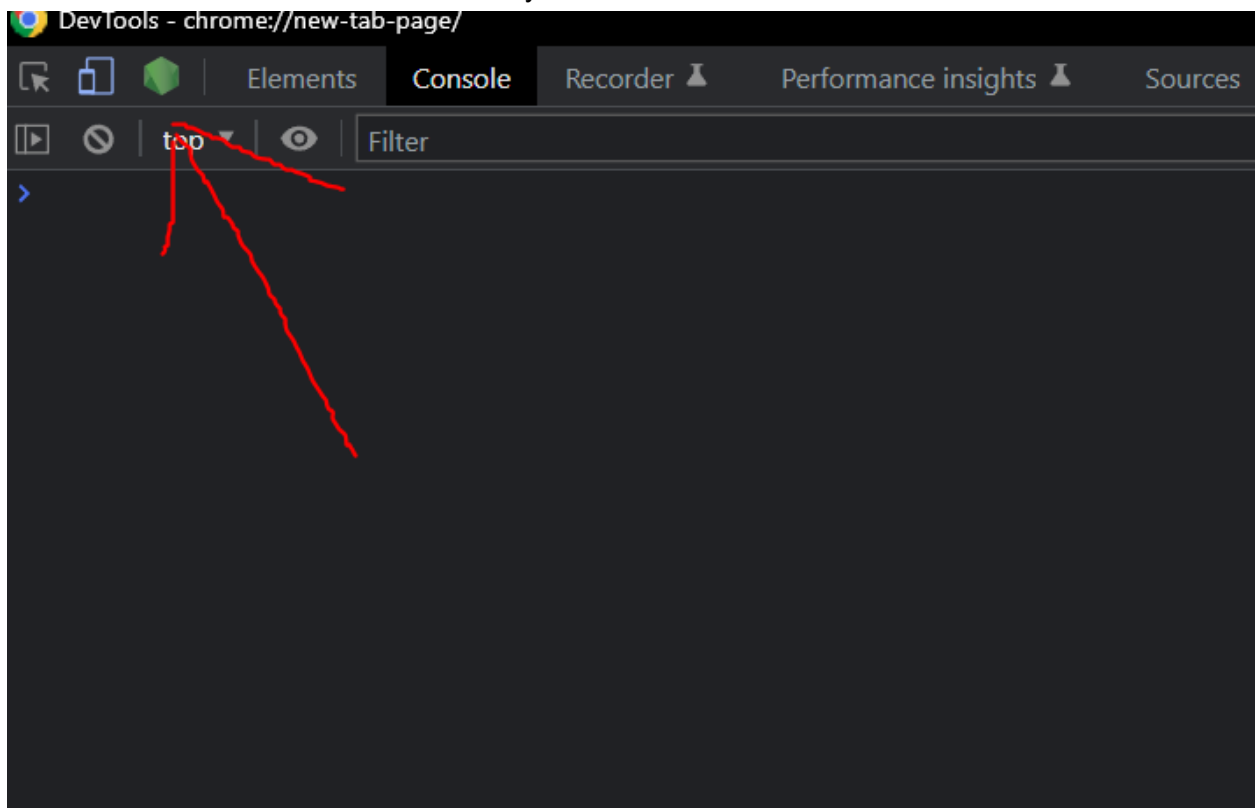
Top Screenshot: Statistical profiling result from isolate-000002A39B685BE0-13828-v8.log, (30274 ticks, 0 unaccounted for)

ticks	total	nonlib	name
29844	95.9%		C:\Windows\SYSTEM32\ntdll.dll
1209	4.0%		C:\Program Files\nodejs\node.exe
1	0.0%		C:\Windows\system32\mswsock.dll
1	0.0%		C:\Windows\System32\KERNELBASE.dll

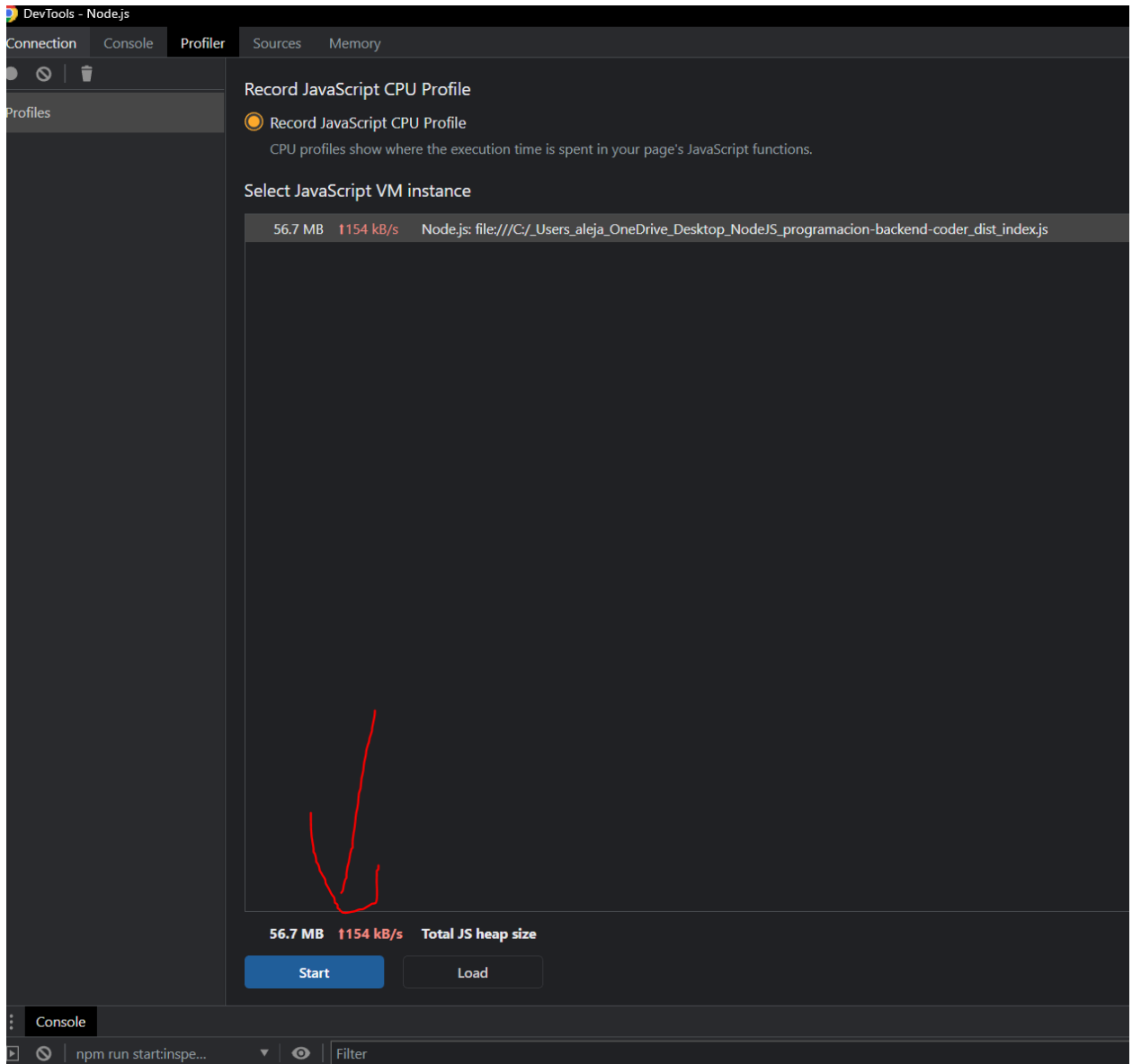
Bottom Screenshot: artillery-with-console-log.txt

metric	value
http.codes.200	122
http.request_rate	56/sec
http.requests	152
http.response_time	
min	14
max	1202
median	450.4
p95	620.3
p99	1043.3
http.responses	122
users.created	50
users.created_by_name.0	50

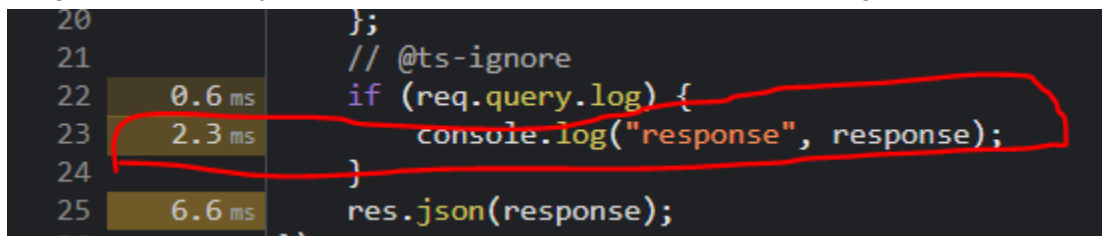
- npm run start:inspect  
este script ejecuta node --inspect dist/index.js para poder hacer el profile con chrome  
Para abrir en chrome abrir las dev tools y hacer click en el ícono verde



luego en start



y ejecutar las pruebas con `npm run autocannon-with-console-log` que es un script que va a ejecutar `autocannon -c 100 -d 20 http://localhost:8080/api/info?log=true` que lo que va a hacer es testear con 100 hilos concurrentes durante 20 segundos. Luego darle a Stop y podemos observar lo que tarda el console log.

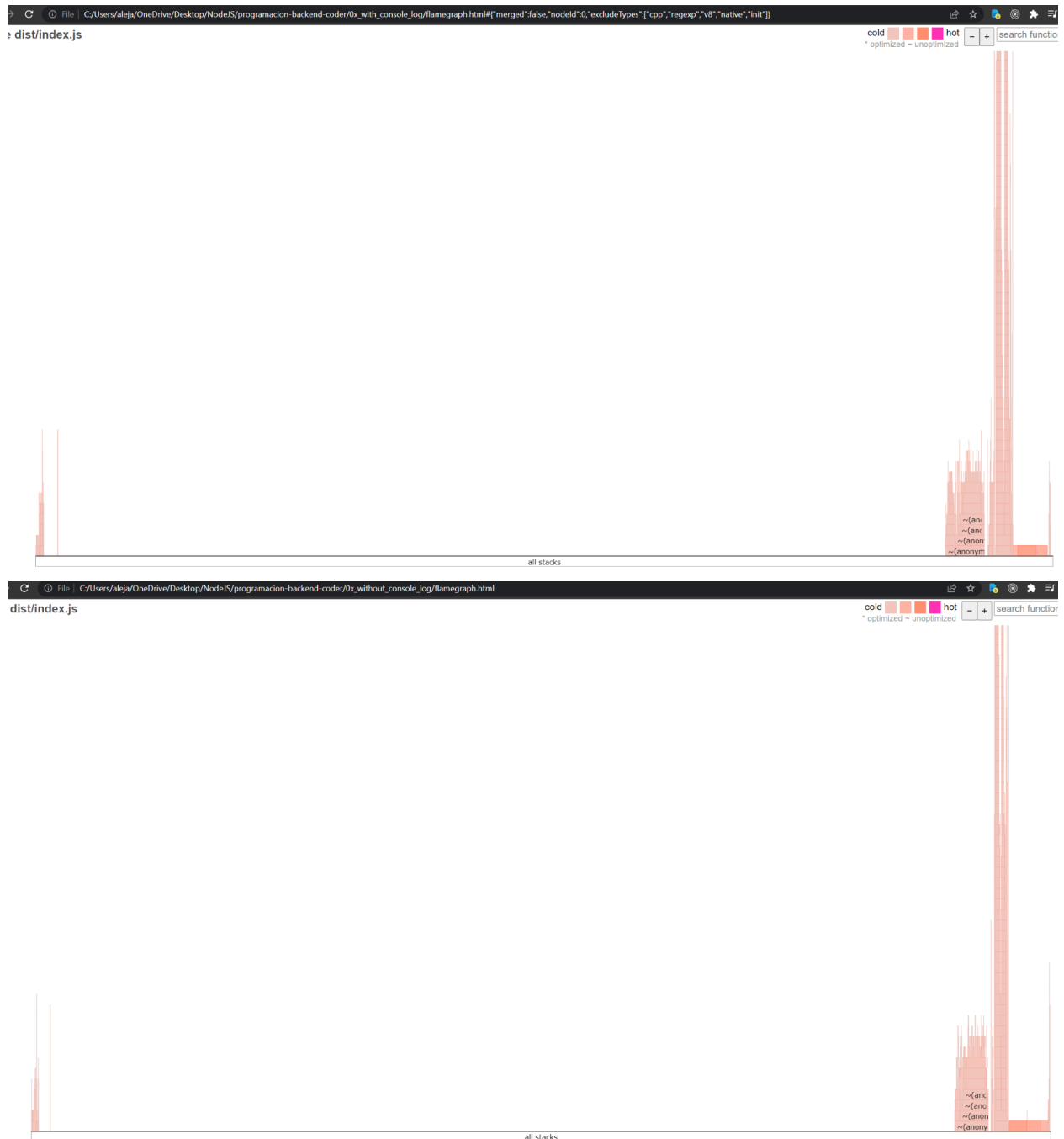


Steps

Console	Profiler	Sources	Memory
	Heavy (Bottom Up) ▼		
		Self Time	Total Time
		29430.0 ms	29430.0 ms
		14794.9 ms 75.73 %	14794.9 ms 75.73 %
			Function
			(idle)
			run
Save		1656.4 ms 8.48 %	2463.9 ms 12.61 %
		655.2 ms 3.35 %	655.2 ms 3.35 %
		172.4 ms 0.88 %	172.4 ms 0.88 %
		72.9 ms 0.37 %	72.9 ms 0.37 %
		53.4 ms 0.27 %	53.4 ms 0.27 %
		40.9 ms 0.21 %	64.1 ms 0.33 %
		40.5 ms 0.21 %	17302.9 ms 88.57 %
			initialize

- npm run start0x  
este script va a correr 0x dist/index.js
- npm run autocannon-without-console-log y npm run autocannon-with-console-log  
este script va a ejecutar autocannon

autocannon-with-console-log.txt M	autocannon-without-console-log.txt U	autocannon-without-console-log.txt -- autocannon-with-console-log.txt M
1- > autocannon -c 100 -d 20 http://localhost:8080/api/info	1- > autocannon -c 100 -d 20 http://localhost:8080/api/info?log=true	1- > autocannon -c 100 -d 20 http://localhost:8080/api/info?log=true
2	2	2
3- Running 20s test @ http://localhost:8080/api/info	3- Running 20s test @ http://localhost:8080/api/info?log=true	3- Running 20s test @ http://localhost:8080/api/info?log=true
4 100 connections	4 100 connections	4 100 connections
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
10	10	10
11	11	11
12	12	12
13	13	13
14	14	14
15	15	15
16	16	16
17	17	17
18	18	18
19	19	19
20 Req/Bytes counts sampled once per second.	20 Req/Bytes counts sampled once per second.	20 Req/Bytes counts sampled once per second.
21 # of samples: 20	21 # of samples: 20	21 # of samples: 20
22	22	22
23 1k requests in 20.11s, 919 kB read	23 1k requests in 20.12s, 883 kB read	23 1k requests in 20.12s, 883 kB read
24	24	24



El gráfico de flema refleja la carga del CPU en el tiempo. El eje horizontal representa el tiempo mientras que el eje vertical representa el call stack. En los espacios en blanco el procesador estuvo libre.

## Conclusión

La versión con console log resultó en 29044 ticks lo que representa más del triple de ticks comparado con los 7754 ticks de la versión sin console log. Cuantos más ticks tenga significa que el procesador estuvo más ocupado. Según los resultados de Artillery, la versión sin console log procesa 57 requests/segundo mientras que la versión con console log procesa 54 requests

por segundo lo que evidencia que es más eficiente la versión sin console log. Los resultados de Autocannon dan que la versión con console log tiene mayor latencia y el promedio de request por segundos da más bajo comparado con la versión sin console log.