Al incorporar el middleware compression a /info, no noté cambios significativos, incluso la version comprimida (/infoZip) tenia un peso levemente mayor en bytes, entiendo que es por el bajo contenido comprimible que posee este endpoint de mi api , Luego, intenté tal como se realizó en clase, añadir a ambas respuestas mucho contenido de tipo texto y ahi si note una gran diferencia de tamaño entre el endpoint sin compresion y el que si la tenia.

Consigna 2
Desactivo el child_process de random
MODO FORK
SIN console.log en /info
[Summary]:
ticks total nonlib name
5 0.3% 100.0% JavaScript
0 0.0% 0.0% C++
14 0.8% 280.0% GC
1716 99.7% Shared libraries

CON console.log en /info
[Summary]:
ticks total nonlib name
7 0.3% 100.0% JavaScript
0 0.0% 0.0% C++
18 0.7% 257.1% GC
2491 99.7% Shared libraries

CONCLUSION: Puede verse en el summary luego de ejecutar el prof de node y luego artillery con la cantidad de instancias y request indicadas en la consigna que la ejecucion sin proceso bloqueante llevo casi la mitad de ticks que la ejecucion con proceso bloqueante.
Se incluyen en el repo la carpeta "performance" con las extracciones de reportes relacionados al analisis.

Consigna 3

1. node inspect: se puede observar que el proceso bloqueante ocasiona mucha mas demora en la ejecucion que el no bloqueante utilizando esta herramienta en chrome inspec

2.

```
84
85  app.use('/info', (req, res, next) => {
86    try {
87  0.8 ms  const info = {
88        ruta: '/info',
89  3.4 ms    NUMERO_DE_PROCESADORES: os.cpus().length,
90  0.2 ms    ARGUMENTOS_ENTRADA: process.argv,
91  0.5 ms    SISTEMA_OPERATIVO: process.platform,
92  0.2 ms    VERSION_NODE: process.version,
93  0.6 ms    MEMORIA_RSS: process.memoryUsage().rss,
94  0.2 ms    PATH_EJECUCION: process.execPath,
95  0.1 ms    PROCESS_ID: process.pid,
96  0.4 ms    CARPETA_PROYECTO: process.cwd()
97      }
98  9.7 ms  return res.json(info)
99    } catch (error) {
100     next(error)
101   }
102 })
103
104 app.use('/info-bloq', (req, res, next) => {
105   try {
106 0.6 ms  const info = {
107       ruta: '/info',
108 0.2 ms    NUMERO_DE_PROCESADORES: os.cpus().length,
109 0.6 ms    ARGUMENTOS_ENTRADA: process.argv,
110 0.2 ms    SISTEMA_OPERATIVO: process.platform,
111 0.2 ms    VERSION_NODE: process.version,
112 0.4 ms    MEMORIA_RSS: process.memoryUsage().rss,
113 0.2 ms    PATH_EJECUCION: process.execPath,
114       PROCESS_ID: process.pid,
115 0.2 ms    CARPETA_PROYECTO: process.cwd()
116     }
117 5.2 ms  console.log(info)
118 18.7 ms  return res.json(info)
119   } catch (error) {
120     next(error)
121   }
122 })
```

AUTOCANNON - Reporte no bloqueante

```
Running 20s test @ http://localhost:8080/info
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	102 ms	138 ms	240 ms	265 ms	145.94 ms	38.39 ms	341 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	523	523	683	795	681.65	82.66	523
Bytes/Sec	332 kB	332 kB	434 kB	505 kB	433 kB	52.5 kB	332 kB

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

14k requests in 20.07s, 8.66 MB read
```

AUTOCANNON - Reporte bloqueante

```
Running 20s test @ http://localhost:8080/info-bloq
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	137 ms	326 ms	434 ms	492 ms	326.19 ms	60.66 ms	549 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	247	247	301	346	304.25	25.47	247
Bytes/Sec	157 kB	157 kB	191 kB	220 kB	193 kB	16.2 kB	157 kB

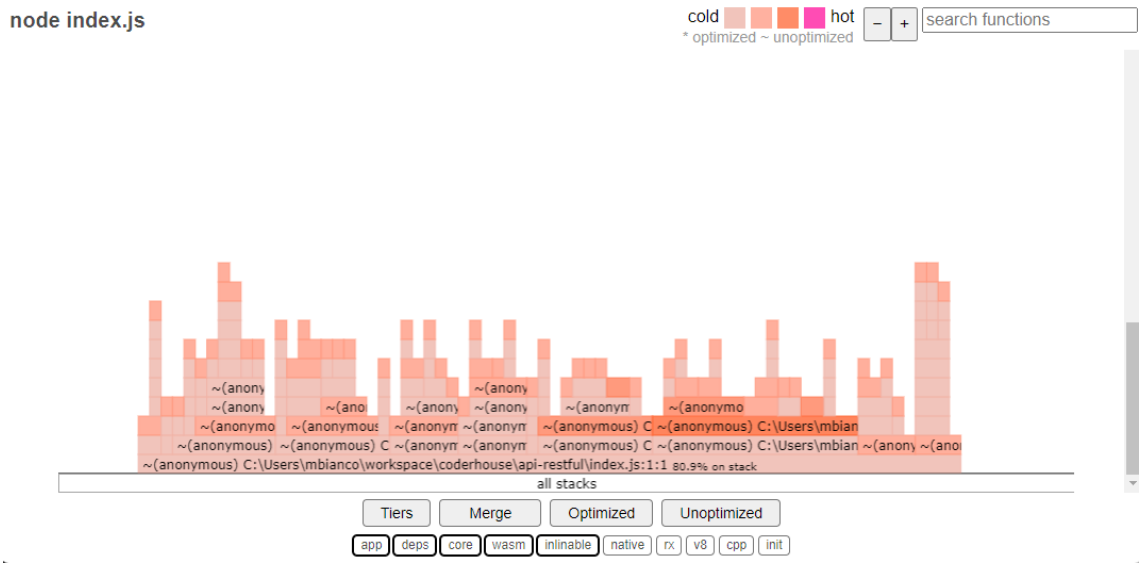
```
Req/Bytes counts sampled once per second.
# of samples: 20

6k requests in 20.07s, 3.86 MB read
```

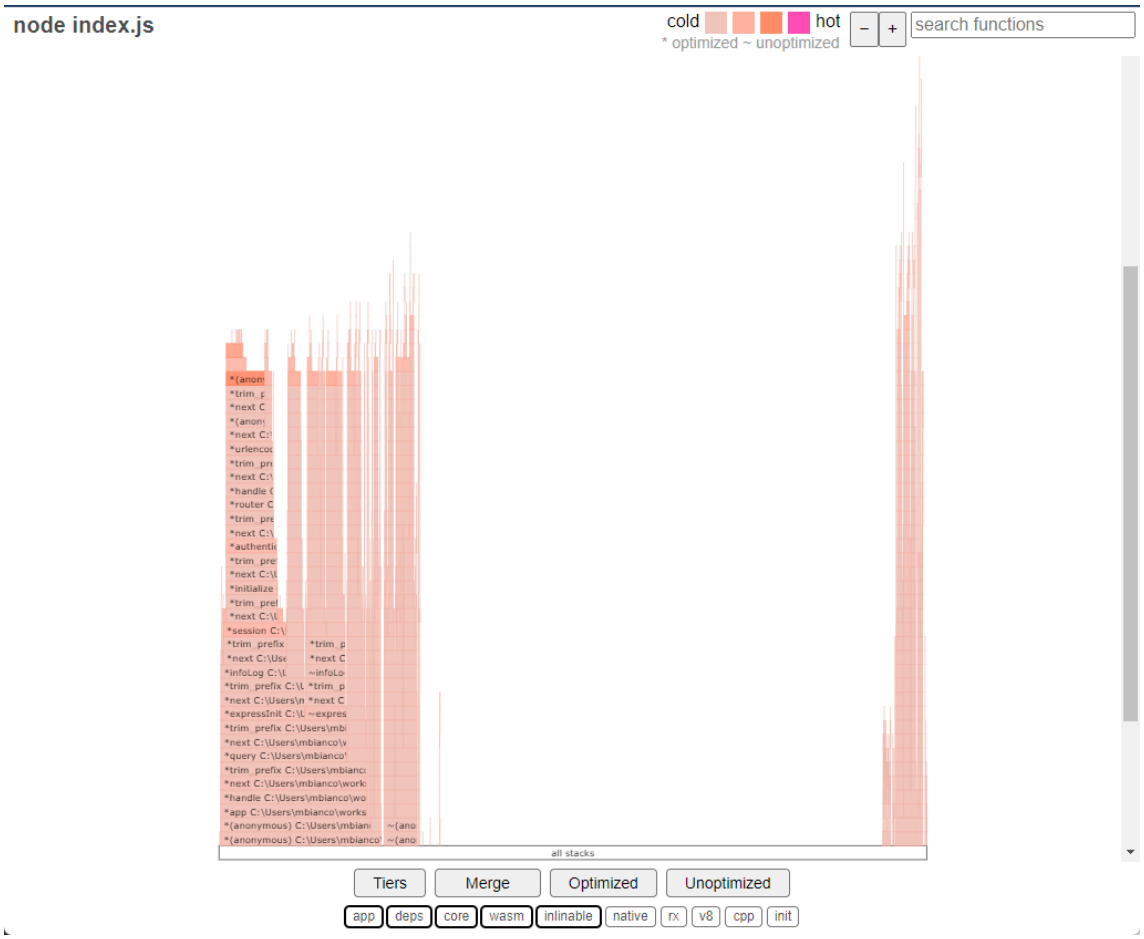
CONCLUSION: Se nota el decrimento de respuesta cuando se trata de un proceso bloqueante

FLAMEGRAPHS

BLOQUEANTE



NO BLOQUEANTE


node index.js
cold
hot
* optimized ~ unoptimized
search functions
all stacks
Tiers
Merge
Optimized
Unoptimized
app
deps
core
wasm
inlinable
native
rx
v8
cpp
init