Desafío 13 - Logs, debug, profiling

Incorporar al proyecto de servidor de trabajo la compresión gzip.
 Verificar sobre la ruta /info con y sin compresión, la diferencia de cantidad de bytes devueltos en un caso y otro.

Ruta /info sin compresión:

```
0 B / 1.9 kB transferred | 0 B / 1.1 kB resources | Finish: 314 ms | DOMContentLoaded: 332 ms | Load: 330 ms | What's New ×
```

Ruta /info con compresión:

```
0 B / 1.1 kB transferred | 0 B / 1.1 kB resources | Finish: 48 ms | DOMContentLoaded: 54 ms | Load: 53 ms

What's New ×
```

Análisis: Efectivamente cuando se aplica compresión se verifica una gran diferencia en la cantidad de bytes y en los milisegundos.

 Vamos a trabajar sobre la ruta '/info', en modo fork, agregando ó extrayendo un console.log de la información colectada antes de devolverla al cliente. Además desactivaremos el child process de la ruta '/randoms'

Para ambas condiciones (con o sin console.log) en la ruta '/info' OBTENER:

1) El perfilamiento del servidor, realizando el test con --prof de node.js. Analizar los resultados obtenidos luego de procesarlos con --prof-process.

Se realizó Artillery emulando 50 conexiones simultaneas con 20 request por cada una y se verifica el siguiente resultado de los logs:

```
[Summary]:
ticks total nonlib name
11 0.4% 100.0% JavaScript
0 0.0% 0.0% C++
7 0.2% 63.6% GC
2849 99.6% Shared libraries
```

Este resultado es CON CONSOLE.LOG

Cuando index.js posee console.log() hay muchos mas ticks que cuando no los posee. La aplicación CON console.log() tardó 2849 ciclos de reloj en ejecutarse.

```
[Summary]:
ticks total nonlib name
5 0.2% 100.0% JavaScript
0 0.0% 0.0% C++
6 0.3% 120.0% GC
2193 99.8% Shared libraries
```

Este resultado es SIN CONSOLE.LOG

Un tic es como un ciclo de reloj utilizado por un proceso de nodo. Entonces, en teoría, la aplicación SIN console.log() tardó 2193 ciclos de reloj en ejecutarse.

La diferencia de ticks es de 650, no es mucha diferencia pero si apuntamos a una mejor performance hay que tener en cuenta que los console.log() afectan a la misma.

<u>Utilizaremos como test de carga Artillery en línea de comandos, emulando 50 conexiones</u> <u>concurrentes con 20 request por cada una. Extraer un reporte con los resultados en archivo de texto.</u>

Con console.log

Sin console.log

Phase started: unnamed (index: 0, duration: 1s) 01:26:07(-0300) Phase completed: unnamed (index: 0, duration: 1s) 01:26:08(-0300)	Phase started: unnamed (index: 0, duration: 1s) 01:36:12(-0300) Phase completed: unnamed (index: 0, duration: 1s) 01:36:13(-0300)
All VUs finished. Total time: 10 seconds	All VUs finished. Total time: 5 seconds
Summary report @ 01:26:15(-0300)	Summary report @ 01:36:15(-0300)
http.codes.200: 1000	http.codes.200: 1000
http.request_rate: 142/sec	http.request_rate: 365/sec
http.requests: 1000	http.requests: 1000
http.response_time:	http.response_time:
min: 5	min: 2
max: 274	max: 54
median: 133	median: 22.9
p95: 179.5	p95: 34.1
p99: 186.8	p99: 44.3
http.responses: 1000	http.responses: 1000
vusers.completed: 50	vusers.completed: 50
vusers.created: 50	vusers.created: 50
vusers.created_by_name.0: 50	vusers.created_by_name.0:50
vusers.failed: 0	vusers.failed: 0
vusers.session_length:	vusers.session_length:
min: 1077.2	min: 169.9
max: 2797.7	max: 599
median: 2618.1	median: 487.9
p95: 2780	p95: 596
p99: 2780	p99: 596

- Luego utilizaremos Autocannon en línea de comandos, emulando 100 conexiones concurrentes realizadas en un tiempo de 20 segundos. Extraer un reporte con los resultados (puede ser un print screen de la consola)
 - 2) El perfilamiento del servidor con el modo inspector de node.js --inspect. Revisar el tiempo de los procesos menos performantes sobre el archivo fuente de inspección.

CON CONSOLE.LOG()

Self Tim	ne ▼	Total T	ime	Function
17164.6 ms		17164.6 ms		(idle)
1131.1 ms	24.58 %	2456.1 ms	53.38 %	▶ consoleCall

SIN CONSOLE.LOG()

Self Time	;	Total Tir	me	Function
13497.4 ms		13497.4 ms		(idle)
131.7 ms	8.03 %	131.7 ms	8.03 %	▶ getCPUs

CON CONSOLE.LOG()

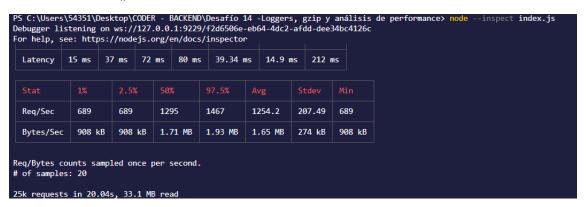
SIN CONSOLE.LOG()

```
app.get('/info', (req,
1.4 ms
        const argEntrada = pro
        const plataforma = pro
0.2 ms
        const versionNode = pr
0.6 ms
        const memoriaTotal = p
        const pathEjecucion =
0.6 ms
        const processId = prod
0.2 ms
        const carpetaProyecto
        const procesadoresNum
0.9 ms
        const array = [argEntr
4.1 ms
        console.log(array);
        res.render('pages/inf
          argumentos: argEntra
           plataforma: platafor
           versionNode: version
0.1 ms
           memoriaTotal: memori
           pathEjecucion: pathE
           processId: processId
           carpetaProyecto: car
0.2 ms
           procesadoresNum: pro
        });
```

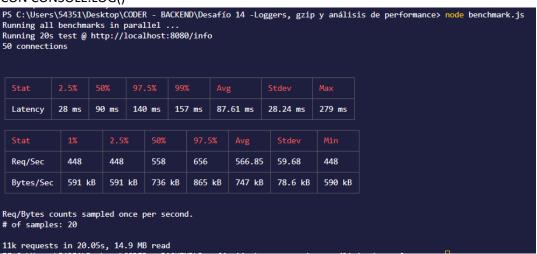
```
app.get('/info', <u>(</u>req, res<u>)</u> => {
         const argEntrada = process.argv
0.7 ms
         const plataforma = process.titl
2.1 ms
        const versionNode = process.ver
0.3 ms
0.8 ms
         const memoriaTotal = process.me
        const pathEjecucion = process.e
0.2 ms
        const processId = process.pid;
0.2 ms
         const carpetaProyecto = process
         const procesadoresNum = require
2.1 ms
0.4 ms
         const array = [argEntrada, plata
         //console.log(array);
         res.render('pages/info/info', {
0.1 ms
           argumentos: argEntrada,
          plataforma: plataforma,
           versionNode: versionNode,
          memoriaTotal: memoriaTotal,
          pathEjecucion: pathEjecucion,
          processId: processId,
           carpetaProyecto: carpetaProyec
0.1 ms
          procesadoresNum: procesadores
      });
```

3) El diagrama de flama con 0x, emulando la carga con Autocannon con los mismos parámetros anteriores.

SIN CONSOLE.LOG()



CON CONSOLE.LOG()





```
k(err);\n // T000: NODE-2882\n callback(undefined\x2C { server: operation.server\x2C session\x2C response });\
n });\n }\n\nexports.ListCollectionsCursor = ListCollectionsCursor;\n//# sourceMappingURL=list_collections_cursor.js.map
code-source-info,0xa555d277fe,492,0,1429,0006C401429,,
code-creation,Function,10,5122121,0x253fef0d36e,121, C:\Users\\54351\\Desktop\\CODER - BACKEND\\Desaf\xedo 14 -Loggers\x2C gzip y a
n\xellisis de performance\\node_modules\\mongodb\\lib\\cursor\\list_collections_cursor.js:1:1,0xa555d27778,~
code-source-info,0x253fef0d36e,492,0,1429,000C11014C15021C23036C31021C36077C370107C410145C440145C480145C500216C530216C570216C590285
n\xellisis de performance\\node modules\\mongodh\\lib\\curson\\list_collections_cursor.js:1:1,0xa555d27778,~
code-source-info,0x253fef0d36e,492,0,1429,C000C11014C15021C23036C31021C36077C370107C410145C440145C480145C500216C570216C570218C502085
C620285C670285C740383C11301323C11501353C12001428,,
code-creation,Eval,10,5122815,0x253fef01356,5, C:\\Users\\54351\\Desktop\\CODER - BACKEND\\Desaf\xedo 14 -Loggers\x2C gzip y an\xell
isis de performance\\node_modules\\mongodb\\lib\\operations\\list_collections.js:1:1,0x253fef01188,~
script-source,493,C:\\Users\\54351\\Desktop\\CODER - BACKEND\\Desaf\xedo 14 -Loggers\x2C gzip y an\xellisis de performance\\node_mod
ules\\mongodb\\lib\\operations\\list_collections.js,"use strict";\nObject.defineProperty(exports\x2C "__esNodule"\x2C \ value: true
]);\nxports.listCollectionsOperation = void 0;\nxports utils 1 = require(".defineProperty(exports\x2C "__esNodule"\x2C \ value: true
]);\nxports.listCollectionsOperation = void 0;\nxports utils 1 = require(".soperation = void 0;\nxports value: true)
]\nxports.listCollectionsOperation = void 0;\nxports \ \text{vision} \ \text
```