DESAFÍO 16: LOGGERS, GZIP Y ANÁLISIS DE PERFORMANCE

ÍNDICE

1- CONSIGNA: LOGGER Y GZIP

LOGGERS Y GZIP

Formato: link a un repositorio en Github con el proyecto cargado.
Sugerencia: no incluir los node_modules

Desafío entregable

>> Consigna:

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.

Luego implementar loggueo (con alguna librería vista en clase) que registre lo siguiente:

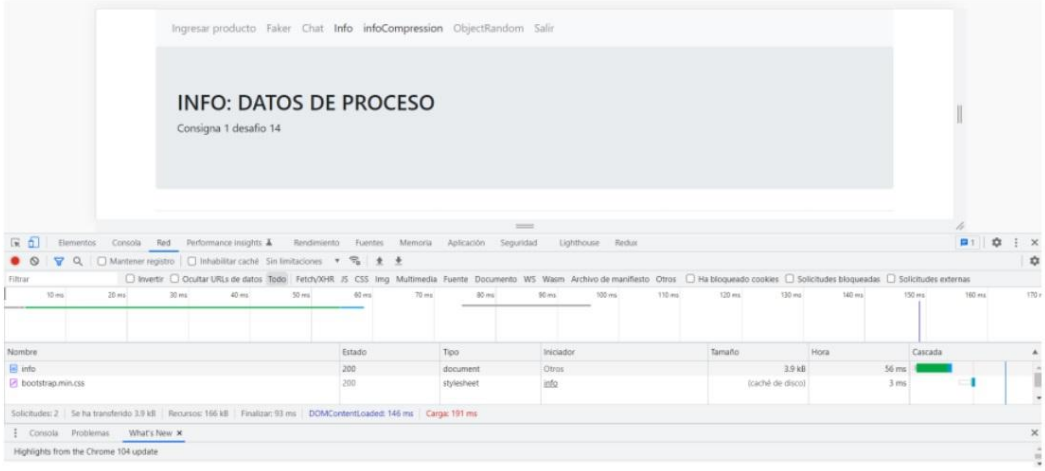
- Ruta y método de todas las peticiones recibidas por el servidor (info)
- Ruta y método de las peticiones a rutas inexistentes en el servidor (warning)
- Errores lanzados por las apis de mensajes y productos, únicamente (error)

Considerar el siguiente criterio:

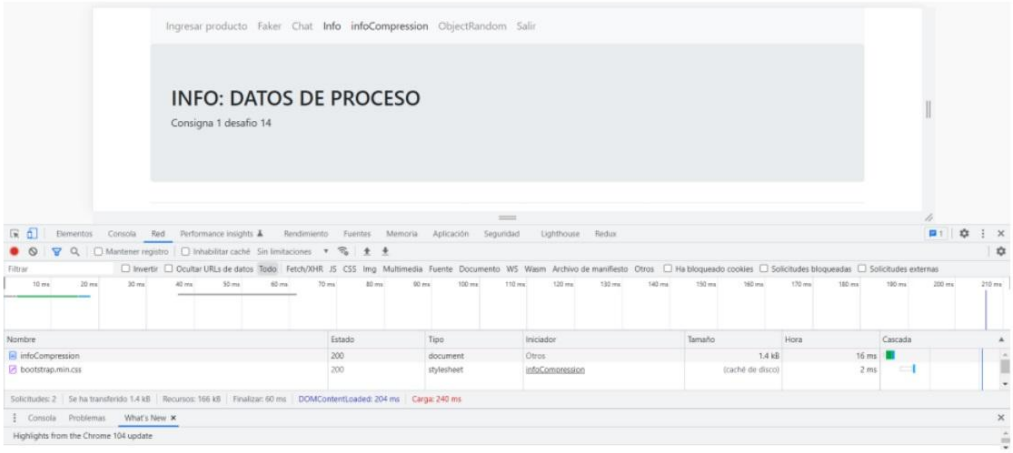
- Loggear todos los niveles a consola (info, warning y error)
- Registrar sólo los logs de warning a un archivo llamada warn.log
- Enviar sólo los logs de error a un archivo llamada error.log

CODER HOUSE

- Accediendo a info (sin compresión Gzip):



- Accediendo a info (con compresión Gzip):



- Errores guardados en “error.log”:

```
desafio-16 > error.log
1  [2023-02-13T12:26:52.748] [ERROR] errorArchive - Error ReferenceError: selectAllProucts is not defined
2  [2023-02-13T12:47:26.197] [ERROR] errorArchive - Error ReferenceError: insertPoduct is not defined
```

- Warn guardados en “warn.log”:

```
desafio-16 > warn.log
1  [2023-02-13T12:43:15.990] [WARN] warnArchive - Estado: 404, Ruta consultada: /asd, Metodo GET
2  [2023-02-13T12:45:38.899] [WARN] warnArchive - Estado: 404, Ruta consultada: /bienvenida/asdasd, Metodo GET
3  [2023-02-13T12:48:10.628] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/formProductos.js, Metodo GET
4  [2023-02-13T12:48:13.561] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/formProductos.js, Metodo GET
5  [2023-02-13T12:48:14.427] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/formProductos.js, Metodo GET
6  [2023-02-13T12:48:19.408] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/formProductos.js, Metodo GET
7  [2023-02-13T12:48:22.242] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/formProductos.js, Metodo GET
8  [2023-02-13T12:48:22.520] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/formProductos.js, Metodo GET
9  [2023-02-13T12:48:22.759] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/formProductos.js, Metodo GET
10 [2023-02-13T12:48:23.036] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/formProductos.js, Metodo GET
11 [2023-02-13T12:48:35.740] [WARN] warnArchive - Estado: 404, Ruta consultada: /favicon.ico, Metodo GET
12 [2023-02-13T12:48:35.781] [WARN] warnArchive - Estado: 404, Ruta consultada: /favicon.ico, Metodo GET
13 [2023-02-13T12:48:49.409] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/formProductos.js, Metodo GET
14 [2023-02-13T12:49:11.542] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/formProductos.js, Metodo GET
15 [2023-02-13T12:49:12.698] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/formProductos.js, Metodo GET
16 [2023-02-13T12:49:18.858] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/formProductos.js, Metodo GET
17 [2023-02-13T12:52:10.908] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/formProductos.js, Metodo GET
18 [2023-02-13T12:53:43.501] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/formProductos.js, Metodo GET
19 [2023-02-13T12:53:50.590] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/formProductos.js, Metodo GET
20 [2023-02-13T12:53:59.395] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/formProductos.js, Metodo GET
21 [2023-02-13T13:48:29.633] [WARN] warnArchive - Estado: 404, Ruta consultada: /favicon.ico, Metodo GET
22 [2023-02-13T13:58:53.001] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/randoms, Metodo POST
23 [2023-02-13T14:37:45.649] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/formProductos.js, Metodo GET
24 [2023-02-13T19:09:43.954] [WARN] warnArchive - Estado: 404, Ruta consultada: /favicon.ico, Metodo GET
25 [2023-02-13T21:23:27.049] [WARN] warnArchive - Estado: 404, Ruta consultada: /favicon.ico, Metodo GET
26 [2023-02-13T21:23:27.162] [WARN] warnArchive - Estado: 404, Ruta consultada: /favicon.ico, Metodo GET
27 [2023-02-13T21:23:27.281] [WARN] warnArchive - Estado: 404, Ruta consultada: /favicon.ico, Metodo GET
28 [2023-02-13T21:23:27.378] [WARN] warnArchive - Estado: 404, Ruta consultada: /favicon.ico, Metodo GET
29 [2023-02-13T21:23:27.505] [WARN] warnArchive - Estado: 404, Ruta consultada: /favicon.ico, Metodo GET
30 [2023-02-13T21:39:01.204] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/formProductos.js, Metodo GET
31 [2023-02-13T21:54:43.859] [WARN] warnArchive - Estado: 404, Ruta consultada: /favicon.ico, Metodo GET
32 [2023-02-13T22:10:18.098] [WARN] warnArchive - Estado: 404, Ruta consultada: /api/formProductos.js, Metodo GET
33 [2023-02-13T22:13:35.832] [WARN] warnArchive - Estado: 404, Ruta consultada: /, Metodo GET
34 [2023-02-13T22:13:40.787] [WARN] warnArchive - Estado: 404, Ruta consultada: /, Metodo GET
35 [2023-02-14T10:35:11.966] [WARN] warnArchive - Estado: 404, Ruta consultada: /json/version, Metodo GET
36 [2023-02-14T10:35:12.001] [WARN] warnArchive - Estado: 404, Ruta consultada: /json, Metodo GET
37 [2023-02-14T10:35:13.982] [WARN] warnArchive - Estado: 404, Ruta consultada: /json/version, Metodo GET
38 [2023-02-14T10:35:14.000] [WARN] warnArchive - Estado: 404, Ruta consultada: /json, Metodo GET
39 [2023-02-14T10:35:17.034] [WARN] warnArchive - Estado: 404, Ruta consultada: /json/version, Metodo GET
```

- Info por consola

```
[2023-02-14T11:18:31.382] [INFO] default -
    Ruta consultada: /
    Metodo GET
```

2- CONSIGNA: ANÁLISIS COMPLETO DE PERFORMANCE

ANÁLISIS COMPLETO DE PERFORMANCE

Formato: link a un repositorio en Github con el proyecto cargado.
Sugerencia: no incluir los node_modules

Desafío entregable

>> Consigna: Luego, realizar el análisis completo de performance del servidor con el que venimos trabajando.

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.

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

CODER HOUSE

ANÁLISIS DE PERFORMANCE: --PROF

infoController.js → con console.log(data) , sin child_process en /randoms

- Terminal 1: node –prof server.js
- Desde postman: GET http://localhost:8080/info
- Terminal 2: node –prof-process infoConsoleLog.log > infoConsoleLog.txt

```
[Summary]:
  ticks  total  nonlib   name
      9    0.3%  100.0%  JavaScript
      0    0.0%    0.0%  C++
      5    0.2%   55.6%  GC
   2566   99.7%          Shared libraries
```

infoController.js → sin console.log(data) , sin child_process en /randoms

- Terminal 1: node –prof server.js
- Desde postman: GET http://localhost:8080/info
- Terminal 2: node –prof-process infoSinConsoleLog.log > infoSinConsoleLog.txt

```
[Summary]:
  ticks  total  nonlib   name
      6    0.6%  100.0%  JavaScript
      0    0.0%    0.0%  C++
      4    0.4%   66.7%  GC
    942   99.4%          Shared libraries
```

ANÁLISIS DE PERFORMANCE: ARTILLERY

infoController.js → con console.log(data), sin child_process en /randoms

- Terminal 1: node server.js
- Terminal 2: artillery quick –-count 50 –n 20 “http://localhost:8080/info” > result_infoConConsoleLog.txt

```
Started phase 0, duration: 1s @ 10:43:33(-0300) 2023-02-14
Report @ 10:43:43(-0300) 2023-02-14
Elapsed time: 10 seconds
  Scenarios launched:  50
  Scenarios completed: 0
  Requests completed:  406
  Mean response/sec: 44.09
  Response time (msec):
    min: 31
    max: 1716
    median: 757.5
    p95: 1104.6
    p99: 1542.6
  Codes:
    200: 406

Report @ 10:43:53(-0300) 2023-02-14
Elapsed time: 20 seconds
  Scenarios launched:  0
  Scenarios completed: 29
  Requests completed:  585
  Mean response/sec: 56.92
  Response time (msec):
    min: 388
    max: 1472
    median: 638
    p95: 792.5
    p99: 1200.6
  Codes:
    200: 585
```

```
Report @ 10:43:53(-0300) 2023-02-14
Elapsed time: 20 seconds
  Scenarios launched:  0
  Scenarios completed: 21
  Requests completed:  9
  Mean response/sec: 9.09
  Response time (msec):
    min: 31
    max: 353
    median: 334
    p95: 353
    p99: 353
  Codes:
    200: 9

All virtual users finished
Summary report @ 10:43:53(-0300) 2023-02-14
  Scenarios launched:  50
  Scenarios completed: 50
  Requests completed:  1000
  Mean response/sec: 49.55
  Response time (msec):
    min: 31
    max: 1716
    median: 669
    p95: 1021.5
    p99: 1450
  Scenario counts:
    0: 50 (100%)
  Codes:
    200: 1000
```

infoController.js → sin console.log(data), sin child_process en /randoms

- Terminal 1: node server.js
- Terminal 2: artillery quick –-count 50 –n 20 “http://localhost:8080/info” > result_infoSinConsoleLog.txt

desafio-16 > result_infoSinConsoleLog.txt

```
Started phase 0, duration: 1s @ 10:15:07(-0300) 2023-02-14
Report @ 10:15:15(-0300) 2023-02-14
Elapsed time: 8 seconds
  Scenarios launched:  50
  Scenarios completed: 50
  Requests completed:  1000
  Mean response/sec: 124.07
  Response time (msec):
    min: 11
    max: 679
    median: 246
    p95: 419.5
    p99: 532.5
  Codes:
    200: 1000

All virtual users finished
Summary report @ 10:15:15(-0300) 2023-02-14
  Scenarios launched:  50
  Scenarios completed: 50
  Requests completed:  1000
  Mean response/sec: 123.76
  Response time (msec):
    min: 11
    max: 679
    median: 246
    p95: 419.5
    p99: 532.5
  Scenario counts:
    0: 50 (100%)
  Codes:
    200: 1000
```

3- CONSIGNA: ANÁLISIS COMPLETO DE PERFORMANCE

ANÁLISIS COMPLETO DE PERFORMANCE

Formato: link a un repositorio en Github con el proyecto cargado.
Sugerencia: no incluir los node_modules

Desafío entregable

>> Consigna:

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.

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

Realizar un informe en formato pdf sobre las pruebas realizadas incluyendo los resultados de todos los test (texto e imágenes).

👉 Al final incluir la conclusión obtenida a partir del análisis de los datos.

CODER HOUSE

ANÁLISIS DE PERFORMANCE: AUTOCANNON

infoController.js → con console.log(data) , sin child_process en /randoms

- Terminal 1: node server.js
- Terminal 2: autocannon –c 100 –d 20 http://localhost:8080/info

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	574 ms	796 ms	1408 ms	1582 ms	842.11 ms	191.16 ms	1897 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	15	15	114	152	116.45	29.68	15
Bytes/Sec	58.8 kB	58.8 kB	447 kB	596 kB	456 kB	116 kB	58.8 kB

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

2k requests in 20.15s, 9.12 MB read
```

infoController.js → sin console.log(data) , sin child_process en /randoms

- Terminal 1: node server.js
- Terminal 2: autocannon –c 100 –d 20 http://localhost:8080/info

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	375 ms	538 ms	1028 ms	1300 ms	586.16 ms	185.67 ms	1508 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	85	85	163	253	168.5	47.58	85
Bytes/Sec	333 kB	333 kB	638 kB	991 kB	660 kB	186 kB	333 kB

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

3k requests in 20.13s, 13.2 MB read
```

ANÁLISIS DE PERFORMANCE: --INSPECTOR

infoController.js → con/sin console.log(data) , sin child_process en /randoms

- Terminal 1: node –inspect server.js
- Acceder: chrome://inspect
- Terminal 2: artillery quick –count 50 –n 20 “http://localhost:8080/info” > result_infoConConsoleLog.txt
- Terminal 3: artillery quick –count 50 –n 20 “http://localhost:8080/info” > result_infoSinConsoleLog.txt

Gruesa (de abajo hacia arriba)

Tiempo individual		Tiempo total		Función
233922.7 ms		233922.7 ms		(idle)
3905.2 ms	20.81 %	6631.7 ms	35.34 %	consoleCall
2078.5 ms	11.08 %	2078.5 ms	11.08 %	writeUtf8String
482.6 ms	2.57 %	482.6 ms	2.57 %	stat
377.4 ms	2.01 %	377.4 ms	2.01 %	(garbage collector)
359.7 ms	1.92 %	359.7 ms	1.92 %	open
346.8 ms	1.85 %	346.8 ms	1.85 %	access
342.0 ms	1.82 %	342.0 ms	1.82 %	getCPUs
298.3 ms	1.59 %	298.3 ms	1.59 %	(program)
291.1 ms	1.55 %	876.0 ms	4.67 %	compile
182.6 ms	0.97 %	182.6 ms	0.97 %	writev
172.5 ms	0.92 %	172.5 ms	0.92 %	writeBuffer
171.1 ms	0.91 %	276.3 ms	1.47 %	scanLine
163.7 ms	0.87 %	2097.6 ms	11.18 %	session

ANÁLISIS DE PERFORMANCE: 0X

infoController.js → con console.log(data) , sin child_process en /randoms

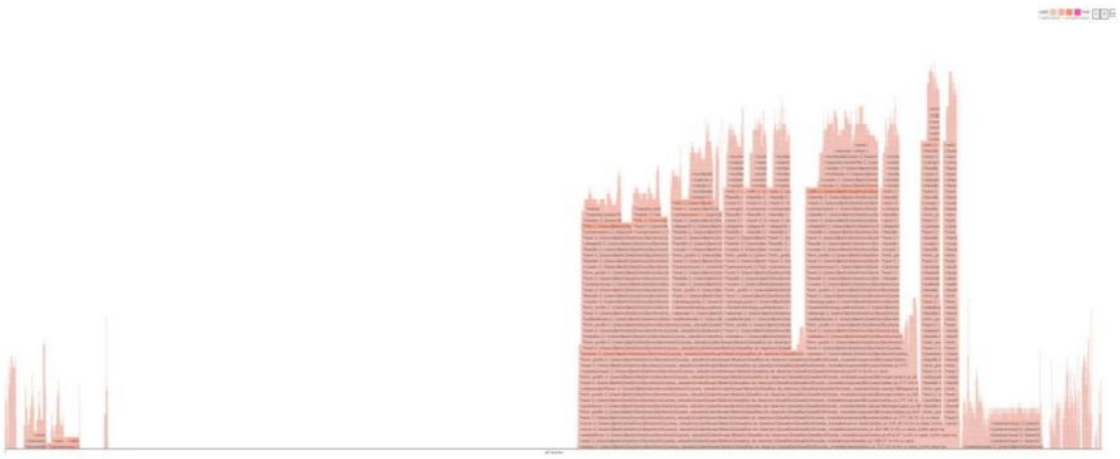
- Terminal 1: 0x server.js
- Terminal 2: autocannon –c 100 –d 20 http://localhost:8080/info
- Obtengo el gráfico

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	302 ms	473 ms	800 ms	944 ms	504.69 ms	120.16 ms	1343 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	98	98	196	295	194.5	41.85	98
Bytes/Sec	384 kB	384 kB	768 kB	1.16 MB	762 kB	164 kB	384 kB

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

4k requests in 20.12s, 15.2 MB read
```



infoController.js → sin console.log(data) , sin child_process en /randoms

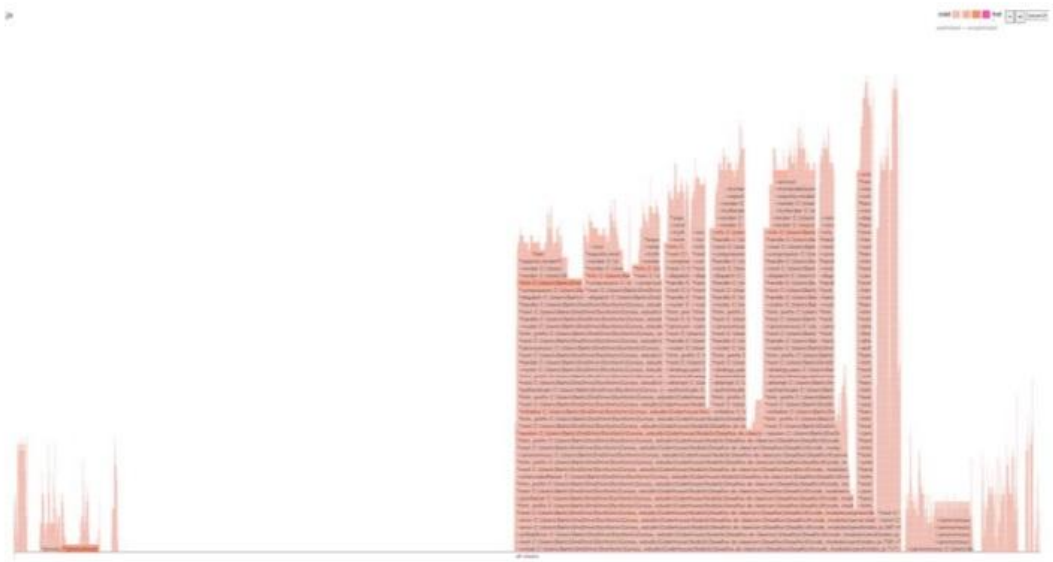
- Terminal 1 : 0x server.js
- Terminal 2: autocannon –c 100 –d 20 http://localhost:8080/info
- Obtengo el gráfico

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	241 ms	342 ms	663 ms	722 ms	361.72 ms	100.68 ms	1107 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	152	152	294	377	273.3	52.17	152
Bytes/Sec	596 kB	596 kB	1.15 MB	1.48 MB	1.07 MB	204 kB	596 kB

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

6k requests in 20.09s, 21.4 MB read
```



4- CONCLUSIÓN

Como se puede ver en los ejemplos anteriores, siempre es recomendable no utilizar funciones bloqueantes, ya que las mismas empeoran la performance del servidor. También, es muy importante correr el servidor en modo Cluster, así el mismo corre en varios procesadores.