Se debia realizar el analisis de performance sobre la ruta /info del proyecto en modo fork. Agregado o extrayendo un console.log antes de devolver la informacion al cliente.

prof-process

1- Utilizando el test con --prof de node js y aplicando Artillery con el siguiente comando: artillery quick --count 20 -n 50 "http://localhost:8080/info". Se debian realizar 50 conexiones concurrentes con 20 request cada una.

Statistical profiling result from v8-conConsole.log, (6482 ticks, 2 unaccounted, 0 excluded).

Statistical profiling result from v8-sinConsole.log, (5837 ticks, 1 unaccounted, 0 excluded).

[Shared libraries]: ticks total nonlib name

Resultado con ConsoleLog

```
79.1%
                        C:\WINDOWS\SYSTEM32\ntdll.dll
                                                          [Summary]:
                        C:\Program Files\nodejs\node.exe
  1264
        19.5%
                                                            ticks total nonlib
                       C:\WINDOWS\System32\KERNELBASE.o
         0.1%
                                                                           97.5% JavaScript
         0.0%
                       C:\WINDOWS\System32\WS2_32.dll
                                                                            0.0% C++
                        C:\WINDOWS\System32\KERNEL32.DLI
                                                                          43.2% GC
                                                                                  Shared libraries
                                                            6401
                                                                  98.8%
[JavaScript]:
                                                                   0.0%
                                                                                  Unaccounted
  ticks total nonlib name
                12.3% LazyCompile: *next C:\Users\Ale
          0.2%
                                                          [C++ entry points]:
                 9.9% LazyCompile: *parse C:\Users\Ale
     8
         0.1%
                                                            ticks
                                                                    cpp total name
                 8.6% LazyCompile: *resolve path.js:1
         0.1%
          0.1%
                 7.4% LazyCompile: *wrap C:\Users\Ale
                                                          [Bottom up (heavy) profile]:
                 4.9% RegExp: [ \t]+$
         0.1%
                                                           Note: percentage shows a share of a particular caller in the total
                  3.7% LazyCompile: *SourceNode_walk C
     3
          0.0%
                                                           amount of its parent calls.
                  2.5% RegExp: ^(?:\{\{(~)?&)
          0.0%
                                                           Callers occupying less than 1.0% are not shown.
                 2.5% LazyCompile: *hidden internal/en
          0.0%
                 2.5% LazyCompile: *anonymous C:\User:
          0.0%
                                                            ticks parent name
                 2.5% LazyCompile: *anonymous C:\Users
         0.0%
                                                            5130 79.1% C:\WINDOWS\SYSTEM32\ntdll.dll
     2
                 2.5% LazyCompile: *SourceNode_add C:\
         0.0%
                  2 5% | JazyComnila: +canonymous C. \ | | | | | Alajandra \ | Dackton \ Codonhouse \ hack
```

ticks total nonlib name 4848 83.1% C:\WINDOWS\SYSTEM32\ntdll. 931 15.9%

[Shared libraries]:

inferior.

Resultado sin ConsoleLog

```
ticks total nonlib
                     C:\Program Files\nodejs\nod
                                                       0.9% 98.2% JavaScript
                                                 54
        0.0%
                     C:\WINDOWS\System32\KERNEL3
                                                       0.0%
                                                              0.0% C++
        0.0%
                     C:\WINDOWS\System32\KERNELE
                                                       0.6%
                                                 34
                                                             61.8% GC
                                                      99.1%
                                                                    Shared libraries
                                                5782
[JavaScript]:
                                                       0.0%
                                                                   Unaccounted
 ticks total nonlib
             10.9% LazyCompile: *resolve path.
                                              [C++ entry points]:
               9.1% LazyCompile: *parse C:\Use
        0.1%
                                                        cpp total
                                                ticks
               7.3% LazyCompile: *wrap C:\User:
        0.1%
               7.3% LazyCompile: *next C:\User:
        0.1%
                                               [Bottom up (heavy) profile]:
               5.5% LazyCompile: *castChunk C:
        0.1%
                                               Note: percentage shows a share of a particular caller in the total
               3.6% RegExp: [ \t]+$
        0.0%
                                               amount of its parent calls.
               3.6% LazyCompile: *pushSource C
        0.0%
                                               Callers occupying less than 1.0% are not shown.
               3.6% LazyCompile: *next C:\User:
        0.0%
               3.6% LazyCompile: *anonymous C:
        0.0%
                                                ticks parent name
               3.6% LazyCompile: *Module._load
        0.0%
                                                     83.1% C:\WINDOWS\SYSTEM32\ntdll.dll
               1.8% RegExp: ^\. |this\b
        0.0%
               1.8% RegExp: ^(?:\}\}\}\})
                                                     15.9% C:\Program Files\nodejs\node.exe
 Desde este profiling se observa una mayor cantidad de Ticks de procesador en la
 version de console.log, mientras que en la version sin console.log una cantidad
```

[Summary]:

Artillery

Report @ 11:38:15(-0300) 2021-08-22 Elapsed time: 10 seconds

Requests completed: 205 Mean response/sec: 21.95 Response time (msec):

Report @ 11:36:27(-0300) 2021-08-22

Elapsed time: 10 seconds

segundos.

Scenarios launched: 20

Scenarios completed: 0

Resultado sin ConsoleLog

Started phase 0, duration: 1s @ 11:36:17(-0300) 2021-08-22

min: 7 max: 983 median: 382 p95: 860.3 p99: 981.5 Codes: 200: 205 Report @ 11:36:37(-0300) 2021-08-22 Elapsed time: 20 seconds Scenarios launched: 0 Scenarios completed: 0 Requests completed: 223 Mean response/sec: 21.85 Response time (msec): version, en este se observa una media en la version sin ConsoleLog de 382

```
Response time (msec):
                                                                 min: 18
                                                                 max: 1072
                                                                 median: 449
                                                                 p95: 835.5
                                                                 p99: 1016.6
                                                                Codes:
                                                                  200: 195
                                                              Report @ 11:38:25(-0300) 2021-08-22
                                                              Elapsed time: 20 seconds
                                                                Scenarios launched: 0
                                                                Scenarios completed: 0
                                                                Requests completed: 202
                                                                Mean response/sec: 20.93
                                                                Response time (msec):
En el profiling resultante de Artillery se puede medir el tiempo de respuesta en cada
segundos mientras que en la otra el tiempo de respuesta promedia los 449
```

Stdev

10

15.7 kB

heapTotal:(process.memoryUsage().heapTotal / 1024 / 1024).toFixed(2)+'MB',

heapUsed:(process.memoryUsage().heapUsed / 1024 / 1024).toFixed(2)+'MB',

s:269

s:172

s:118

s:216

s:892

source-noc

source-nod

code-ge

code-ge

compil

~(anonymous) C: ~(anonymous) C:\ ~(anonymous) C:

~(anonymous) C:\Users\Alej ~(anonymous) C:\Users\Aleja

~executeUserEntryPoint inter

~(anonymous) C: ~(anonymous) C:

~par

~_getDir C:\Use ∘getTemplates

~(anonymous)

~getPartials C:

~renderView C

~render C:\Use

~(anonymous) C:\Users\Ale ~(anonymous) C:\Users\Ale

~executeUserEntryPoint inte

~tryStat C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\exp ~getPartials ~join path.j

~resolve C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\exp ~renderView C:\Users\A ~lookup C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\expr ~render C:\Users\Alejar

~View C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\expre: ~tryRender C:\Users\Ale

~render C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\application.js:531

~render C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\response.js:989:

Max

Resultado con ConsoleLog

Scenarios launched: 20

Requests completed: 195

Scenarios completed: 0

Mean response/sec: 20

Started phase 0, duration: 1s @ 11:38:05(-0300) 2021-08-22

Autocannon 2- Utilizando Autocannon en linea de comando con 100 conexiones y en un tiempo de 20 segundos. autocannon -c 100 -d 20 "http://localhost:8080/info" Resultado con Console.log

97.5% 99% 50%

52

Running 20s test @ http://localhost:8080/info?console=true

44

Avg 1954 ms 3562 ms 3753 ms 2387.99 ms 476.9 ms 4134 ms Latency 2130 ms 2.5% 50% 97.5% Stat 1% Stdev Min Avg

34.65

18.42

0

Θ

2.5%

100 connections

Stat

Req/Sec

/0.4	0 B	0 5	ттэ кв		90.7 K	B 46.2	*****	26.2 KB	
eq/Bytes c 93 request									
esultad	o SIN	Cons	ole.lo	g					
lunning 20: .00 connect		http:/	/localho	st:8080/i	nfo				
		http:/			nfo 99%	Avg		Stdev	Max
.00 connect	ions	50%	9	7.5%		Avg 2032.38	ms	Stdev 364.45 ms	11/3/61
00 connect	2.5%	50%	9	7.5%	99%	1004	ms	364.45 ms	Max 3794 ms

152 kB

123 kB

40 kB

Bytes/Sec 0 B 0 B 136 kB

Req/Bytes counts sampled once per second.	
En el resultado por consola de Autocannon se observa la latencia, es decir el tiempo de carga o respuesta. En esta se ve en los test realizado con consoleLe un promedio de 2387 ms, mientras en el otro de 2032.38 ms. En el segundo cuadro se observa la cantidad de peticiones atendidas en un tiempo determinado, en está el test realizado sobre la ruta info con console.lo promedia 34,65, el otro promedia 46.85. En el ultimo item se muestra la cantidad de bytes devueltos, en el test de console.log se observa 90.7kb de promedio mientras que en el otro 123 kb, un cantidad superior.	og g

Profile 1 2064.3 ms 4.06 % ▼ handleWriteReq internal/stream... commons.js:47 2064.3 ms 4.06 % 2064.3 ms 4.06 % Profile 2 2064.3 ms 4.06 % 2064.3 i 882.2 ms 1.74 % 3012.7 r router.get('/info',(req,res)=>{ 882.2 ms 1.74 % 3012.7 i 2.9 ms const info = { 882.2 ms 1.74 % 3012.7 i 0.4 ms argvs : process.argv, 882.2 ms 1.74 % 3012.7 0.2 ms so: process.platform, 269.6 ms 0.53 % 410.6 0.7 ms nodeVersion: process.version, **0.1** ms memoryUse: { 251.1 ms 0.49 % 1188.2 1 rss:(process.memoryUsage().rss / 1024 / 1024).toFixed(2)+'MB', 6.3 ms 236.7 ms 0.47 % 838.4

2- Utilizando Autocannon en linea de comando con 100 conexiones y en un

autocannon -c 100 -d 20 "http://localhost:8080/info"

Total Time

2064.3 ms 4.06 % 2064.3 ms 4.06 % ▼ writeUtf8String

39399.8 ms 77.58 % 39399.8 ms 77.58 % (program)

235.81

1881.5

Function

3.8 ms

1.6 ms

12.6 ms

1.2 ms

Resultado con ConsoleLog

Profiles

CPU PROFILES

Self Time

235.8 ms 0.46 %

97.1 ms 0.19 %

12.6 ms.

index.js

parametros anteriores:

~compileInput

Vista Ampliada

~ret C:\Users\/ ~senc

~_renderTempl ~done ~render C:\Use ~rende

~formatValue internal/ul

~inspect internal/util/ins

~formatWithOptionsInte

~(anonymous) C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\src\routes\index.routes.js:16:20 1% on stack

handle C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\router\layer.js:86:49 1% on stack ~next C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\router\route.js:114:16 1% on stack ~dispatch C;\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\router\route.js:98:45 1% on stack *handle C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\router\layer.js:86:49 1% on stack *next C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\router\index.js:176:16 1% on stack ~handle C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\router\index.js:136:31 1% on stack

~once even ~formatWithOptions inte

~value internal/console/ ~value internal/console/

~log internal/console/constructor.js:347:6 0.2% or

89.4 ms 0.18 % 451.7 r

tiempo de 20 segundos con el siguiente comando con node en modo **inspect** :

212.1 ms 0.42 % 290.21 external:(process.memoryUsage().external / 1024 / 1024).toFixed(2)+'MB', 1.8 ms 157.3 ms 0.31 % 157.3 : arrayBuffers:(process.memoryUsage().arrayBuffers / 1024 / 1024).toFixed(2)+'MB' , 2.2 ms 145.4 ms 0.29 % 145.4 135.4 ms 0.27 % 131.7 ms 0.26 % 691.1 1 path: process.execPath, 120.0 ms 0.24 % 481.4 0.4 ms processId: process.pid, folder: process.cwd(), 118.8 ms 0.23 % 118.8 ı 2.1 ms numCPUs: numCPUs

if(req.query.console){

router.get('/randoms',(req,res)=>{

computo.on('message',numbers => {

return res.render('info',{ layout:'index', info })

/* const computo = fork('./src/child_processes/generateRandoms.js')

const cantidad = req.query.cant | 100000000

console.log(info)

computo.send(cantidad)

```
return res.json({ numbers })
Resultado sin ConsoleLog
                          Self Time ▼
                                          Total Time
                                                      Function
                      38348.4 ms 83.09 % 38348.4 ms 83.09 % (program)
 DFILES
                       390.5 ms 0.85 % 416.8 ms 0.90 % ▶ next
                       300.7 ms 0.65 %
                                       300.7 ms 0.65 % ▶ writeUtf8String
Profile 1
                       226.1 ms 0.49 %
                                      1049.6 ms 2.27 % ▶ SourceNode_walk
                       204.8 ms 0.44 %
                                       204.8 ms 0.44 % (garbage collector)
                                       271.4 ms 0.59 % ► SourceNode_add
                       198.5 ms 0.43 %
                                       775.6 ms 1.68 % ▶ parse
                       195.2 ms 0.42 %
                                       146.4 ms 0.32 % ▶ stat
                       146.4 ms 0.32 %
                                       658.3 ms 1.43 % ▶ wrap
                       136.4 ms 0.30 %
                       132.1 ms 0.29 %
                                        132.1 ms 0.29 % ▶ quotedString
                       130.8 ms 0.28 %
                                        130.8 ms 0.28 % ▶ writeBuffer
                                        453.3 ms 0.98 % ▶ createFunctionContext
                       125.9 ms 0.27 %
                                                                                                                                                                javascript-compili
                       116.0 ms 0.25 %
                                        116.0 ms 0.25 % ▶ writev
                        96.4 ms 0.21 % 1787.4 ms 3.87 % ▶ compile
                                                                                                                                                                 <u>javascript-compi</u>
                        95.4 ms 0.21 %
                                       451.3 ms 0.98 % ▶ replaceStack
                                                                                                                                                                javascript-compil
                        74.9 ms 0.16 %
                                      3248.8 ms 7.04 % render
                                                                                                                                                               express-handlebar
                                        65.1 ms 0.14 % ▶ nextTick
                        56.6 ms 0.12 %
                                                                                                                                                           internal/proces... queue
                        55.5 ms 0.12 %
                                       229.9 ms 0.50 % ▶ accept
                        49.5 ms 0.11 %
                                        113.7 ms 0.25 % ▶ anonymous
                        47.6 ms 0.10 %
                                       483.4 ms 1.05 % ▶ accept
                        47.6 ms 0.10 % 4434.5 ms 9.61 % ▶ next
                        46.7 ms 0.10 % 162.9 ms 0.35 % ▶ deserializeObject
       Utilizando la funcion inspect de Chrome se pueden ver las diferencias
       observando el teimpo de carga en ms, en el cual en el proceso (program) se
```

Por otra parte, si vemos el archivo index.routes, que es donde se realiza el

3- Utilizando diagrama de Flama con 0x y autocannon con los mismos

0x -P 'autocannon -c 100 -d 20 "http://localhost:8080/info" src/index.js

0x -P 'autocannon -c 100 -d 20 "http://localhost:8080/info?console=true" src/

console.log se puede observar en esa linea un tiempo mayor a las restantes de

observa un mayor tiempo de carga en el de console.log.

Resultado con ConsoleLog node src/index.js + search function optimized ~ unoptimized

all stacks

~router C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\router\index.js:46:18 1% on stack handle C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\router\layer.js:86:49 1% on stack "next C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\router\index.js:176:16 1,2% on stack *emit events is:263:44 + 204 an Resultado sin ConsoleLog search functions node src/index.js cold hot * optimized ~ unoptimized ~(anonymous) C:

all stacks

~memoryUsage internal/process ~render C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\response.js:989:29 0.6% on stack, 0.06% stack top

~tryStat C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\view.js:1

~resolve C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\view.js:1

~lookup C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\view.js:1

~View C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\view.js:52: ~module.export ~tryRender C:\l

~render C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\application.js:531:29 0.6% on stack

~StatSync fs.js:1081:18 0,4% on stack, 0,43% stack top

~handle C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\router\index.js:136:31 0.9% on stack, 0.06% stack top

*process_params C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\router\index.js:327:47 1% on stack

~router C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\router\index.js:46:18 1% on stack

next C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\router\index.js:176:16 1% on stack

verror C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\serve-static\index.js:115:39 1% on stack

(anonymous) C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\src\routes\index.routes.is:16:20 0.7% on stack ~handle C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\router\layer.js:86:49 0.7% on stack ~next C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\router\route.js:114:16 0.7% on stack ~dispatch C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\router\route.js:98:45 0.8% on stack, 0.06% stack top ~handle C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\router\layer.js:86:49 0,8% on stack 'process_params C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\router\index.js:327:47 0.8% on stack next C:\Users\Alejandro\Desktop\Coderhouse\backend\desafios\desafio-32\node_modules\express\lib\router\index.js:176:16 0.8% on stack

~compileIn ~ret C:\Use

~_renderTe ~render C:\

Vista Ampliada

puede apreciar una diferencia en el ancho de los picos de la parte derecha, propia del proceso bloqueante que produce el consoleLog, haciendo que este ocupe mas tiempo. En la vista ampliada se observa el porcentaje en la pila, en el cual, el archivo index.routes, en el que se encuentra la funcion de console.log tiene un porcentaje mas alto dentro de esta.

En el diagrama resultante de ambos profiling utilizando la herramienta de Ox se

A partir de estos test se pudo observar como afecta tanto en tiempo de procesador, como en tiempo de respuesta el incluir un proceso bloqueante como lo es el

consoleLog, en todos los datos obtenidos se puede ver una diferencia.