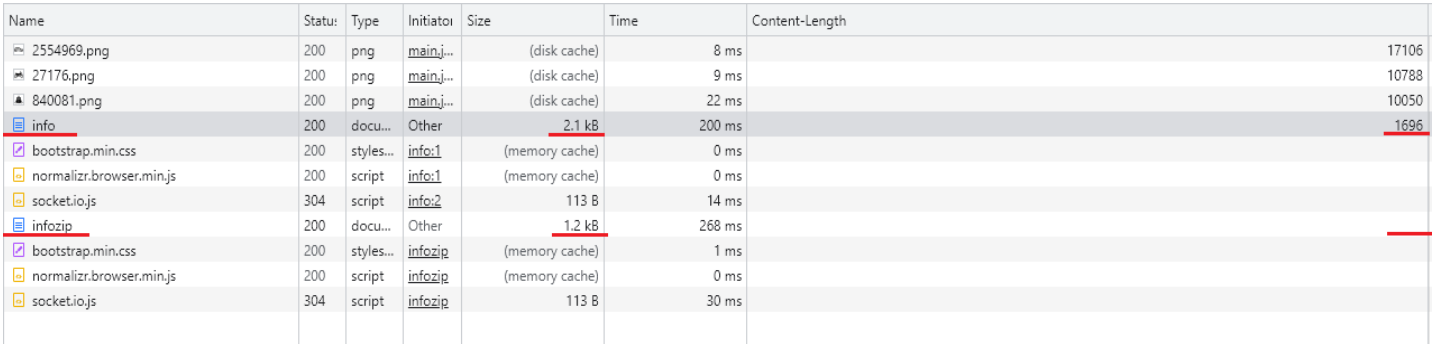
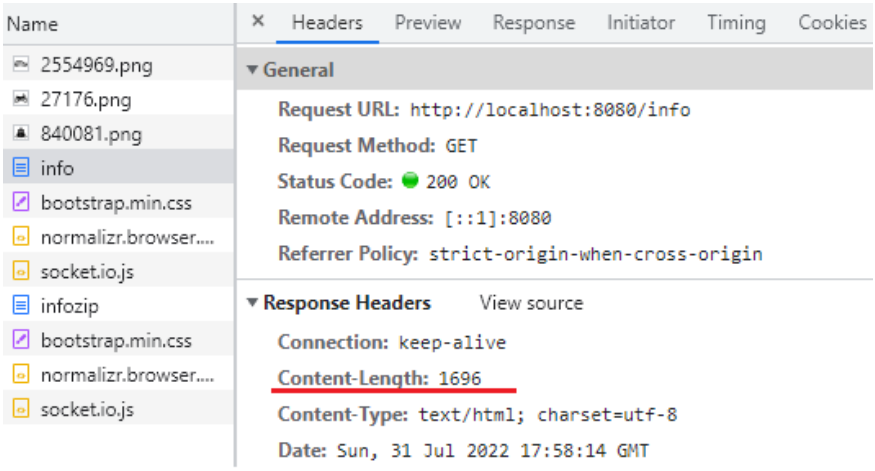
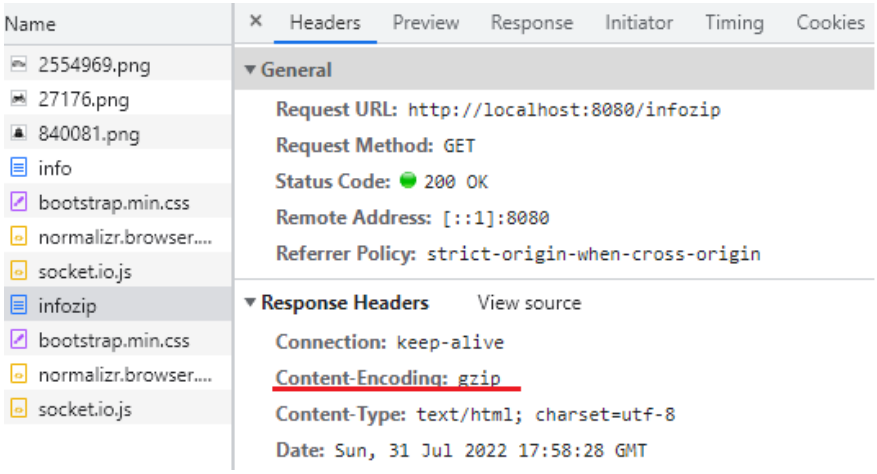
Gzip

1. Se realizo la prueba de la ruta /info con la librería compression, al tener el middleware se observa que ocupa menos espacio como se puede observar en la siguiente imagen, donde /infozip tiene el middleware y /info no lo tiene:



Name	Statu	Type	Initiator	Size	Time	Content-Length
2554969.png	200	png	main.j...	(disk cache)	8 ms	17106
27176.png	200	png	main.j...	(disk cache)	9 ms	10788
840081.png	200	png	main.j...	(disk cache)	22 ms	10050
info	200	docu...	Other	2.1 kB	200 ms	1696
bootstrap.min.css	200	styles...	info:1	(memory cache)	0 ms	
normalizr.browser.min.js	200	script	info:1	(memory cache)	0 ms	
socket.io.js	304	script	info:2	113 B	14 ms	
infozip	200	docu...	Other	1.2 kB	268 ms	
bootstrap.min.css	200	styles...	infozip	(memory cache)	1 ms	
normalizr.browser.min.js	200	script	infozip	(memory cache)	0 ms	
socket.io.js	304	script	infozip	113 B	30 ms	





Análisis Performance

1. Artillery: node –prof server.js FORK

```
$ node --prof server.js FORK
{"level":"info","message":"Iniciando en el puerto: 8080 modo:FORK pid:9540"}
{"level":"info","message":"Ruta /login, Metodo GET"}
Conexión a MongoDB correcta
```

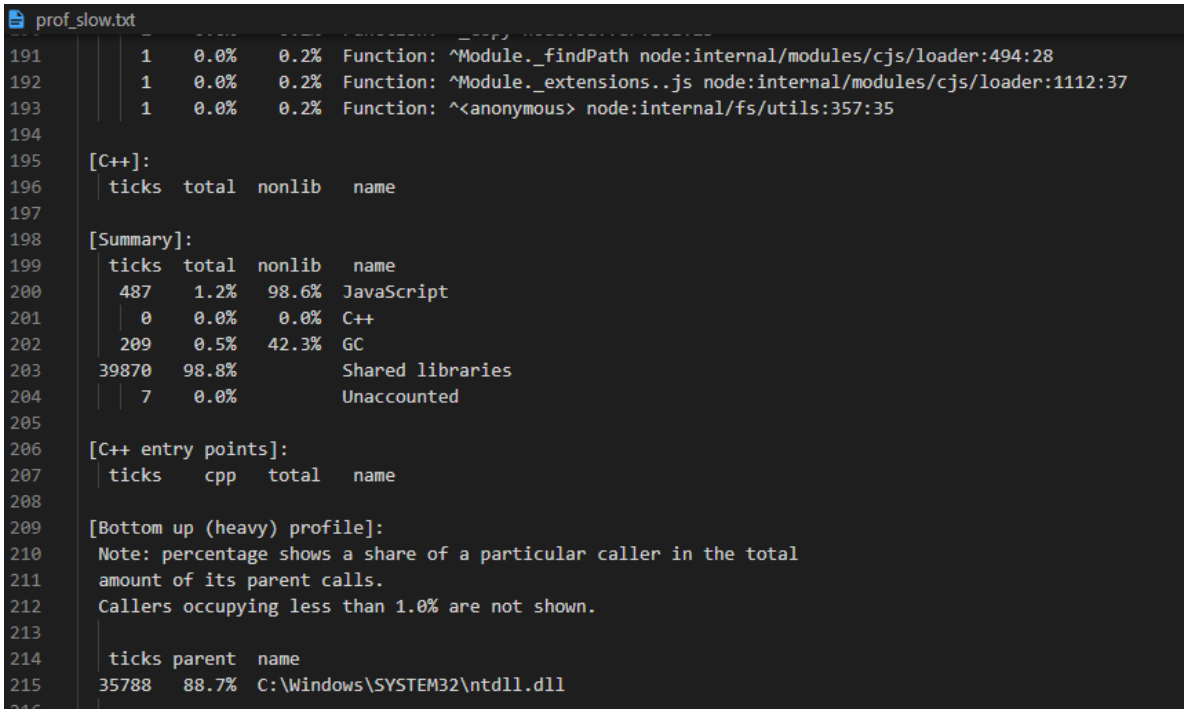
En /infodebug es donde tenemos lo de /info pero adicional mostramos información por console.log(), por lo cual ejecutamos lo siguiente: artillery quick -c 50 -n 20 "http://localhost:8080/infodebug" > artillery_slow.txt
Con -c 50 = 50 conexiones y -n 20 = 20 request

Después de ejecutar el comando en una segunda consola, se verán los console.log() de cada request:

```
{"level":"info","message":"Ruta /infodebug, Metodo GET"}
{
  datos: [
    [ 'FORK' ],
    'C:\\Program Files\\nodejs\\node.exe',
    15876,
    'C:\\Users\\duvam\\Documents\\Coder House backend\\desafio_loggers_gzip\\Loggers_gzip',
    144920576,
    'win32',
    'v16.14.1',
    4
  ]
}
{"level":"info","message":"Ruta /infodebug, Metodo GET"}
{
  datos: [
    [ 'FORK' ],
    'C:\\Program Files\\nodejs\\node.exe',
    15876,
    'C:\\Users\\duvam\\Documents\\Coder House backend\\desafio_loggers_gzip\\Loggers_gzip',
    144920576,
    'win32',
    'v16.14.1',
    4
  ]
}
{"level":"info","message":"Ruta /infodebug, Metodo GET"}
```

Luego le cambiamos el nombre del archivo que creo y tiene nombre de isolate-…. Por slow-v8.log, luego corremos en la segunda consola: node --prof-process slow-v8.log > prof_slow.txt

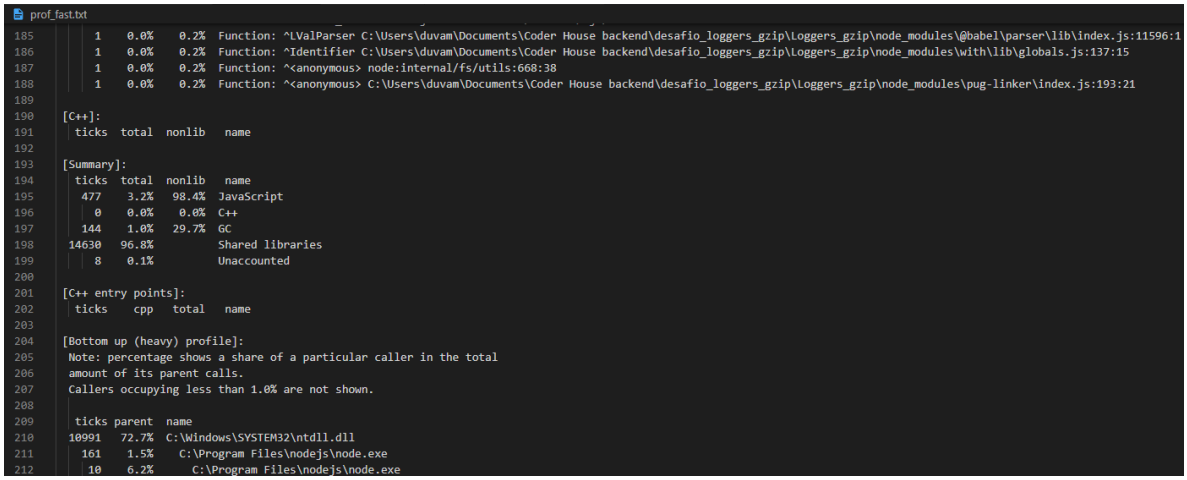
Abrimos el archivo prof_slow.txt y encontramos lo siguiente:



```
prof_slow.txt
191       1    0.0%    0.2%  Function: ^Module._findPath node:internal/modules/cjs/loader:494:28
192       1    0.0%    0.2%  Function: ^Module._extensions..js node:internal/modules/cjs/loader:1112:37
193       1    0.0%    0.2%  Function: ^<anonymous> node:internal/fs/utils:357:35
194
195   [C++]:
196     ticks  total  nonlib   name
197
198   [Summary]:
199     ticks  total  nonlib   name
200       487    1.2%   98.6%  JavaScript
201         0    0.0%    0.0%  C++
202       209    0.5%   42.3%  GC
203     39870   98.8%          Shared libraries
204         7    0.0%          Unaccounted
205
206   [C++ entry points]:
207     ticks    cpp   total   name
208
209   [Bottom up (heavy) profile]:
210     Note: percentage shows a share of a particular caller in the total
211     amount of its parent calls.
212     Callers occupying less than 1.0% are not shown.
213
214     ticks parent  name
215     35788   88.7%  C:\Windows\SYSTEM32\ntdll.dll
```

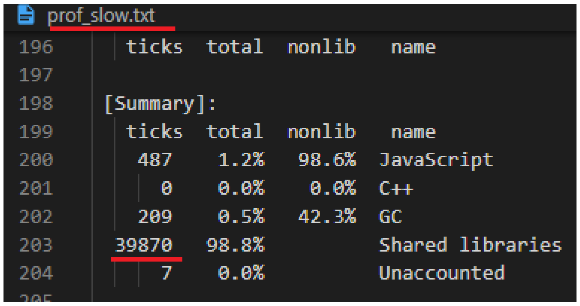
Ahora hacemos el mismo proceso para la ruta /info que no tiene el console.log(): artillery quick -c 50 -n 20 "http://localhost:8080/info" > artillery_fast.txt
Luego le cambiamos el nombre del archivo que creo y tiene nombre de isolate-…. Por fast-v8.log, luego corremos en la segunda consola: node --prof-process fast-v8.log > prof_fast.txt

Abrimos el archivo prof_fast-v8.txt y encontramos lo siguiente:

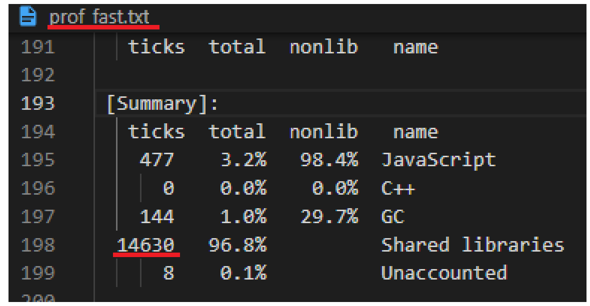


```
prof_fast.txt
185       1    0.0%    0.2%  Function: ^LValParser C:\Users\duvam\Documents\Coder House backend\desafio_loggers_gzip\Loggers_gzip\node_modules\@babel\parser\lib\index.js:11596:1
186       1    0.0%    0.2%  Function: ^Identifier C:\Users\duvam\Documents\Coder House backend\desafio_loggers_gzip\Loggers_gzip\node_modules\with\lib\globals.js:137:15
187       1    0.0%    0.2%  Function: ^<anonymous> node:internal/fs/utils:668:38
188       1    0.0%    0.2%  Function: ^<anonymous> C:\Users\duvam\Documents\Coder House backend\desafio_loggers_gzip\Loggers_gzip\node_modules\pug-linker\index.js:193:21
189
190   [C++]:
191     ticks  total  nonlib   name
192
193   [Summary]:
194     ticks  total  nonlib   name
195       477    3.2%   98.4%  JavaScript
196         0    0.0%    0.0%  C++
197       144    1.0%   29.7%  GC
198     14630   96.8%          Shared libraries
199         8    0.1%          Unaccounted
200
201   [C++ entry points]:
202     ticks    cpp   total   name
203
204   [Bottom up (heavy) profile]:
205     Note: percentage shows a share of a particular caller in the total
206     amount of its parent calls.
207     Callers occupying less than 1.0% are not shown.
208
209     ticks parent  name
210     10991   72.7%  C:\Windows\SYSTEM32\ntdll.dll
211       161    1.5%    C:\Program Files\nodejs\node.exe
212       10     6.2%      C:\Program Files\nodejs\node.exe
```

Al validar se evidencia que el proceso que es más rápido es el que no tiene el console.log(), en este caso el archivo prof_fast.txt el cual se generó a partir de la ruta /info, la comparación de los dos archivos es la siguiente:



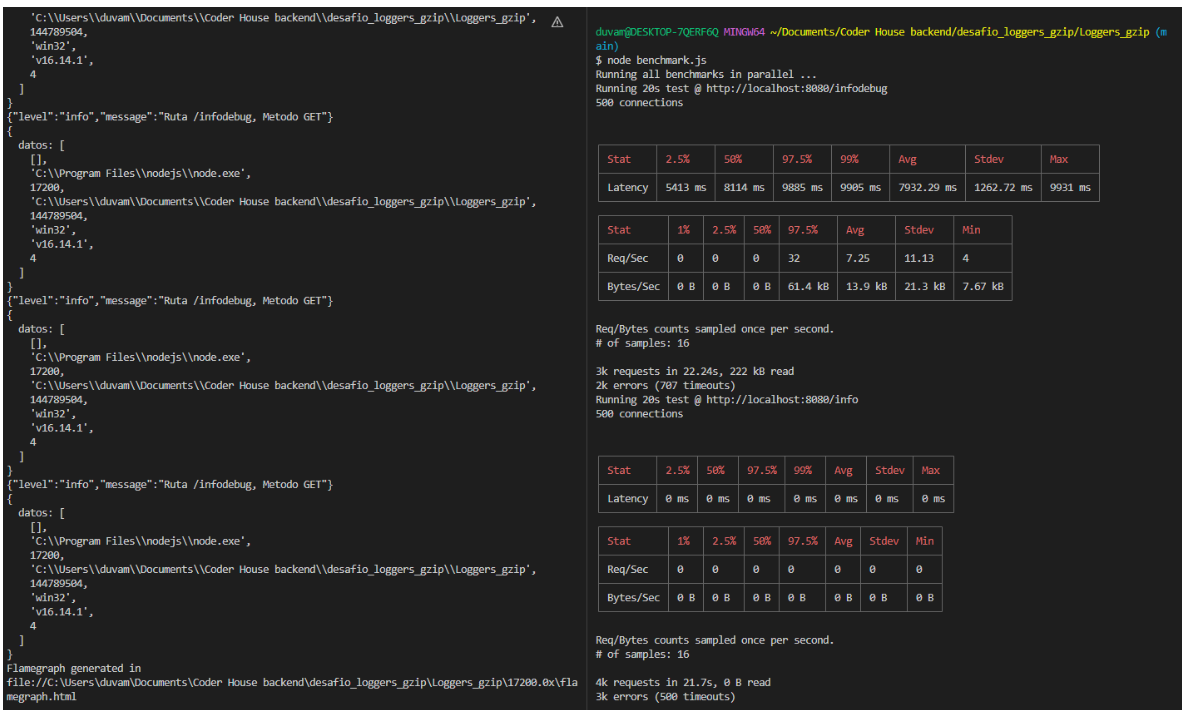
```
prof_slow.txt
196   ticks  total  nonlib   name
197
198  [Summary]:
199   ticks  total  nonlib   name
200     487   1.2%   98.6%  JavaScript
201       0   0.0%    0.0%  C++
202     209   0.5%   42.3%  GC
203   39870  98.8%          Shared libraries
204       7   0.0%          Unaccounted
```



```
prof_fast.txt
191   ticks  total  nonlib   name
192
193  [Summary]:
194   ticks  total  nonlib   name
195     477   3.2%   98.4%  JavaScript
196       0   0.0%    0.0%  C++
197     144   1.0%   29.7%  GC
198   14630  96.8%          Shared libraries
199       8   0.1%          Unaccounted
```

Autocannon

1. corremos el comando: 0x server.js y en una segunda consola corremos el comando: node benchmark.js



```
    'C:\\Users\\duvam\\Documents\\Coder House backend\\desafio_loggers_gzip\\Loggers_gzip',
    144789504,
    'win32',
    'v16.14.1',
    4
  ]
}
{"level":"info","message":"Ruta /infodebug, Metodo GET"}
{
  datos: [
    [],
    'C:\\Program Files\\nodejs\\node.exe',
    17200,
    'C:\\Users\\duvam\\Documents\\Coder House backend\\desafio_loggers_gzip\\Loggers_gzip',
    144789504,
    'win32',
    'v16.14.1',
    4
  ]
}
{"level":"info","message":"Ruta /infodebug, Metodo GET"}
{
  datos: [
    [],
    'C:\\Program Files\\nodejs\\node.exe',
    17200,
    'C:\\Users\\duvam\\Documents\\Coder House backend\\desafio_loggers_gzip\\Loggers_gzip',
    144789504,
    'win32',
    'v16.14.1',
    4
  ]
}
{"level":"info","message":"Ruta /infodebug, Metodo GET"}
{
  datos: [
    [],
    'C:\\Program Files\\nodejs\\node.exe',
    17200,
    'C:\\Users\\duvam\\Documents\\Coder House backend\\desafio_loggers_gzip\\Loggers_gzip',
    144789504,
    'win32',
    'v16.14.1',
    4
  ]
}
Flamegraph generated in
file://C:\Users\duvam\Documents\Coder House backend\desafio_loggers_gzip\Loggers_gzip\17200.0x\flamegraph.html
```

```
duvam@DESKTOP-7QERF6Q MINGW64 ~/Documents/Coder House backend/desafio_loggers_gzip/Loggers_gzip (main)
$ node benchmark.js
Running all benchmarks in parallel ...
Running 20s test @ http://localhost:8080/infodebug
500 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	5413 ms	8114 ms	9885 ms	9905 ms	7932.29 ms	1262.72 ms	9931 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	0	0	0	32	7.25	11.13	4
Bytes/Sec	0 B	0 B	0 B	61.4 kB	13.9 kB	21.3 kB	7.67 kB

```
Req/Bytes counts sampled once per second.
# of samples: 16

3k requests in 22.24s, 222 kB read
2k errors (707 timeouts)
Running 20s test @ http://localhost:8080/info
500 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms

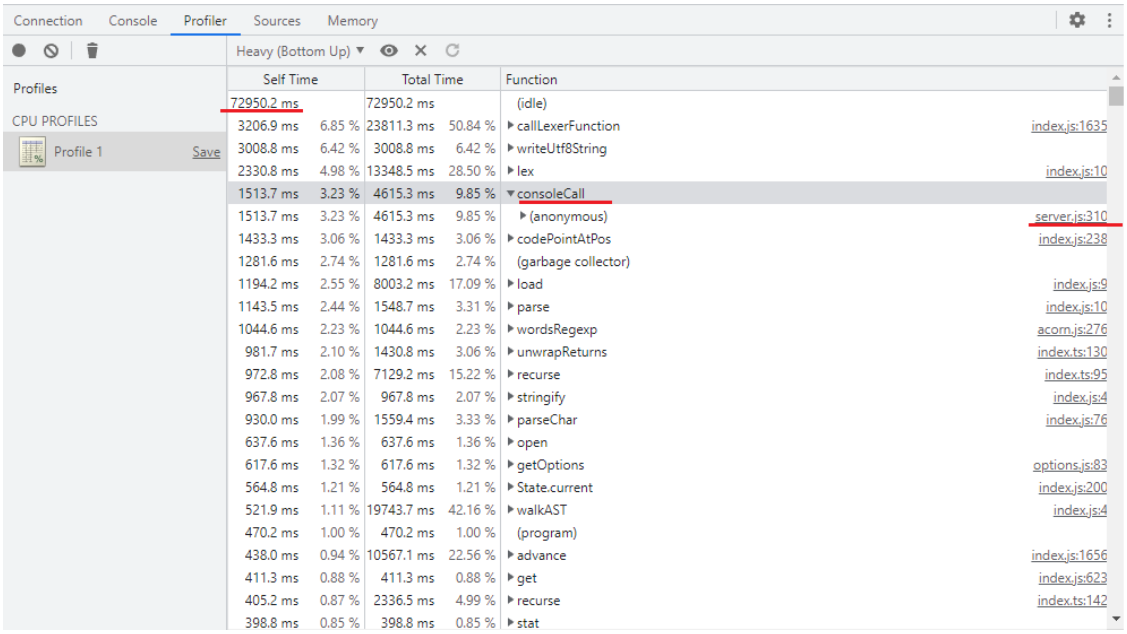
Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	0	0	0	0	0	0	0
Bytes/Sec	0 B	0 B	0 B	0 B	0 B	0 B	0 B

```
Req/Bytes counts sampled once per second.
# of samples: 16

4k requests in 21.7s, 0 B read
3k errors (500 timeouts)
```

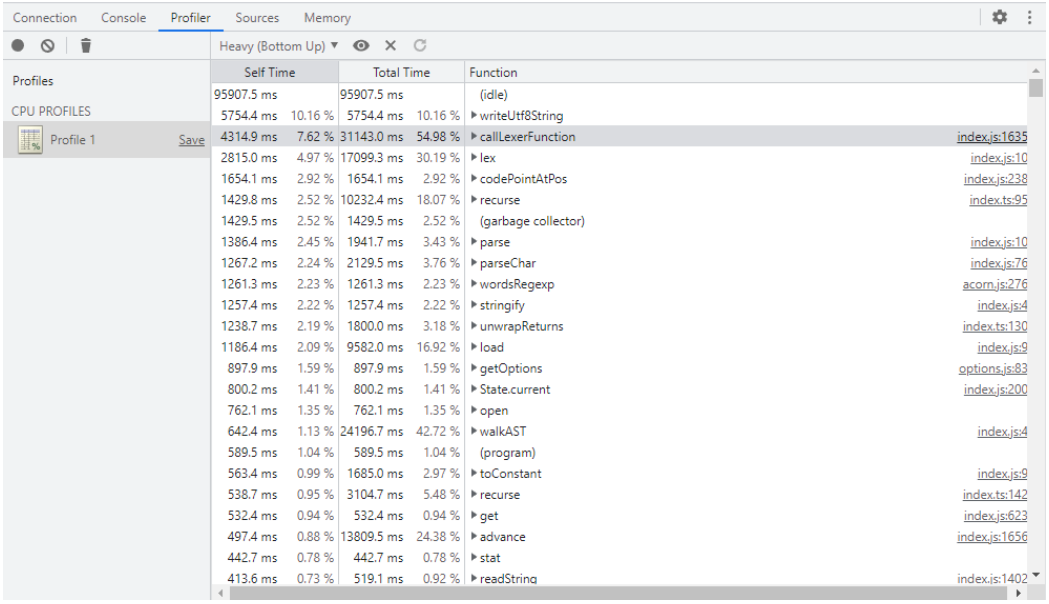
2. inspect- Google:
   corremos el comando: node --inspect server.js, entramos al navegador y colocamos chrome://inspect, luego en Open dedicated DevTools for Node, luego en profiler y run, despues en otra consola corremos el comando: artillery quick -c 50 -n 20 "http://localhost:8080/infodebug " > artillery_slow.txt, cuando termine de hacer el proceso le damos stop en profiler y arroja lo siguiente:

   modo bloqueante



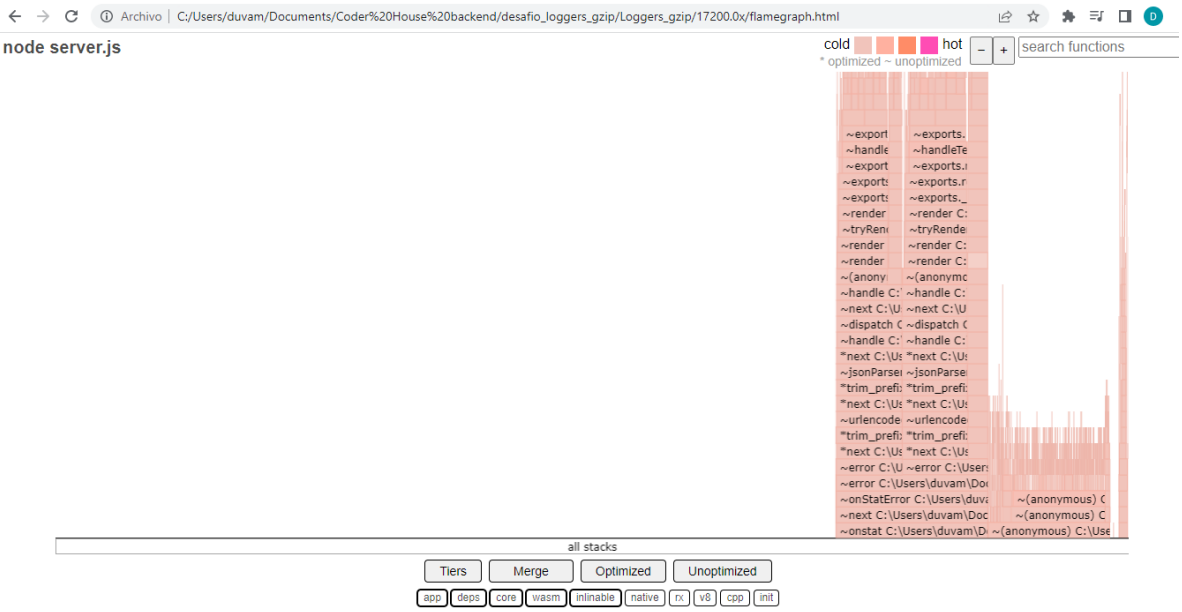
   Realizamos el mismo proceso para el otro: : artillery quick -c 50 -n 20 "http://localhost:8080/info " > artillery_fast.txt y arroja lo siguiente:

   modo no bloqueante



Como se puede observar, en el bloqueante, el proceso de consoleCall es uno de los que mas tarda en procesar y retarda la operación contraria del no bloqueante que no aparece ese proceso.

3. diagrama flama generado:



En el diagrama de Flama podemos observar que los picos los genera los procesos bloqueantes y adicional los procesos que están mas arriba tienen que correr primero que los que están abajo, los que son planos son los que no son bloqueantes, en este diagrama se ven varios picos evidenciando el proceso bloqueante que se ejecutó en el proyecto con los console.log().

Conclusión General

Durante este desafío se pudo observar diferentes procesos y como mejorar su rendimiento como con Gzip, se realizaron diferentes validaciones en las cuales se evidencia el rendimiento de cuando es un proceso bloqueante y no bloqueante, todo esto con el fin de ser consientes de los recursos que estamos gastando al correr un proyecto y así saber como sacar el mayor provecho con los recursos que se tienen.