

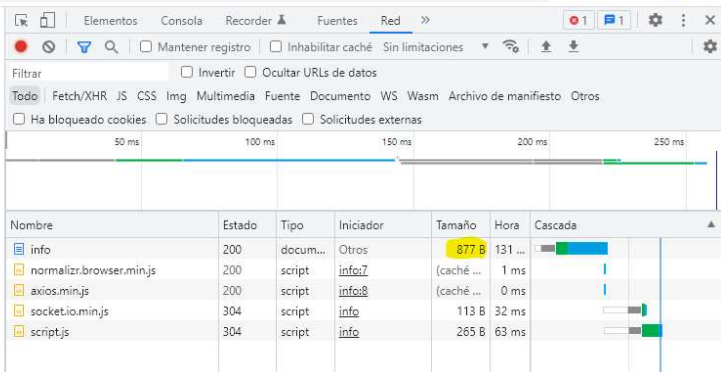
Análisis de pruebas.

- Comparación con y sin compresión ruta /info:

Con compresión:

Información

- Carpeta del proyecto: C:\Users\jfigu\OneDrive\Escritorio\Desafios_BackEnd
- Id del proceso: 7228
- Versión de Node.js: v18.2.0
- Título del proceso: C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell.exe
- Sistema operativo: win32
- Directorio de ejecución: C:\Program Files\nodejs\node.exe
- Memoria total reservada (rss): 93437952
- Número de procesadores: 2



Nombre	Estado	Tipo	Iniciador	Tamaño	Hora	Cascada
info	200	document	Otros	877 B	131 ms	
normalizr.brower.min.js	200	script	info7	(caché ...)	1 ms	
axios.min.js	200	script	info8	(caché ...)	0 ms	
socket.io.min.js	304	script	info	113 B	32 ms	
script.js	304	script	info	265 B	63 ms	

Encabezados

General

Solicitar URL: `http://localhost:8080/api/info`

Método de la solicitud: `GET`

Código de estado: 200 OK

Dirección remota: `[::1]:8080`

Política de referencia: `strict-origin-when-cross-origin`

Encabezados de respuesta

Ver origen

Connection: `keep-alive`

Content-Encoding: `gzip`

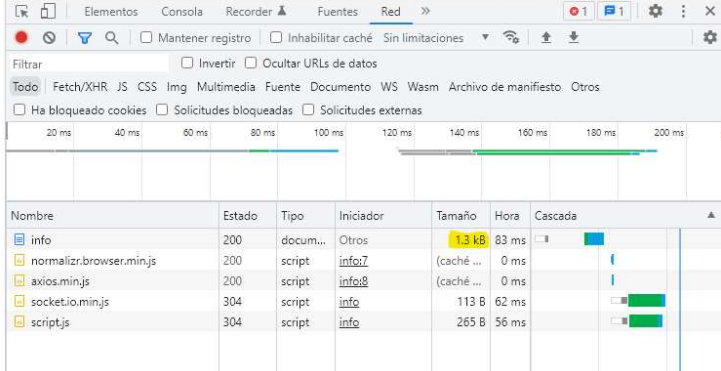
Content-Type: `text/html; charset=utf-8`

Date: `Wed, 14 Dec 2022 23:54:26 GMT`

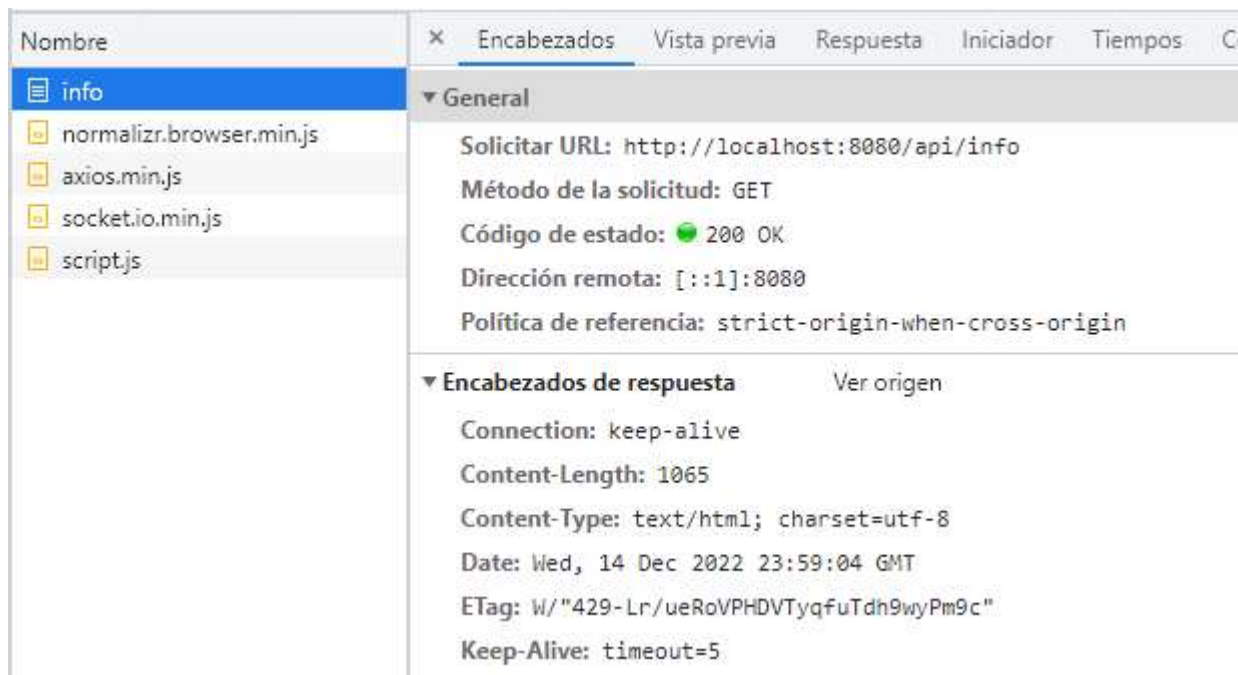
Sin compresión:

Información

- Carpeta del proyecto: C:\Users\jfigu\OneDrive\Escritorio\Desafios_BackEnd
- Id del proceso: 2596
- Versión de Node.js: v18.2.0
- Título del proceso: C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell.exe
- Sistema operativo: win32
- Directorio de ejecución: C:\Program Files\nodejs\node.exe
- Memoria total reservada (rss): 93499392
- Número de procesadores: 2



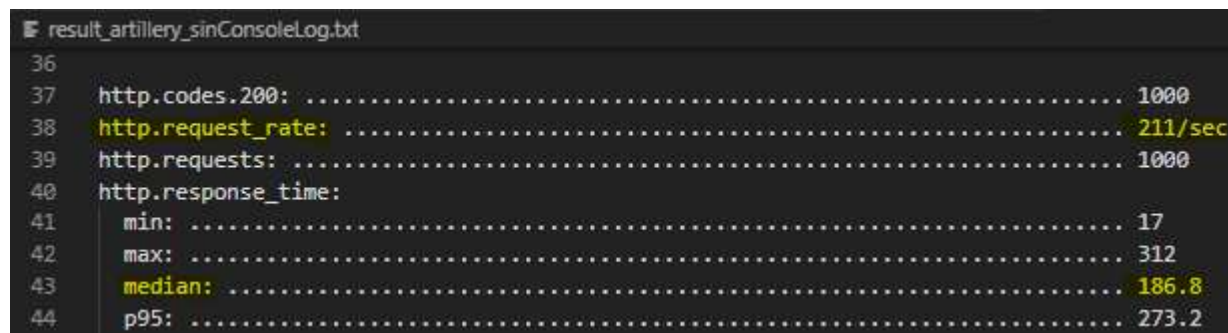
Nombre	Estado	Tipo	Iniciador	Tamaño	Hora	Cascada
info	200	document	Otros	1.3 kB	83 ms	
normalizr.brower.min.js	200	script	info7	(caché ...)	0 ms	
axios.min.js	200	script	info8	(caché ...)	0 ms	
socket.io.min.js	304	script	info	113 B	62 ms	
script.js	304	script	info	265 B	56 ms	



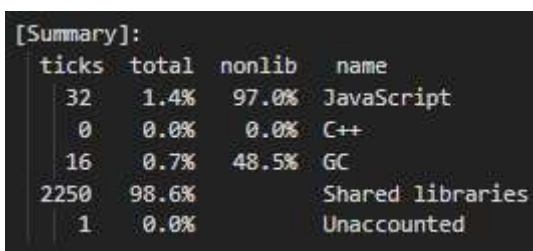
- Comparación con y sin “console.log” en ruta /info:

node --prof index.js

artillery quick --count 50 -n 20 http://localhost:8080/api/info > result_artillery_sinConsoleLog.txt



node --prof-process sinConsoleLog.log > result_sinConsoleLog.txt



artillery quick --count 50 -n 20 http://localhost:8080/api/info > result_artillery_conConsoleLog.txt

```

result_artillery_conConsoleLog.txt
53 http.codes.200: ..... 1000
54 http.request_rate: ..... 115/sec
55 http.requests: ..... 1000
56 http.response_time:
57   min: ..... 4
58   max: ..... 650
59   median: ..... 415.8
60   p95: ..... 550.1
61   p99: ..... 584.3

```

node --prof-process conConsoleLog.log > result_conConsoleLog.txt

```

[Summary]:
  ticks total  nonlib   name
    49    1.6%    98.0%  JavaScript
     0    0.0%     0.0%    C++
    37    1.2%    74.0%    GC
  3109   98.4%           Shared libraries
     1    0.0%           Unaccounted

```

- Prueba con Autocannon, Node inspect y 0x:

Autocannon

```

PS C:\Users\jfigu\OneDrive\Escritorio\Desafios_BackEnd> node benchmark.js
Running all benchmarks in parallel...
Running 20s test @ http://localhost:8080/api/info
100 connections

```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	297 ms	366 ms	695 ms	799 ms	387.48 ms	89.67 ms	843 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	102	102	269	321	256	45.81	102
Bytes/Sec	135 kB	135 kB	356 kB	424 kB	338 kB	60.5 kB	135 kB


```

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

5k requests in 20.2s, 6.76 MB read
PS C:\Users\jfigu\OneDrive\Escritorio\Desafios_BackEnd>

```

Inspect

Tiempo individual ▾	Tiempo total	Función
171.5 ms 1.07 %	557.0 ms 3.48 %	▶ setupHelperArgs javascript-compiler.js:1210
163.9 ms 1.02 %	163.9 ms 1.02 %	▶ open index.js:129
162.5 ms 1.02 %	4561.3 ms 28.48 %	▶ logRuta index.js:596
158.0 ms 0.99 %	158.0 ms 0.99 %	▶ Hash index.js:596
145.0 ms 0.91 %	352.9 ms 2.20 %	▶ hash index.js:596
118.8 ms 0.74 %	2944.4 ms 18.39 %	▶ compile javascript-compiler.js:64
115.9 ms 0.72 %	115.9 ms 0.72 %	▶ extend utils.js:18
113.1 ms 0.71 %	132.9 ms 0.83 %	▶ resolve node:path:158
106.4 ms 0.66 %	435.3 ms 2.72 %	▶ replaceStack javascript-compiler.js:994
105.2 ms 0.66 %	105.2 ms 0.66 %	▶ registerDestroyHook
99.5 ms 0.62 %	27963.9 ms 174.63 %	▶ next index.js:177
95.9 ms 0.60 %	652.7 ms 4.08 %	▶ castChunk code-gen.js:49
95.4 ms 0.60 %	5483.5 ms 34.24 %	▶ session index.js:179
94.7 ms 0.59 %	94.7 ms 0.59 %	▶ setWriteHeadHeaders index.js:105
92.1 ms 0.58 %	94.7 ms 0.59 %	▶ parse index.js:106
88.1 ms 0.55 %	88.1 ms 0.55 %	▶ module.exports index.js:16

esafios_BackEnd_index.js	596	3.6 ms	function hash(sess) {
	597		// serialize
	598	134.5 ms	var str = JSON.stringify(sess, function (key, val) {
	599		// ignore sess.cookie property
	600	0.7 ms	if (this === sess && key === 'cookie') {
	601		return
	602		}
	603		
	604	0.6 ms	return val
	605		})
	606		
	607		// hash
	608		return crypto
	609	1.2 ms	.createHash('sha1')
	610	3.9 ms	.update(str, 'utf8')
	611	0.6 ms	.digest('hex')
	612		}

orio_Desafios_BackEnd_index.js	126		////////////////////////////////
	127		////////////////////////////////
	128		let today = new Date()
	129		const logRuta = (req, res, next) => {
	130	151.1 ms	let now = today.toLocaleString()
	131	10.0 ms	logger.info(`[\${now}] Ruta \${req.url} método \${req.method}`)
	132	1.0 ms	next()
	133		}
	134		app.use(logRuta)
	135		////////////////////////////////
	136		////////////////////////////////
	137		app.get('/', (req, res) => {
	138		res.send(`<h1>Servidor express en \${PORT} - PID \${process.pid
	139		})
	140		app.use('/api', router)
	141		
	142		app.get('/datos', (req, res) => {
	143		console.log(`Here from process \${process.pid} listening in port

node index.js



Conclusión.

Al realizar las pruebas se evidencia la importancia de generar un código lo más óptimo posible, ya que una sentencia u otra claramente hace la diferencia en el rendimiento y ejecución de nuestro programa.