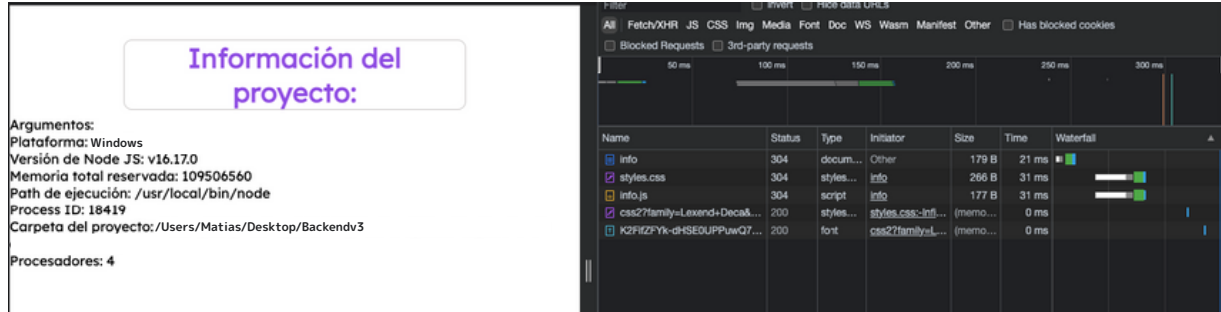


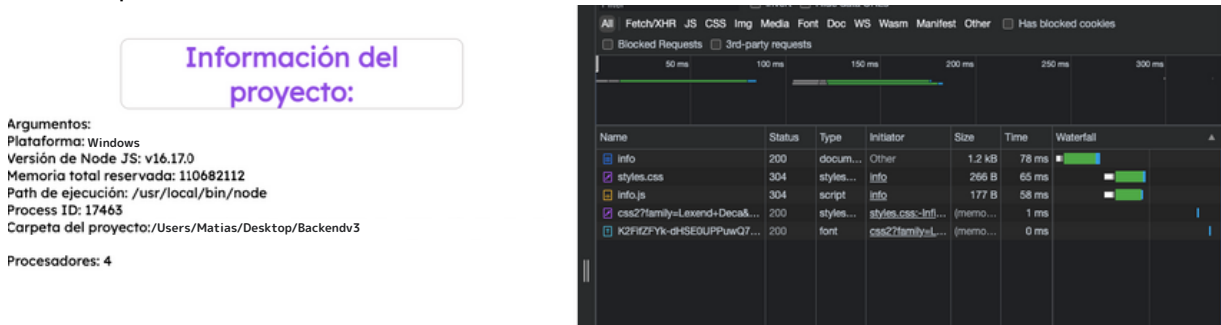
RESULTADOS DE TESTS.

1. Ruta /info con y sin compresión de Gzip:

Sin compresión 179B



Con compresión 1.2kB



Conclusión:

- Tamaño significativamente menor utilizando compress y menor tiempo de carga.

2. Artillery

Sin console.log:

```
artillery quick \  
--count 50 \  
--num 20 \  
http://localhost:8080/info \  
--output ./data/artillery.txt
```

Con console.log:

```
artillery quick \  
--count 50 \  
--num 20 \  
http://localhost:8080/info \  
--output ./data/artilleryConConsole.txt
```

```

JS server.js U  artilleryConConsole.txt U  artillery.txt U  README.md M  JS info.js U
data > artilleryConConsole.txt
15  "firstCounterAt": 1671302855252,
16  "firstHistogramAt": 1671302855350,
17  "lastCounterAt": 1671302862099,
18  "lastHistogramAt": 1671302862099,
19  "firstMetricAt": 1671302855252,
20  "lastMetricAt": 1671302862099,
21  "period": 1671302860000,
22  "summaries": {
23    "http.response_time": {
24      "min": 11,
25      "max": 702,
26      "count": 1000,
27      "p50": 247.2,
28      "median": 247.2,
29      "p75": 347.3,
30      "p90": 487.9,
31      "p95": 510.1,
32      "p99": 699.4,
33      "p999": 699.4
    }
  }

data > artillery.txt
15  "firstCounterAt": 1671302898294,
16  "firstHistogramAt": 1671302898363,
17  "lastCounterAt": 1671302902289,
18  "lastHistogramAt": 1671302902289,
19  "firstMetricAt": 1671302898294,
20  "lastMetricAt": 1671302902289,
21  "period": 1671302900000,
22  "summaries": {
23    "http.response_time": {
24      "min": 8,
25      "max": 477,
26      "count": 1000,
27      "p50": 144,
28      "median": 144,
29      "p75": 170.5,
30      "p90": 237.5,
31      "p95": 333.7,
32      "p99": 407.5,
33      "p999": 459.5
    }
  }

```

Conclusión:

- Con console.log el tiempo de respuesta y carga del servidor es mayor.

3. Autocannon

Sin console.log:

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	139 ms	195 ms	337 ms	362 ms	209.12 ms	55.61 ms	550 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	365	365	471	578	476.05	63.07	365
Bytes/Sec	419 kB	419 kB	541 kB	664 kB	547 kB	72.4 kB	419 kB

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

10k requests in 20.13s, 10.9 MB read

Con console.log:

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	198 ms	299 ms	485 ms	515 ms	317.31 ms	79.92 ms	737 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	214	214	315	361	312.86	36.64	214
Bytes/Sec	246 kB	246 kB	362 kB	414 kB	359 kB	42.1 kB	246 kB

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

6k requests in 20.12s, 7.18 MB read

Conclusión: sin console.log es más rápida la respuesta (10k requests in 20.13s vs. 6k requests in 20.12s)

4. Node inspect

C:\Users\Matias\Desktop\Backend-v3> node --inspect server.js

Debugger listening on ws://127.0.0.1:9229/3cb31aaf-4125-437e-85ae-2248908089aa

For help, see: <https://nodejs.org/en/docs/inspector>

Servidor http escuchando en el puerto 8080 en modo FORK

Debugger attached.

Sin console.log, procesos menos performantes:

Connection Console Profiler Sources Memory			
Heavy (Bottom Up) ▾ 🔍 ⌂			
Profiles	Self Time ▾	Total Time	Function
CPU PROFILES	32573.3 ms	32573.3 ms	(idle)
Profile 1 Save	630.4 ms 26.05 %	760.9 ms 31.44 %	▶ consoleCall
	98.4 ms 4.07 %	98.4 ms 4.07 %	▶ stat
	98.1 ms 4.05 %	98.1 ms 4.05 %	▶ writew
	90.8 ms 3.75 %	90.8 ms 3.75 %	(garbage collector)
	46.0 ms 1.90 %	46.0 ms 1.90 %	▶ open
	42.7 ms 1.77 %	148.4 ms 6.13 %	▶ compile
	36.8 ms 1.52 %	36.8 ms 1.52 %	(program)
	33.6 ms 1.39 %	33.6 ms 1.39 %	▶ writeUtf8String
	33.4 ms 1.38 %	13862.0 ms 572.41 %	▶ next
	29.7 ms 1.23 %	49.8 ms 2.06 %	▶ scanLine
	25.8 ms 1.06 %	451.0 ms 18.64 %	▶ send
	21.6 ms 0.89 %	71.1 ms 2.94 %	▶ (anonymous)
	19.5 ms 0.81 %	37.7 ms 1.56 %	▶ Hash
	19.4 ms 0.80 %	1819.3 ms 75.18 %	▶ initialize
	18.0 ms 0.74 %	18.0 ms 0.74 %	▶ Hash
	17.6 ms 0.73 %	32.7 ms 1.35 %	▶ writeHead
	17.1 ms 0.71 %	17.1 ms 0.71 %	▶ _addUtfOutput
	15.9 ms 0.66 %	52.7 ms 2.18 %	▶ hash
	15.7 ms 0.65 %	3079.7 ms 127.26 %	▶ handle
	15.7 ms 0.65 %	12797.8 ms 528.84 %	▶ handle
	15.6 ms 0.64 %	17.1 ms 0.71 %	▶ parse
	13.3 ms 0.55 %	24.9 ms 1.03 %	▶ nextTick
	12.6 ms 0.52 %	14.4 ms 0.59 %	▶ createWriteWrap

Con console.log, procesos menos performantes:

Connection Console Profiler Sources Memory			
Heavy (Bottom Up) ▾ 🔍 ⌂			
Profiles	Self Time ▾	Total Time	Function
CPU PROFILES	42825.7 ms	42825.7 ms	(idle)
Profile 1 Save	1583.2 ms 41.08 %	1864.5 ms 48.37 %	▶ consoleCall
	116.6 ms 3.03 %	116.6 ms 3.03 %	▶ writew
	110.1 ms 2.86 %	110.1 ms 2.86 %	▶ stat
	91.6 ms 2.38 %	91.6 ms 2.38 %	(garbage collector)
	56.4 ms 1.46 %	56.4 ms 1.46 %	(program)
	52.9 ms 1.37 %	52.9 ms 1.37 %	▶ getColorsDepth
	44.1 ms 1.14 %	140.3 ms 3.64 %	▶ compile
	41.1 ms 1.07 %	41.1 ms 1.07 %	▶ open
	36.4 ms 0.95 %	3025.6 ms 78.50 %	▶ initialize
	32.9 ms 0.85 %	32.9 ms 0.85 %	▶ writeUtf8String
	32.7 ms 0.85 %	49.0 ms 1.27 %	▶ scanLine
	27.3 ms 0.71 %	49.7 ms 1.29 %	▶ nextTick
	26.9 ms 0.70 %	25506.5 ms 661.76 %	▶ next
	25.2 ms 0.65 %	522.3 ms 13.55 %	▶ send
	23.5 ms 0.61 %	69.9 ms 1.81 %	▶ hash
	23.0 ms 0.60 %	23.0 ms 0.60 %	▶ Hash
	22.8 ms 0.59 %	46.3 ms 1.20 %	▶ Hash
	22.1 ms 0.57 %	166.5 ms 4.32 %	▶ value
	21.1 ms 0.55 %	38.5 ms 1.00 %	▶ writeHead
	20.0 ms 0.52 %	3152.1 ms 81.78 %	▶ session
	19.6 ms 0.51 %	261.1 ms 7.29 %	▶ log
	19.4 ms 0.50 %	53.4 ms 1.39 %	▶ formatRaw
	17.0 ms 0.44 %	66.9 ms 1.74 %	▶ resOnFinish
	16.8 ms 0.44 %	23340.6 ms 605.56 %	▶ handle

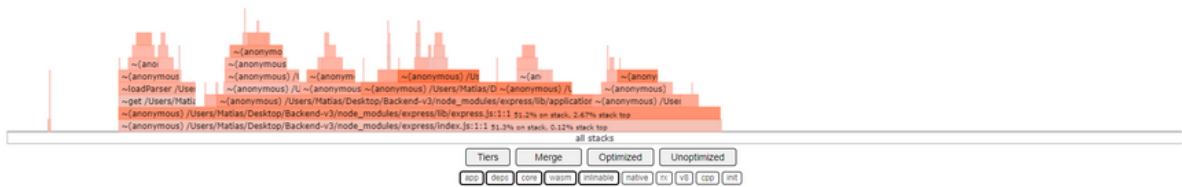
5. Diagrama de Flama 0x en la ruta routes/info:

Sin console.log

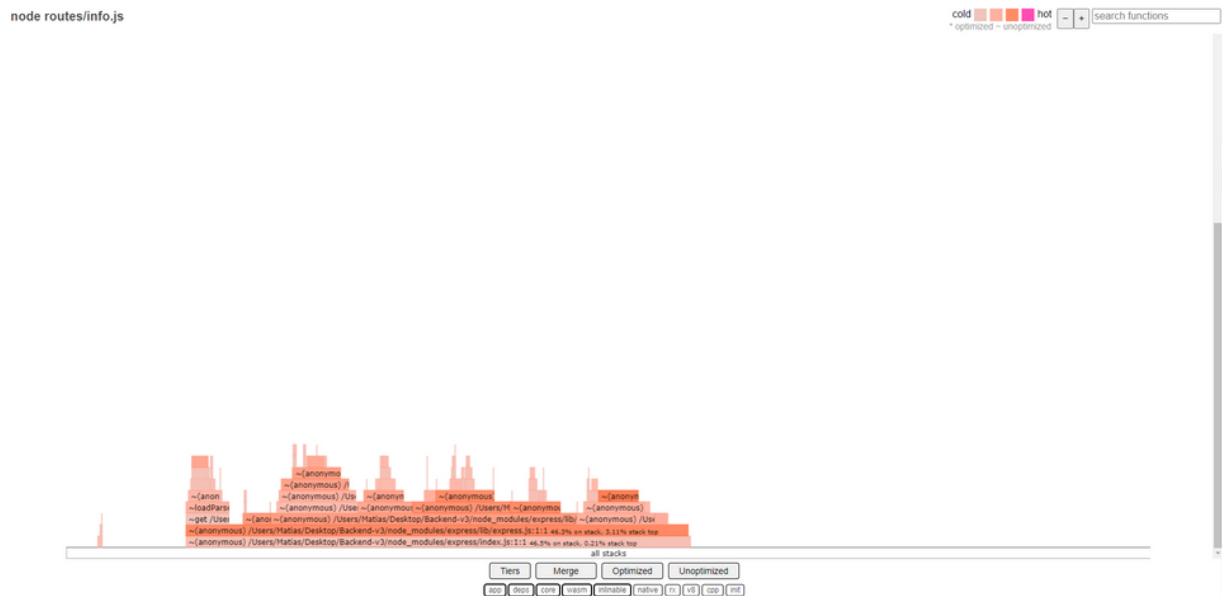
Carpeta 16037.0x dentro de data

node routes/info.js

cold hot
* optimized - unoptimized search functions



Con console.log



Conclusión general:

- Si bien no es mucha la diferencia porque la información no es muy extensa, en todos los casos la respuesta fue más rápida sin el console.log y el esfuerzo del servidor fue menor.