

TEST RUTA COMPRIMIDA: /api/info

Sin la librería Compresión como middleware corriendo

```
const MODEO = args.MODEO;  
//const compression = require("compression");  
//  
//app.use(compression())  
app.use(express.json());  
app.use(express.urlencoded({ extended: true }));
```

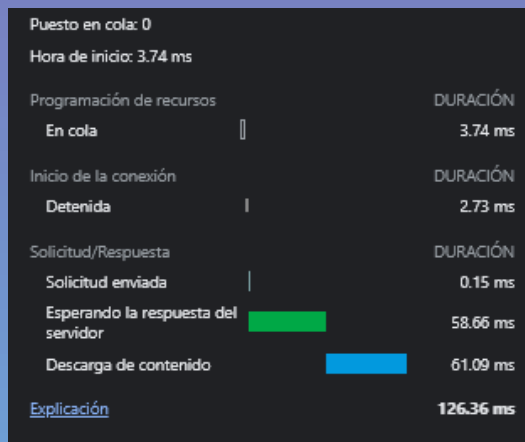
Texto comentado
en app.js

REQUEST DE http://localhost:8080/api/info



Checkbox "on" inhabilitando caché de los datos.

Se observa que el tamaño de la respuesta enviada por el servidor al cliente es de 957 bites y tardo 123ms



EN DETALLE:
Respuesta en 1.26ms -
de descarga 61.09ms

HEADERS

Encabezados	Vista previa	Respuesta	Iniciador	Tiempos
General				
Solicitar URL: http://localhost:8080/api/info				
Método de la solicitud: GET				
Código de estado: 200 OK				
Dirección remota: [::1]:8080				
Política de referencias: strict-origin-when-cross-origin				
Encabezados de respuesta				
Connection: keep-alive				
Content-Length: 549				
Content-Type: application/json; charset=utf-8				
Date: Sat, 13 Aug 2022 22:31:03 GMT				
ETag: W/"225-ZjSvQ57I36IzzlqDwv5Jn7nhCPI"				
Keep-Alive: timeout=5				
Set-Cookie: connect.sid=s%3AVry_PLf_R2DpL73gjsUKtwZOp-y190				
X-Powered-By: Express				
Encabezados de solicitud				
Accept: text/html,application/xhtml+xml,application/xml;q				
Accept-Encoding: gzip, deflate, br				
Accept-Language: es-ES				
Cache-Control: no-cache				
Connection: keep-alive				
Host: localhost:8080				
Pragma: no-cache				
Sec-Fetch-Dest: document				
Sec-Fetch-Mode: navigate				
Sec-Fetch-Site: none				
Sec-Fetch-User: ?1				
Sec-GPC: 1				
Upgrade-Insecure-Requests: 1				
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) App				

Se observa que hay 8
encabezados en la
respuesta

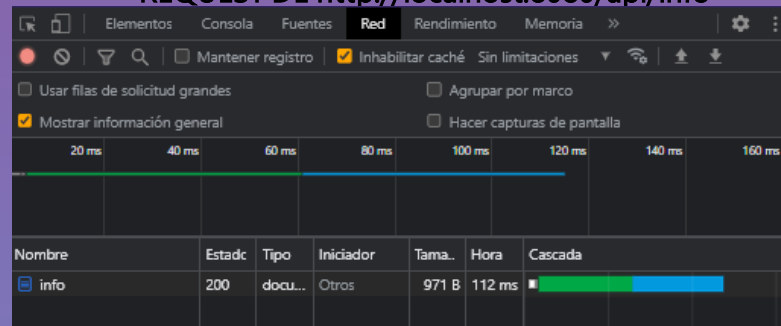
CON la librería Compresión como middleware corriendo

```
const MODEO = args.MODEO;
const compression = require("compression");

app.use(compression())
app.use(express.json());
```

Se quitan las '//'
app.js y se deja
correr el proceso.

REQUEST DE http://localhost:8080/api/info



Checkbox "on" inhabilitando caché de los datos.

Se observa que el tamaño aumentó 14 bites, pasó de 957 bites a 971 bites (interpreto que el bloque de datos y su contenido es tan pequeño que gzip sólo alcanza a sumar su estructura de compresión y no comprimir efectivamente) y tardó 112ms, unos 12ms menos que en la request sin compresión.

Puesto en cola: 0	
Hora de inicio: 2.79 ms	
Programación de recursos	
En cola	2.79 ms
Inicio de la conexión	
Detenida	0.99 ms
Solicitud/Respuesta	
Solicitud enviada	0.12 ms
Esperando la respuesta del servidor	56.95 ms
Descarga de contenido	54.03 ms
Explicación	114.89 ms

EN DETALLE:
Respuesta en 114ms -
de descarga 54.03ms,
unos 7ms menos

HEADERS

Encabezados Vista previa Respuesta Iniciador Tiempos Cookies

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

Connection: keep-alive

Content-Length: 549

Content-Type: application/json; charset=utf-8

Date: Sat, 13 Aug 2022 22:33:19 GMT

ETag: W/"225-Bkpq73h6+AZ+s3pmeK/Rimn8xfg"

Keep-Alive: timeout=5

Set-Cookie: connect.sid=s%3AVry_PLf_R2DpLJ3gjsUKtWzOp-y19C-g.28Uk6

Vary: Accept-Encoding

X-Powered-By: Express

Encabezados de solicitud

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,

Accept-Encoding: gzip, deflate, br

Accept-Language: es-ES

Cache-Control: no-cache

Connection: keep-alive

Cookies: connect.sid=s%3AVry_PLf_R2DpLJ3gjsUKtWzOp-y19C-g.28Uk6

Host: localhost:8080

Pragma: no-cache

Sec-Fetch-Dest: document

Sec-Fetch-Mode: navigate

Sec-Fetch-Site: none

Sec-Fetch-User: ?1

Sec-GPC: 1

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit

Se observa que fue
agregado el encabezado
VARY: Accept-Encoding

PROFILING con ARTILLERY:

/api/info - /api/infolog

```
router.get("/info", (req, res) => {
  logger.info(`Método Http: ${JSON.stringify(req.method)}, Ruta [${JSON.stringify(req.path)}]`);
  let info = {
    ArgumetosEntrada: GV.argv,
    NombrePlataforma: GV.platform,
    VersionNodeJS: GV.version,
    MemoriaTotalReservada: GV.memoryUsage().rss,
    PathEjecucion: GV.execPath,
    ProcessID: GV.pid,
    CarpetaProyecto: GV.cwd(),
    NumeroDeCPUS: cpus().length
  }

  res.json(info);
});

router.get("/infolog", (req, res) => {
  logger.info(`Método Http: ${JSON.stringify(req.method)}, Ruta [${JSON.stringify(req.path)}]`);

  let info = {
    ArgumetosEntrada: GV.argv,
    NombrePlataforma: GV.platform,
    VersionNodeJS: GV.version,
    MemoriaTotalReservada: GV.memoryUsage().rss,
    PathEjecucion: GV.execPath,
    ProcessID: GV.pid,
    CarpetaProyecto: GV.cwd(),
    NumeroDeCPUS: cpus().length
  }

  console.log(info)

  res.json(info);
});
```

Se hizo una división de la ruta /info para facilitar el paso a paso del test:

/api/info (sin console.log)

/api/infolog (con console.log)

```
//RUTA DE EJEMPLO http://localhost:8080/api/randoms/?amount=12
router.get("/randoms/", (req, res) => {
  logger.info(`Método Http: ${JSON.stringify(req.method)}, Ruta [${JSON.stringify(req.path)}]`);
  const amount = req.query.amount;
  const child = fork("./src/routes/child.js");
  amount == undefined || amount == NaN
  ? child.send({ amount: 10000000 })
  : child.send({ amount });
  child.on("message", (message) => {
    res.render("randoms", {
      text: `El servidor express (Nginx) en ${PORT} en PID ${
        process.pid
      } - ${new Date().toLocaleString()}`,
      message,
    });
  });
});
```

A pedido del desafío en la ruta /randoms se deja comentada la pieza de código con el child Forkeado y se la deja lista para accionar.

```
/*
//RUTA DE EJEMPLO http://localhost:8080/api/randoms/?amount=12
router.get("/randoms/", (req, res) => {
  logger.info(`Método Http: ${JSON.stringify(req.method)}, Ruta [${JSON.stringify(req.path)}]`);
  const amount = req.query.amount;
  const child = fork("./src/routes/child.js");
  amount == undefined || amount == NaN
  ? child.send({ amount: 10000000 })
  : child.send({ amount });
  child.on("message", (message) => {
    res.render("randoms", {
      text: `El servidor express (Nginx) en ${PORT} en PID ${
        process.pid
      } - ${new Date().toLocaleString()}`,
      message,
    });
  });
});
*/

router.get("/randoms/", (req, res) => {
  logger.info(`Método Http: ${JSON.stringify(req.method)}, Ruta [${JSON.stringify(req.path)}]`);

  const amount = req.query.amount;

  amount == undefined || amount == NaN ? (amount = 10000000) : amount;

  const message = count(amount);

  res.render("randoms", { text: `Iniciado en ${PORT} con PID ${process.pid} - ${new Date().toLocaleString()}`, message });
});
```

Pasos para el perfilamiento de /api/info

2) En una consola en paralelo se ejecuta artillery

```
PROBLEMAS  SALIDA  TERMINAL  SQL CONSOLE  CONSOLA DE DEPURACIÓN

Ari@DESKTOP-PPR8KQD MINGW64 ~/Documents/repobackend/curso-ch-backend/desafioClase32 (main)
$ node --prof ./src/app.js -p 8080 -m FORK
{"level":"info","message":"Server iniciado en modo:[FORK] con id de Proceso [12228] en puerto [8080]"}
{"level":"info","message":"Conectado a la base de datos de los Usuarios"}
```

1) Se abre consola y se inicia en --prof y modo fork, se crea el archivo isolate

*--prof: Genera una salida, un archivo de 'ticks' o marcas en el mismo directorio de ejecución local de la aplicación, del profiler V8. El profiler almacena en ese archivo la actividad registrada en el test.

```
isolate-0000020E271705F0-12228-v8.log
package-lockjson
packagejson
```

```
Ari@DESKTOP-PPR8KQD MINGW64 ~/Documents/repobackend/curso-ch-backend/desafioClase32 (main)
$
```

```
package-lockjson
packagejson
prueba1.log
```

3) Se cierra consola y el proceso. Se cambia el nombre al archivo log, por "prueba 1.log"

```
Ari@DESKTOP-PPR8KQD MINGW64 ~/Documents/repobackend/curso-ch-backend/desafioClase32 (main)
$ node --prof-process prueba1.log > result_prueba1.log
stdout is not a tty

Ari@DESKTOP-PPR8KQD MINGW64 ~/Documents/repobackend/curso-ch-backend/desafioClase32 (main)
$ node.exe --prof-process prueba1.log > result_prueba1.log
(node:14848) ExperimentalWarning: VM Modules is an experimental feature. This feature could change at any time
(Use 'node --trace-warnings ...' to show where the warning was created)
```

```
result_prueba1.log
warn.log
```

4) Se ejecuta --prof-process, para imprimir el resultado de la prueba1

*Desconozco el motivo, pero el --prof-process solo se ejecuta con node.exe

Para que el archivo isolate tenga sentido, necesitamos usar el procesador de ticks que esta incluido en Node.js. Para ejecutar el procesador, se usa la flag --prof-process sobre el isolate (que ahora es prueba1.log)

```
Ari@DESKTOP-PPR8KQD MINGW64 ~/Documents/repobackend/curso-ch-backend/desafioClase32 (main)
$ artillery quick --count 20 -n 50 "http://localhost:8080/api/info"
Running scenarios...
Phase started: unnamed (index: 0, duration: 1s) 18:45:56(-0300)

Phase completed: unnamed (index: 0, duration: 1s) 18:45:57(-0300)

-----
Metrics for period to: 18:46:00(-0300) (width: 3.618s)
-----

http.codes.200: ..... 126
http.request_rate: ..... 38/sec
http.requests: ..... 135
http.response_time:
  min: ..... 3
  max: ..... 939
  median: ..... 210.6
  p95: ..... 415.8
  p99: ..... 757.6
http.responses: ..... 126
vusers.created: ..... 20
vusers.created_by_name.0: ..... 20

All VUs finished. Total time: 25 seconds

-----
Summary report @ 18:46:19(-0300)
-----

http.codes.200: ..... 1000
http.request_rate: ..... 23/sec
http.requests: ..... 1000
http.response_time:
  min: ..... 3
  max: ..... 1031
  median: ..... 122.7
  p95: ..... 572.6
  p99: ..... 742.6
http.responses: ..... 1000
vusers.completed: ..... 20
vusers.created: ..... 20
vusers.created_by_name.0: ..... 20
vusers.failed: ..... 0
vusers.session_length:
  min: ..... 19203.8
  max: ..... 21216.9
  median: ..... 20136.3
  p95: ..... 20958.1
  p99: ..... 20958.1
```

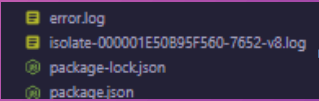

Pasos para el perfilamiento de /api/infolog

2) En una consola en paralelo se ejecuta artillery

1) Se abre consola y se inicia en --prof y modo fork, se crea el archivo isolate

```
PROBLEMAS SALIDA TERMINAL SQL CONSOLE CONSOLA DE DEPURACIÓN

Ari@DESKTOP-PPR8KQD MINGW64 ~/Documents/repobackend/curso-ch-backend/desafioClase32 (main)
$ node --prof ./src/app.js -p 8080 -m FORK
{"level":"info","message":"Server iniciado en modo:[FORK] con id de Proceso [12228] en puerto [8080]"}
{"level":"info","message":"Conectado a la base de datos de los Usuarios"}
```



```
PROBLEMAS SALIDA TERMINAL SQL CONSOLE CONSOLA DE DEPURACIÓN

}
{"level":"info","message":"Método Http: \"GET\", Ruta [\"/infolog\"]"}
{
  ArgumetosEntrada: [
    'C:\\Program Files\\nodejs\\node.exe',
    'C:\\Users\\Ari\\Documents\\repobackend\\curso-ch-backend\\desafioClase32\\src\\app.js',
    '-p',
    '8080',
    '-m',
    'FORK'
  ],
  NombrePlataforma: 'win32',
  VersionNodeJS: 'v16.16.0',
  MemoriaTotalReservada: 123827456,
  PathEjecucion: 'C:\\Program Files\\nodejs\\node.exe',
  ProcessID: 7652,
  CarpetaProyecto: 'C:\\Users\\Ari\\Documents\\repobackend\\curso-ch-backend\\desafioClase32',
  NumeroDeCPUS: 4
}
{"level":"info","message":"Método Http: \"GET\", Ruta [\"/infolog\"]"}
{
  ArgumetosEntrada: [
    'C:\\Program Files\\nodejs\\node.exe',
    'C:\\Users\\Ari\\Documents\\repobackend\\curso-ch-backend\\desafioClase32\\src\\app.js',
    '-p',
    '8080',
    '-m',
    'FORK'
  ],
  NombrePlataforma: 'win32',
  VersionNodeJS: 'v16.16.0',
  MemoriaTotalReservada: 123827456,
  PathEjecucion: 'C:\\Program Files\\nodejs\\node.exe',
  ProcessID: 7652,
  CarpetaProyecto: 'C:\\Users\\Ari\\Documents\\repobackend\\curso-ch-backend\\desafioClase32',
  NumeroDeCPUS: 4
}
```

*Ya se puede observar diferencias de trabajo en la gitbash se estan imprimiendo 20 veces por cada rafaga los datos consultados.

```
Ari@DESKTOP-PPR8KQD MINGW64 ~/Documents/repobackend/curso-ch-backend/desafioClase32 (main)
$ artillery quick --count 20 -n 50 "http://localhost:8080/api/infolog"
Running scenarios...
Phase started: unnamed (index: 0, duration: 1s) 10:30:44(-0300)

Phase completed: unnamed (index: 0, duration: 1s) 10:30:45(-0300)

-----
Metrics for period to: 10:30:50(-0300) (width: 5.604s)
-----

http.codes.200: ..... 236
http.request_rate: ..... 46/sec
http.requests: ..... 255
http.response_time:
  min: ..... 8
  max: ..... 1390
  median: ..... 190.6
  p95: ..... 550.1
  p99: ..... 772.9
http.responses: ..... 236
vusers.created: ..... 20
vusers.created_by_name.0: ..... 20

All WJs finished. Total time: 25 seconds

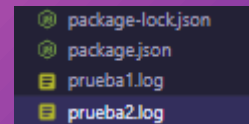
-----
Summary report @ 10:31:07(-0300)
-----

http.codes.200: ..... 1000
http.request_rate: ..... 24/sec
http.requests: ..... 1000
http.response_time:
  min: ..... 8
  max: ..... 1390
  median: ..... 175.9
  p95: ..... 539.2
  p99: ..... 671.9
http.responses: ..... 1000
vusers.completed: ..... 20
vusers.created: ..... 20
vusers.created_by_name.0: ..... 20
vusers.failed: ..... 0
vusers.session_length:
  min: ..... 19060.9
  max: ..... 20771.8
  median: ..... 20136.3
  p95: ..... 20543.1
  p99: ..... 20543.1
```

```
PROBLEMAS  SALIDA  TERMINAL  SQL CONSOLE  CONSOLA DE DEPURACIÓN

'-m',
'FORK'
],
NombrePlataforma: 'win32',
VersionNodeJS: 'v16.16.0',
MemoriaTotalReservada: 123494400,
PathEjecucion: 'C:\\Program Files\\nodejs\\node.exe',
ProcessID: 7652,
CarpetaProyecto: 'C:\\Users\\Ari\\Documents\\repobackend\\curso-ch-backend\\desafioClase32',
NumeroDeCPUS: 4
}

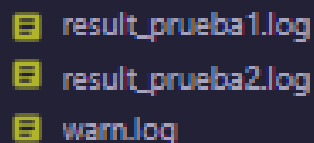
Ari@DESKTOP-PPRBKQD MINGW64 ~/Documents/repobackend/curso-ch-backend/desafioClase32 (main)
$
```



3) Se cierra consola y el proceso. Se cambia el nombre al archivo log, por "prueba2.log"

```
Ari@DESKTOP-PPRBKQD MINGW64 ~/Documents/repobackend/curso-ch-backend/desafioClase32 (main)
$ node.exe --prof-process prueba2.log > result_prueba2.log
(node:15064) ExperimentalWarning: VM Modules is an experimental feature. This feature could change at any time
(Use 'node --trace-warnings ...' to show where the warning was created)
```

4) Se ejecuta --prof-process, para imprimir el resultado de la prueba2



Teniendo el resultado desglosado y con los dos archivos ya creados, se realizan las primeras observaciones.

Datos del perfilamiento vista individual de /api/info

```
[Summary]:
  ticks  total  nonlib   name
    34     0.2% 100.0% JavaScript
     0     0.0%   0.0% C++
    27     0.2%  79.4% GC
 16349    99.8%         Shared libraries
```

Summary en el podemos observar:

Ticks: la cantidad de datos aleatorios extraídos de la aplicación que desarrollé.

Total: el número total de muestras recolectadas.

Nonlib: el porcentaje de código que no es de una libraries.

Name: en que lenguaje se encuentra desarrollado.

```
[JavaScript]:
  ticks  total  nonlib   name
     3     0.0%   8.8% LazyCompile: *serializeInto
     3     0.0%   8.8% LazyCompile: *deserializeOb
     2     0.0%   5.9% LazyCompile: *next C:\Users
     2     0.0%   5.9% Function: ^realpathSync nod
     1     0.0%   2.9% RegExp: ^[!$%&'*+.^_`|-@-9
     1     0.0%   2.9% LazyCompile: *write node:bu
```

JavaScript Esto muestra que se recopilaban 34 datos aleatorios, que esos datos representan el 0.2% del total de muestras que se recolectaron al ejecutar el profiling. Que el 100% de lo redactado no pertenece o esta compartido a una librerie.

```
[Shared libraries]:
  ticks  total  nonlib   name
  15833    95.4%         C:\WINDOWS\SYSTEM32\ntdll.dll
    712    4.3%         C:\Program Files\nodejs\node.exe
     3     0.0%         C:\WINDOWS\System32\KERNEL32.DLL
     1     0.0%         C:\WINDOWS\System32\KERNELBASE.dll
```

Shared libraries Esto muestra del proceso de profiling se recolectaron muestras por un total de 16349 que pertenecen a librerias compartidas y que eso fue representativamente el 99.8% de las muestras recopiladas. Si vemos en detalle "Shared Libraries": se registra que el entorno de tiempo de ejecución de Node.js ocupa el 4.3% y las funciones del kernel de Windows ocupan el resto (puntualmente el ntdll.dll que es un archivo que contiene todas las posibles funciones que están relacionadas con la capa del NT kernel).

```
[C++ entry points]:
  ticks  cpp  total   name
```

C++ Si ejecutamos en nuestro proceso de análisis con:

artillery quick --count 20 -n 50 "http://localhost:8080/api/randoms"

Es previsible un aumento significativo del registro que recopila C++, porque se producirían más interacciones en el procesamiento dentro de la aplicación de node.

Datos del perfilamiento vista individual de /api/infolog

```
[Summary]:
```

ticks	total	nonlib	name
63	0.2%	98.4%	JavaScript
8	0.0%	0.0%	C++
29	0.1%	45.3%	GC
32652	99.8%		Shared libraries
1	0.0%		Unaccounted

Summary en el podemos observar:

Unaccounted: Se debe a que no fue recopilado un dato y este fue ignorado voluntariamente, puede ser por una incompatibilidad o por falta de adaptación entre node y una librería.

```
1 0.0% 1.6% Function: ^<anonymous> node:internal/crypto/utl1:114:46
1 0.0% 1.6% Function: ^<anonymous> node:buffer:349:36
1 0.0% 1.6% Function: ^<anonymous> C:\Users\Ari\Documents\repobackend\curso-ch-backend\desafioClase32\src\routes\mainRoutes.js:46:24
1 0.0% 1.6% Function: ^<anonymous> C:\Users\Ari\Documents\repobackend\curso-ch-backend\desafioClase32\node_modules\mongodb\lib\sdam\mon
```

JavaScript Se puede observar que el 1.6% del 0.2%, de los datos observados y recopilados pertenece la función que se encuentra dentro de la ruta get /infolog

```
[Shared libraries]:
```

ticks	total	nonlib	name
31697	96.8%		C:\WINDOWS\SYSTEM32\ntdll.dll
951	2.8%		C:\Program Files\nodejs\node.exe
3	0.0%		C:\WINDOWS\System32\KERNEL32.DLL
1	0.0%		C:\WINDOWS\System32\KERNELBASE.dll

Shared libraries Hay un incremento de actividad recopilada por el archivo ntdll, se debe a que cada rafaga de artillery ejecuta un console.log; y esa pieza esta realizando llamadas al core de NT. Puntualmente a este archivo "ntdll.dll", el cual almacena funciones "de bajo nivel" del kernel de windows.


```
mainRoutes.js result_prueba1.log ... result_prueba2.log
desafioClase32 > result_prueba1.log
43
44 [Summary]:
45 ticks total nonlib name
46 34 0.2% 100.0% JavaScript
47 0 0.0% 0.0% C++
48 27 0.2% 79.4% GC
49 16349 99.8% Shared libraries
50
result_prueba2.log
desafioClase32 > result_prueba2.log
60
61 [Summary]:
62 ticks total nonlib name
63 63 0.2% 98.4% JavaScript
64 0 0.0% 0.0% C++
65 29 0.1% 45.3% GC
66 32652 99.8% Shared libraries
67 1 0.0% Unaccounted
```

result_prueba1.log		result_prueba2.log	
desafioClase32 > result_prueba1.log		desafioClase32 > result_prueba2.log	
[Shared libraries]:		[Shared libraries]:	
ticks	total nonlib name	ticks	total nonlib name
15633	95.4%	31697	96.6%
712	4.3%	951	2.9%
3	0.0%	3	0.0%
1	0.0%	1	0.0%

CONCLUSIÓN

Lo que podemos inferir es que la aplicación, en la segunda prueba, incorporó una pieza de código que la lleva a volcar más trabajo en la CPU.

En este caso, canaliza esas tareas en funciones de bajo nivel, a través de NT core. Lo que en una prueba real sería un síntoma, pero no necesariamente el problema. Puesto que nuestro objetivo sería buscar dentro de la aplicación, algo que esté en el código y que ocupe mayor poder de procesamiento del CPU, pero que no necesariamente reclame o vuelque trabajo de funciones por debajo del sistema en el NT de Windows.

INICIO DE PROFILING CON INSPECT - AUTOCANNON Y 0x

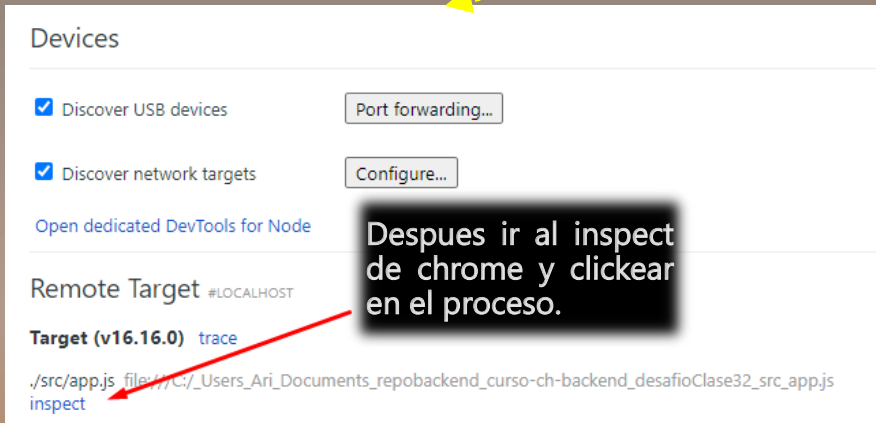
Para iniciar es necesario poner un doble flag (--) antes de elegir el modo inspect

```
PROBLEMAS SALIDA TERMINAL SQL CONSOLE CONSOLA DE DEPURACIÓN
Ejecutando tarea: npm run start

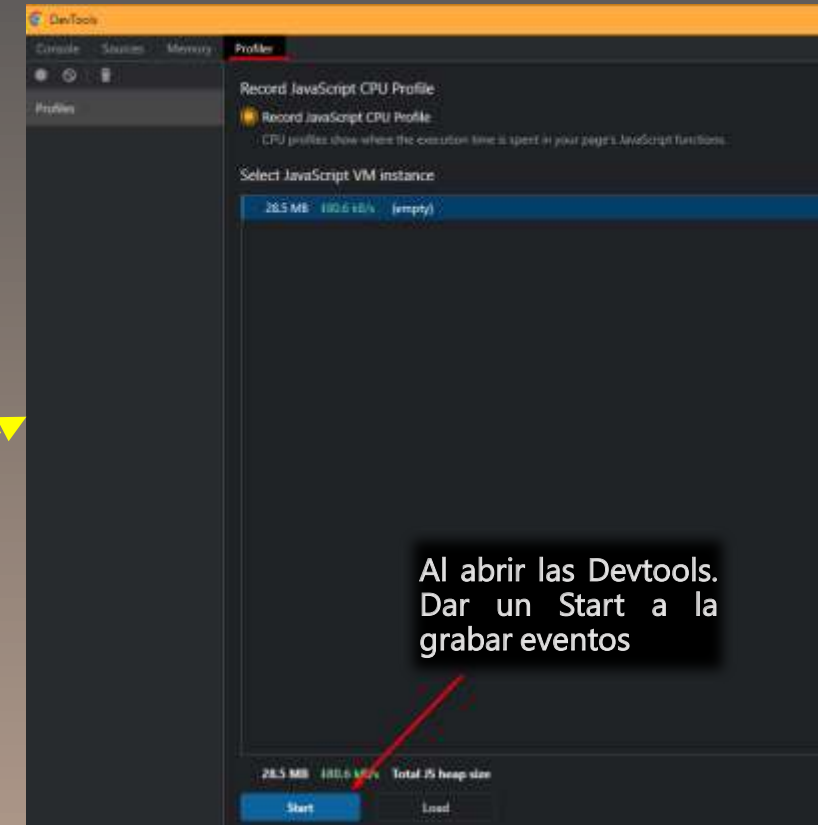
npm WARN config global '--global', '--local' are deprecated. Use '--location=global' instead.

> desafioClase22@1.0.0 start
> 0x -- node --inspect ./src/app.js

ProfilingDebugger listening on ws://127.0.0.1:9229/c36a2f80-c502-44f7-bb3b-449bc6b1602d
For help, see: https://nodejs.org/en/docs/inspector
{"level":"info","message":"Server iniciado en modo:[FORK] con id de Proceso [496] en puerto [8080]"}
{"level":"info","message":"Conectado a la base de datos de los Usuarios"}
Debugger attached.
Debugger ending on ws://127.0.0.1:9229/c36a2f80-c502-44f7-bb3b-449bc6b1602d
For help, see: https://nodejs.org/en/docs/inspector
```



Despues ir al inspect de chrome y clickear en el proceso.



Al abrir las Devtools. Dar un Start a la grabar eventos

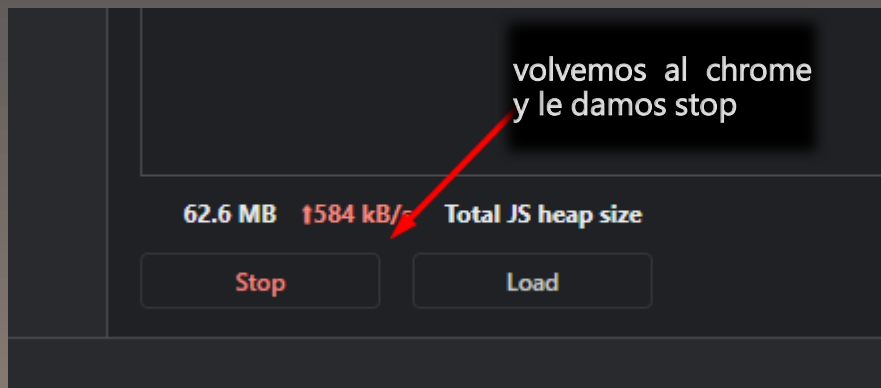
Se da inicio al test en una consola paralela

```
Ari@DESKTOP-PPR8KQO MINGW64 ~/Documents/repobackend/curso-ch-backend/desafioClase32 (main)
$ npm run test
npm WARN config global '--global', '--local' are deprecated. Use '--location=global' instead.

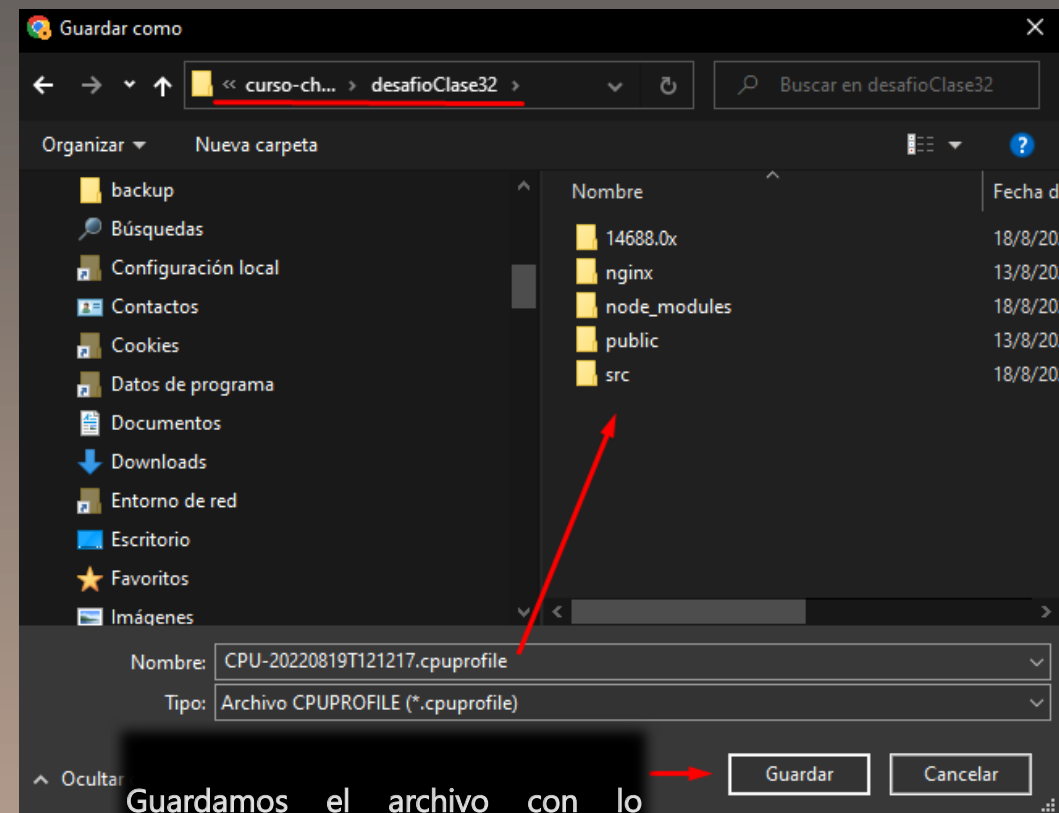
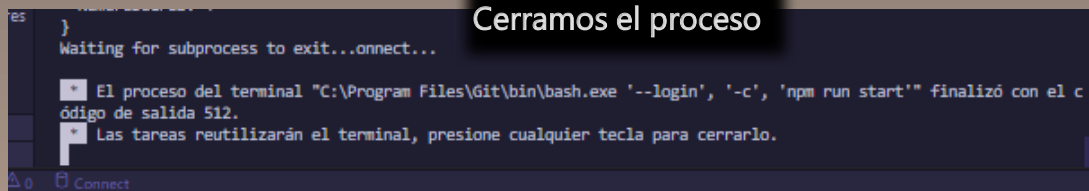
> desafioClase32@1.0.0 test
> node ./src/autocannon.js

Running all benchmarks in parallel...
|
```

volvemos al chrome y le damos stop



Cerramos el proceso



Guardamos el archivo con lo recopilado por las devtools de chrome en la carpeta del proyecto

```

28
29
30 router.get("/info", (req, res) => {
31   20.8 ms logger.info(`Método Http: ${JSON.stringify(req.method)}, Ruta [${JSON.stringify(req.path)}]`);
32   1.5 ms let info = {
33     0.8 ms ArgumetosEntrada: GV.argv,
34     0.2 ms NombrePlataforma: GV.platform,
35     VersionNodeJS: GV.version,
36     1.8 ms MemoriaTotalReservada: GV.memoryUsage().rss,
37     0.2 ms PathEjecucion: GV.execPath,
38     ProcessID: GV.pid,
39     0.7 ms CarpetaProyecto: GV.cwd(),
40     3.6 ms NumeroDeCPUS: cpus().length
41   }
42
43   21.3 ms res.json(info); TOTAL
44   });
45
46 router.get("/infoLog", (req, res) => {
47   17.7 ms logger.info(`Método Http: ${JSON.stringify(req.method)}, Ruta [${JSON.stringify(req.path)}]`);
48
49   1.8 ms let info = {
50     2.1 ms ArgumetosEntrada: GV.argv,
51     0.2 ms NombrePlataforma: GV.platform,
52     0.2 ms VersionNodeJS: GV.version,
53     1.0 ms MemoriaTotalReservada: GV.memoryUsage().rss,
54     0.5 ms PathEjecucion: GV.execPath,
55     0.5 ms ProcessID: GV.pid,
56     0.3 ms CarpetaProyecto: GV.cwd(),
57     3.1 ms NumeroDeCPUS: cpus().length
58   }
59
60   10.8 ms console.log(info)
61
62   23.4 ms res.json(info); TOTAL
63
64   });
65

```

En el proceso podemos ver como las funciones de las rutas fueron los proceso que consumieron mayor cantidad de tiempo.

Y a su vez, podemos ver como la ruta /infoLog predispuso una carga mayor de datos con una demora total mayor a 2 segundos


```

autocannon.js X
desafioClase32 > src > autocannon.js > ...
1  const autocannon = require("autocannon");
2  const { PassThrough } = require("stream");
3
4  const run = (url) => {
5    const buf = [];
6    const outputStream = new PassThrough();
7    //estas dos constantes un array vacio y un stream de
8    const inst = autocannon({
9      url,
10     connections: 100,
11     duration: 20,
12   });
13   //aca se define como hacer la prueba
14   autocannon.track(inst, { outputStream });
15   //aca se define la inspección de la prueba y ese out
16   outputStream.on("data", (data) => buf.push(data));
17   //aca indica que es en el array vacío BUF
18   inst.on("done", () => {
19     //cuando termina de capturar todos los datos que f
20     process.stdout.write(Buffer.concat(buf));
21   });
22 }
23 console.log("Running all benchmarks in parallel...");
24
25 run("http://localhost:8080/api/info");
26 run("http://localhost:8080/api/infoLog");
27 //aca se ven que rutas se van a ejecutar
28

```

```

< Depurar
"scripts": {
  "test": "node ./src/autocannon.js",
  "start": "0x ./src/app.js"
},
"keywords": [],

```

```

Ari@DESKTOP-PPRBKQD MINGW64 ~/Documents/repobackend/curso-ch-backend/desafioClase32 (main)
$ npm run start
npm WARN config global '--global', '--local' are deprecated. Use '--location=global' instead.

```

```

> desafioClase22@1.0.0 start
> 0x ./src/app.js

```

```

Profiling{"level":"info","message":"Server iniciado en modo:[FORK] con id de Proceso [14036] en puerto [8080]"}
{"level":"info","message":"Conectado a la base de datos de los Usuarios"}
0

```

```

Ari@DESKTOP-PPRBKQD MINGW64 ~/Documents/repobackend/curso-ch-backend/desafioClase32 (main)
$ npm run test

```

```
> desafioclase22@1.0.0 test
> node ./src/autocannon.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	1640 ms	2053 ms	3000 ms	3175 ms	2118.86 ms	312.96 ms	3742 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	0	0	32	100	44.6	37.51	2
Bytes/Sec	0 B	0 B	28.1 kB	87.7 kB	39.1 kB	32.9 kB	1.75 kB

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

```
992 requests in 20.24s, 782 kB read
Running 20s test @ http://localhost:8080/api/infolog
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	1740 ms	2067 ms	3810 ms	3908 ms	2225.01 ms	510.54 ms	4339 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	0	0	24	90	43.55	33.8	8
Bytes/Sec	0 B	0 B	21 kB	79 kB	38.2 kB	29.6 kB	7.01 kB

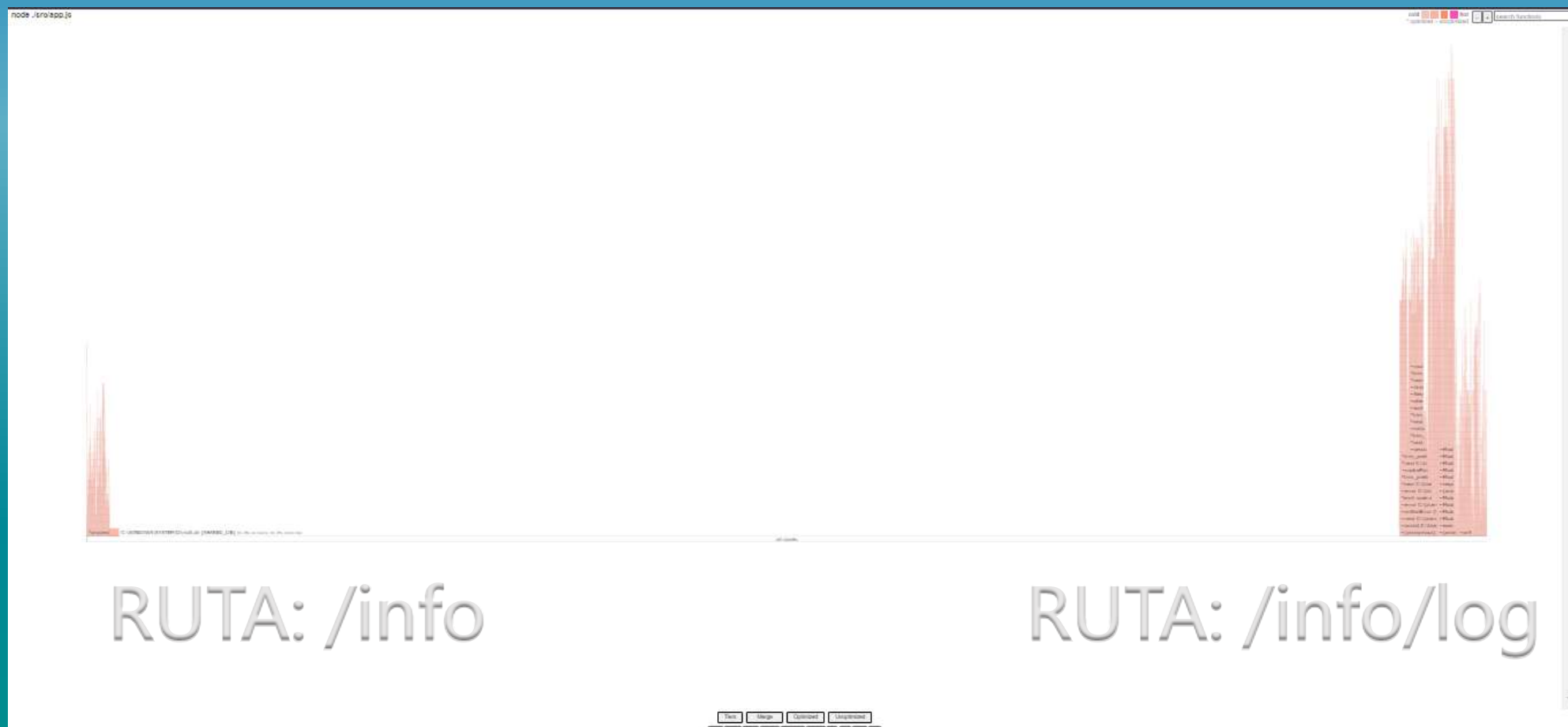
```
Req/Bytes counts sampled once per second.
# of samples: 20
```

```
971 requests in 20.21s, 764 kB read
```

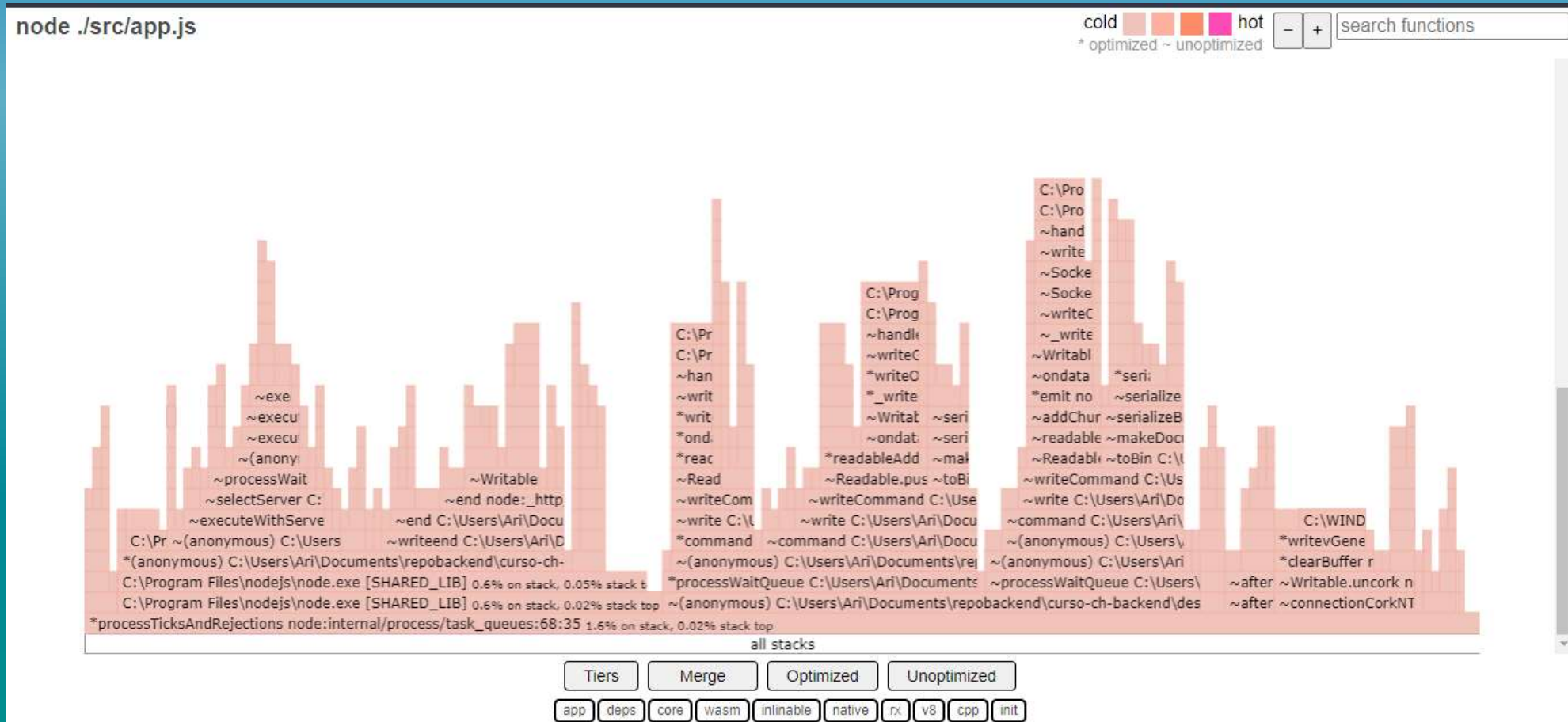
```
Ari@DESKTOP-PPR8KQD MINGW64 ~/Documents/repobackend/curso-ch-backend/desafioClase32 (main)
Flamegraph generated in
file:///C:/Users/Ari/Documents/repobackend/curso-ch-backend/desafioClase32/14688.0x/flamegraph.html
```

Flamegraph

STACK COMPLETO



DETALLE RUTA: /info



DETALLE RUTA: /infolog

node ./src/app.js

cold hot - + search functions
* optimized ~ unoptimized



Tiers Merge Optimized Unoptimized
app deps core wasm inlinable native rx v8 cpp init

OBSERVACIONES:

Si observamos la altura del diagrama, vemos que el stack de Node en la ruta /infolog se vuelve más profundo.

Entre las procesos que se encuentran al tope del stack en la ruta /infolog, se encuentran llamados a librerías.

CONCLUSIONES:

Se puede inferir que los procesos han tenido mayor tiempo de espera, aguardando la resolución de otros procesos bloqueantes en el caso de la ruta /infolog. Del mismo modo podemos ver en la vista horizontal como los procesos de la ruta /info, han logrado que desarrollarse en menor cantidad de tiempo.

Igualmente por los colores que vemos y por lo breve que son los procesos en la ruta /infolog, y extrema delgadez, en el stack, que no llegan a bloquear a los demás procesos en ningún momento.