Desarrollo de Aplicaciones Web Empresariales

Exequiel Fuentes Lettura
exequiel.fuentes@ucn.cl

Información de contacto

- Exequiel Fuentes Lettura
  - Email: exequiel.fuentes@ucn.cl
  - Horario de Atención: Jueves y Viernes, bloque C

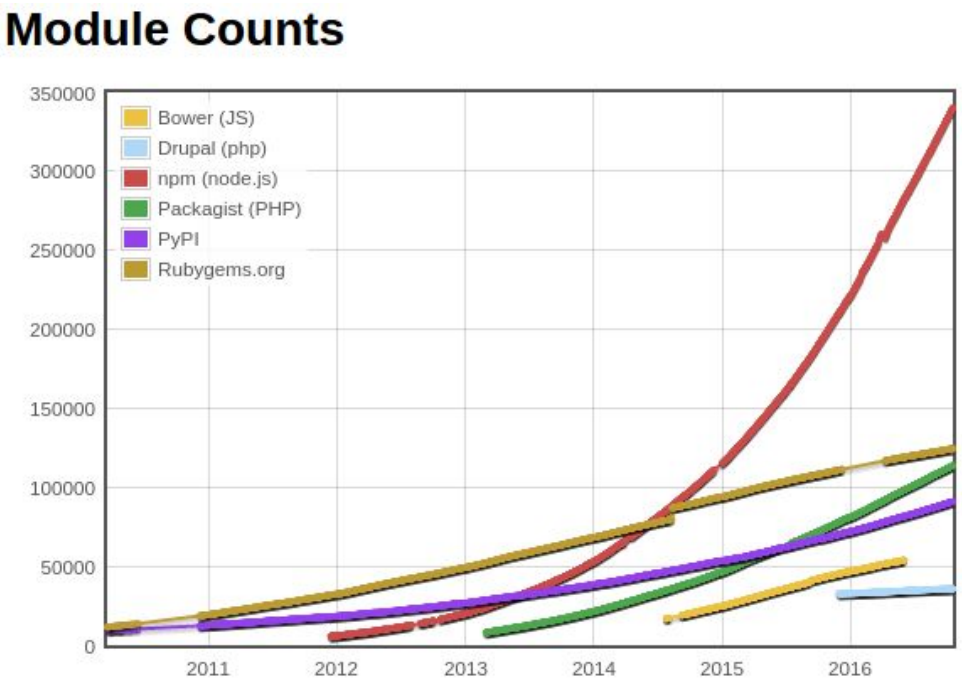
- Departamento de Ingeniería de Sistemas y Computación
  - Oficina: Y1 - 329
  - http://www.disc.ucn.cl

Arquitectura de aplicaciones Web: Node.js

Material:

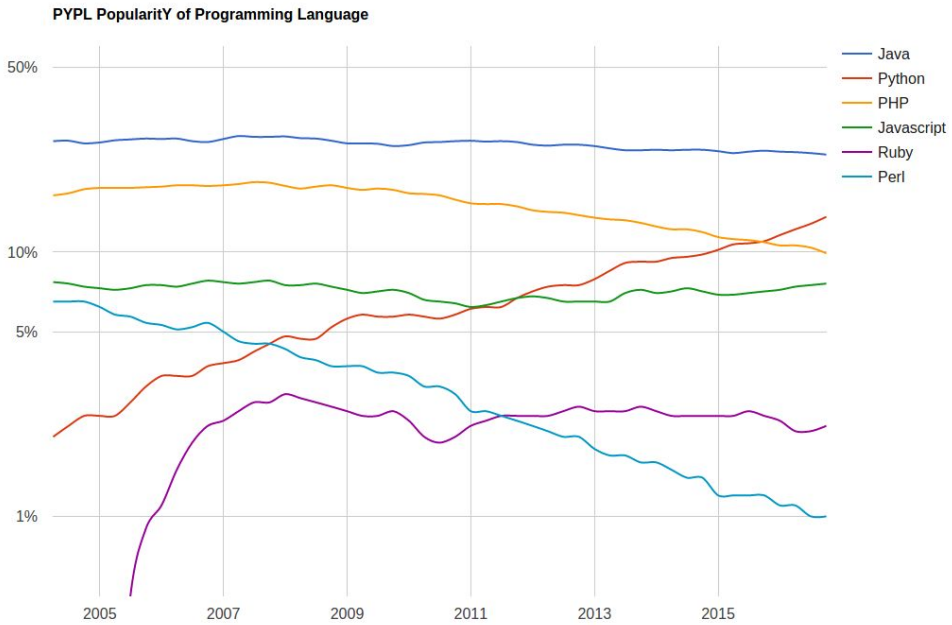
- https://nodejs.org/en/docs/
- http://www.tutorialspoint.com/nodejs/

Algunas estadísticas



- http://www.modulecounts.com/ analiza los sitios de los lenguajes más importantes para determinar la descarga.

Algunas estadísticas



- http://pypl.github.io/PYPL.html analiza la frecuencia de búsqueda de tutoriales de lenguajes de programación en Google.
- Quiénes usan Node.js: https://www.quora.com/What-companies-are-using-Node-js-in-production

Introducción: Hilos vs Eventos

Hilos/Hebras/Threads

```
request = readRequest(socket);
reply = processRequest(request);
sendReply(socket, reply);
```

Implementación: Se crea un hilo, bloquea en espera que termine el proceso.

Eventos

```
readRequest(socket, function(request) {
    processRequest(request,
        function (reply) {
            sendReply(socket, reply);
        });
});
```

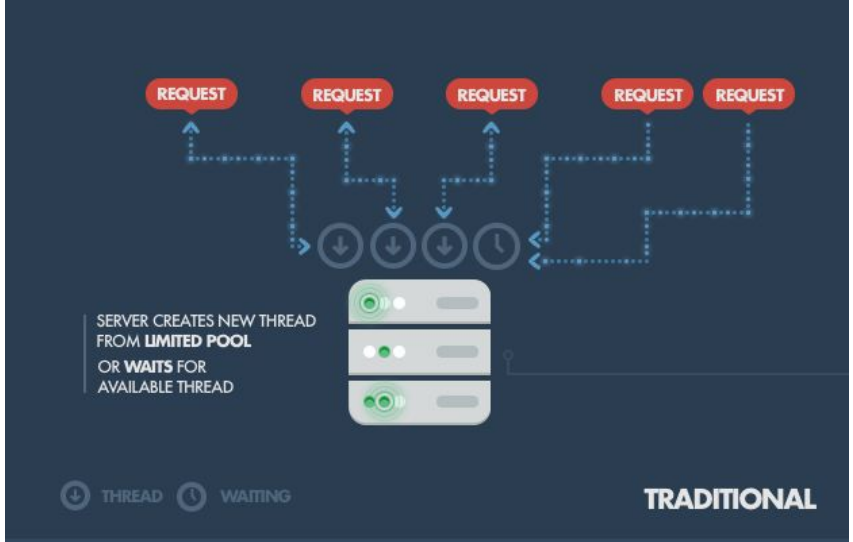
Implementación: Procesado por una cola de eventos.

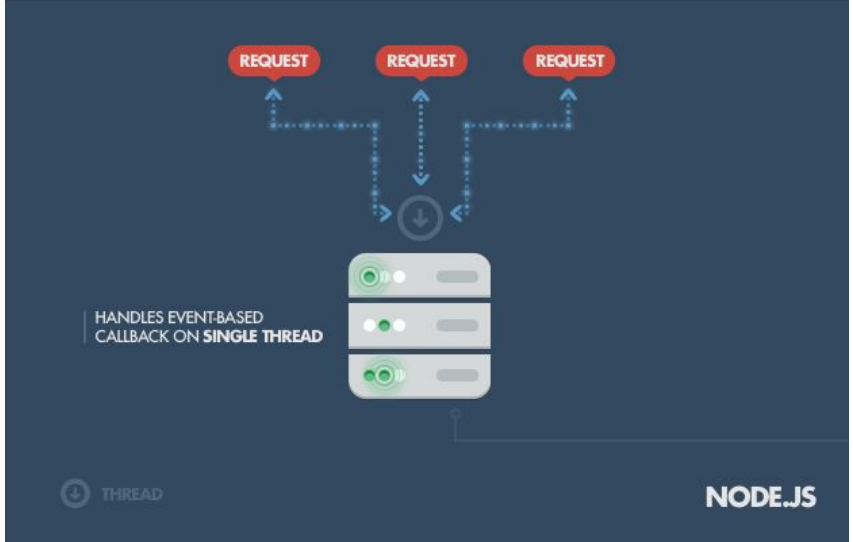
Cola de eventos

- Un ciclo sin fin a la espera de eventos. Ejemplo:
  - while (true) {
    if (!eventQueue.notEmpty()) {
    eventQueue.pop().call();
    }
    }
- Nunca se espera o se bloquea un manejador de eventos. Ejemplo:
  - launchReadRequest(socket); // Retorna inmediatamente
  - eventQueue.push(readDoneEventHandler); // Cuando lee, termina

Node.js

- Construído sobre el motor de Javascript de Chrome
  - Se tiene Javascript en el browser y en el servidor.
  - No se necesita DOM en el servidor.
- Es un framework asíncrono y basado en eventos
  - Todo se ejecuta como una llamada desde el ciclo de eventos.
- La interfaz de eventos permite acceder a todas las operaciones del SO.
  - Empaqueta todas las llamadas al sistema operativo (archivos, socket/red)
  - Soporte para manejo de datos.


REQUEST
REQUEST
REQUEST
REQUEST
REQUEST
SERVER CREATES NEW THREAD
FROM LIMITED POOL
OR WAITS FOR
AVAILABLE THREAD
THREAD
WAITING
TRADITIONAL


REQUEST
REQUEST
REQUEST
HANDLES EVENT-BASED
CALLBACK ON SINGLE THREAD
THREAD
NODE.JS

Diferencia entre PHP (y muchos) y Node.js

- En PHP:

```php
$response = file_get_contents("http://example.com");
print_r($response);
```

- En Node:
  - Permite realizar otras tareas mientras se obtiene la respuesta.
  - No almacena en memoria.

```node
var http = require('http');

http.request({ hostname: 'example.com' }, function(res) {
  res.setEncoding('utf8');
  res.on('data', function(chunk) {
    console.log(chunk);
  });
}).end();
```

Node.js

- Es ideal para aplicaciones que tengan alta demanda de entrada/salida, streaming, etc.
- npm es el administrador oficial de paquetes de Node.js.
  - Instala las dependencias localmente.
  - Puede manejar múltiples versiones del mismo módulo.
  - Puede especificar las dependencias.
- Una aplicación en Node.js consiste principalmente de:
  - Importar módulos requeridos.
  - Crear un servidor.
  - Leer una solicitud y retornar una respuesta.

Node.js: Módulos

- Para obtener otras dependencias se utiliza “requiere”
  - const http = require('http');
  - const dns = require('dns');
- También se puede importar archivos con una ruta relativa:
  - const miArchivo = require('./miArchivo.extension');
- Cuando se importa un directorio, se obtiene:
  - const miDir = require('./miDir'); // Lee miDir/index.js
- Instalar módulos es fácil:
  - $ npm install express
  - const express = require('express');

Node.js: Módulos

- Los módulos no son inyectados en el alcance global, son asignados a variables.
- Los módulos no comparten el mismo alcance.
- Un módulo se carga sólo una vez.
- A diferencia del browser, no hay el objeto global window, pero existe: globals y process (no se recomienda agregar propiedades en ellos).

Node.js: Módulos - Ejemplo: Leyendo un archivo

```
// Se importa el modulo fs que empaqueta llamadas al sistema
// para manejo de archivos
const fs = require("fs");

// Recordar que es una llamada asíncrona
fs.readFile("small.txt", readDoneCallback);

// Por convencion el primer argumento es objeto JavaScript Error
// dataBuffer es un objeto de tipo Buffer
function readDoneCallback(error, dataBuffer) {
    //console.log(dataBuffer); // Imprime un objeto de la clase Buffer
    if (!error) {
        console.log(dataBuffer.toString());
    }
}
```

La clase Buffer

- Manipular una gran cantidad de datos no es una fortaleza para un motor de Javascript.
  - Desafortunadamente es lo que se hace en servidores Web.
- Node tiene la clase Buffer para optimizar el manejo de datos
  - La interfaz luce como un arreglo de bytes.
  - La memoria se pide fuera del heap del motor.
- Usado para empaquetar llamadas I/O (fs, net, etc).
- Puede hacer conversiones:
  - buf.toString("utf8");
  - buf.toString("hex");
  - buf.toString("base64");

Programando con Events/Callbacks

Hilos

```
r1 = step1();
console.log('step1 done', r1);
r2 = step2(r1);
console.log('step2 done', r2);
r3 = step3(r2);
console.log('step3 done', r3);
console.log('All Done!');
```

Callbacks

```
step1(function(r1) {
    console.log('step1 done', r1);
    step2(r1, function (r2) {
        console.log('step2 done', r2);
        step3(r2, function (r3) {
            console.log('step3 done',r3);
        });
    });
});
console.log('Hecho!'); // Malo!!
```

Programando con Events/Callbacks

Hilos

```
r1 = step1();
console.log('step1 done', r1);
r2 = step2(r1);
console.log('step2 done', r2);
r3 = step3(r2);
console.log('step3 done', r3);
console.log('All Done!');
```

Callbacks

```
step1(function(r1) {
    console.log('step1 done', r1);
    step2(r1, function (r2) {
        console.log('step2 done', r2);
        step3(r2, function (r3) {
            console.log('step3 done',r3);
            console.log('Hecho!'); // Malo!!
        });
    });
});
```

Node.js: Callbacks

- Un programa asíncrono no retorna un valor cuando la función termina
  - Utiliza el estilo conocido por continuation-passing style (CPS).
  - La función recibe un parámetro adicional que representa la continuación de la función. La función invocará la continuación recibida pasando el valor de retorno.
- Ejemplo:

```
const dns = require('dns');

// La funcion resolve4 se utiliza para resolver la direccion IPv4 de un hostname
// https://nodejs.org/api/dns.html#dns_dns_resolve4_hostname_callback
dns.resolve4('www.google.com', function(err, addresses) {
    if (err) throw err;
    console.log('addresses: ' + JSON.stringify(addresses));
});
```

Node.js: Eventos

- El patrón callback funciona bien sirve cuando se quiere notificar cuando una función termina.
- Sin embargo, hay situaciones que requieren notificar diferentes eventos que no ocurren al mismo tiempo.
- Para este tipo de situaciones, se utiliza el patrón EventEmitter.
- EventEmitter permite emitir un evento a todos los suscritos
    - Este concepto existe en el browser: se agrega un manejador de eventos en el DOM.

EventEmitter

```
const EventEmitter = require('events').EventEmitter;
const emitter = new EventEmitter();

let contador = 0;

// Notar que esta sintaxis is ES6
// https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Functions/Arrow_functions
emitter.on('event', (p1) => {
    console.log("Evento", p1, "ocurrio", ++contador, "veces");
});

setInterval(function() {
    emitter.emit('event', 'a');
}, 3000);
```


Típico patrón EventEmitter

- Tener múltiples eventos diferentes para diferentes estados o acciones:
  - emitter.on('condicionA', doConditionA);

    emitter.on('condicionB', doConditionB);

    emitter.on('condicionC', doConditionC);

    emitter.on('error', handleErrorCondition);
- Manejar “error” es importante, Node termina cuando un error no es atrapado
  - emitter.emit('error', new Error('Ouch!'));

Node.js: Streams

- Interfaz para manipular un flujo continuo de datos de manera asíncrona.
- El mayor beneficio que no es necesario adquirir todos los datos en memoria.
- Permite producir y/o consumir stream de datos.
- La API de Node utiliza ampliamente esto:
  - Leer. Ejemplo: fs.createReadStream
  - Escribir. Ejemplo: fs.createWriteStream
  - Duplex stream (Leer y Escribir). Ejemplo: net.createConnection
  - Transform stream. Ejemplo: zlib, crypto

Node.js: Streams - Leer datos

```
const fs = require("fs");
const readableStreamEvent = fs.createReadStream("big.txt");

readableStreamEvent.on('data', function (chunkBuffer) {
    console.log('Se obtuvo un trozo', chunkBuffer.length, 'bytes');
});

readableStreamEvent.on('end', function() {
    console.log('Se obtuvo todo el archivo');
});

readableStreamEvent.on('error', function (err) {
    console.error('Hmm, tenemos un error aca!', err);
});
```


Node.js: Streams - Escribir datos

```
const fs = require("fs");
const writableStreamEvent = fs.createWriteStream('salida.txt');

writableStreamEvent.on('finish', function () {
    console.log('Se ha escrito el archivo');
});

writableStreamEvent.write('Hola Node.js!\n');
writableStreamEvent.end();
```

Node.js: Streams - Encriptar

```
const crypto = require('crypto');
const fs = require('fs');
const zlib = require('zlib');

const password = new Buffer(process.env.PASS || 'password');
const encryptStream = crypto.createCipher('aes-256-cbc', password);
const gzip = zlib.createGzip();

const readStream = fs.createReadStream('small.txt');
const writeStream = fs.createWriteStream('encriptado.gz');

readStream // Lee el archivo
    .pipe(encryptStream) // encripta
    .pipe(gzip)  // comprime
    .pipe(writeStream)  // escribe la salida
    .on('finish', function () {  // imprimimos OK
        console.log('OK');
});
```

Node.js: Streams - Desencriptar

```
const crypto = require('crypto');
const fs = require('fs');
const zlib = require('zlib');

const password = new Buffer(process.env.PASS || 'password');
const decryptStream = crypto.createDecipher('aes-256-cbc', password);
const gzip = zlib.createGunzip();

const readStream = fs.createReadStream('encriptado.gz');

readStream // Lee el archivo
    .pipe(gzip) // encripta
    .pipe(decryptStream)  // comprime
    .pipe(process.stdout)  // escribe la salida en el terminal
    .on('finish', function () {  // imprimimos OK
        console.log('OK');
});
```

Node.js: Manejo de errores

- El manejo de errores es importante porque sino se manejan:
  - La aplicación web se puede caer.
  - La aplicación web puede quedar en estado inconsistente.
- Convención: "error-first" callback
  - El primer argumento de una función callback debe ser un objeto error.

```
fs.readFile('/foo.txt', function(err, data) {
    // Hacer algo
});
```

- Hay dos posibles escenarios:
  - Si el error es nulo, entonces la operación se ejecutó correctamente.
  - Si el error es un set, entonces ocurrió un error y debe ser manejado.

Node.js: Manejo de errores

- Cuando se utiliza EventEmitter se debe tener cuidado al emitir errores
  - Un error no manejado apropiadamente puede quebrar la aplicación.
  - Dependiendo de la aplicación puede ser fatal o sólo un mensaje error.

```
const EventEmitter = require('events').EventEmitter;
const emitter = new EventEmitter();

emitter.emit('error', new Error('Esto es un error'));

emitter.on('error', function(err) {
    console.error('Algo muy malo paso!:', err.message);
});
```

Node.js: Manejo de errores

- Un error es una instancia de la clase Error. Cuando un error se lanza es una excepción.
  - callback(new Error('Algo malo paso'));
  - throw new Error('Algo malo paso');
- Hay muchas casos en que queramos delegar el error a un callback y no lidiar con el error.
- El módulo **verror** permite empacar un error y proporcionar un mensaje descriptivo.
- Cómo manejar errores, detalles acá: https://www.joyent.com/node-js/production/design/errors

Node.js: Manejo de errores

```
// npm install verror
const verror = require('verror');

function readFiles(archivos, callback) {
    for(let archivo of archivos) {
        return callback(new verror.VError('Fallo al leer el archivo %s', archivo));
    }
}

readFiles(['no-existo'], function(err, contenido) {
    if (err) { throw err; }
    console.log(contenido);
});
```

Debugging en Node.js

- Para pequeños bugs siempre se puede usar console.log.
- Que pasa cuando hay situaciones complejas: node-inspector
  - Se puede agregar puntos de quiebre.
  - Se puede ir paso a paso.
  - Inspecciona el alcance, variables, propiedades de objetos.
  - También permite editar valores de variables y propiedades de objetos.
- Instalación:
  - $ npm install -g node-inspector

Debugging en Node.js

```
const http = require('http');
const port = process.env.PORT || 1337;

http.createServer(function(req, res) {
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.end(new Date() + '\n');
}).listen(port);

console.log('Servidor corriendo en el puerto %s', port);
```

- Ejecutar:
  - $ node-debug simple_server.js

Node.js y Express

- Express es el framework web más popular de Node.
- Proporciona un conjunto de características para desarrollar aplicaciones web y móviles.
- Permite configurar el middleware para responder a solicitudes HTTP.
- Define una tabla de rutas, la cual se utiliza para ejecutar diferentes acciones basada en los métodos HTTP.
- Permite renderizar dinámicamente páginas HTML basada en plantillas.

Node.js y Express

- Instalar Express:
  - $ npm install express --save
- Adicionalmente instalar:
  - body-parser: Maneja JSON, Raw, Texto y URL.
    $ npm install body-parser --save
  - cookie-parser: Maneja cookies
    $ npm install cookie-parser --save
  - multer: Maneja multipart/form-data
    $ npm install multer --save
- Ahora, creemos una aplicación simple.

Express: Métodos HTTP

```
app.get(urlPath, requestProcessFunction);
app.post(urlPath, requestProcessFunction);
app.put(urlPath, requestProcessFunction);
app.delete(urlPath, requestProcessFunction);
app.all(urlPath, requestProcessFunction);
```

- urlPath puede contener parámetros como ngRoute de Angular. Ejemplo: /user/:user_id

Express: Un servidor simple

```
const express = require('express');
const app = express();

app.get('/', function (req, res) {
    res.send('Hola Express');
})

const server = app.listen(8080, function () {
    const host = server.address().address
    const port = server.address().port

    console.log("Ejemplo de aplicacion en http://%s:%s", host, port)
});
```


Express: Servidor simple para archivos estáticos

```
const express = require('express');
const app = express();

app.use(express.static('public')); // Debe existir esta carpeta en el directorio

app.get('/', function (req, res) {
   res.send('Hola Express');
})

const server = app.listen(8080, function () {
   const host = server.address().address
   const port = server.address().port

   console.log("Ejemplo de aplicacion en http://%s:%s", host, port)
});
```

Express: GET

```
<html>
   <body>
      <form action="http://127.0.0.1:8080/process_get" method="GET">
         Nombre: <input type="text" name="nombre">
         <br>
         Apellido: <input type="text" name="apellido">
         <input type="submit" value="Submit">
      </form>
   </body>
</html>

// Abrir un browser: http://127.0.0.1:8080/index_get.html
```

Express: GET

```
const express = require('express');
const app = express();

// Debe existir esta carpeta en el directorio
//app.use(express.static('public'));

app.get('/index_get.html', function (req, res) {
   res.sendFile(__dirname + "/" + "index_get.html");
});
```

Express: GET

```
app.get('/process_get', function (req, res) {
   // Salida en formato JSON
   respuesta = {
      nombre: req.query.nombre, apellido: req.query.apellido
   };
   console.log(respuesta);
   res.end(JSON.stringify(respuesta));
});

const server = app.listen(8080, function () {
    const host = server.address().address
    const port = server.address().port
    console.log("Ejemplo de aplicacion en http://%s:%s", host, port)
});
```

Express: POST

```
<html>
   <body>
      <form action="http://127.0.0.1:8080/process_post" method="POST">
         Nombre: <input type="text" name="nombre">
         <br>
         Apellido: <input type="text" name="apellido">
         <input type="submit" value="Submit">
      </form>
   </body>
</html>

// Abrir un browser: http://127.0.0.1:8080/index_post.html
```

Express: POST

```
const express = require('express');
const app = express();
const bodyParser = require('body-parser');

const urlencodedParser = bodyParser.urlencoded({ extended: false });

// Debe existir esta carpeta en el directorio
//app.use(express.static('public'));

app.get('/index_post.html', function (req, res) {
   res.sendFile(__dirname + "/" + "index_post.html");
});
```

Express: POST

```
app.post('/process_post', urlencodedParser, function (req, res) {
   // Salida en formato JSON
   respuesta = {
      nombre: req.body.nombre, apellido: req.body.apellido
   };
   console.log(respuesta);
   res.end(JSON.stringify(respuesta));
});

const server = app.listen(8080, function () {
    const host = server.address().address
    const port = server.address().port
    console.log("Ejemplo de aplicacion en http://%s:%s", host, port)
});
```


Express: Manejando Cookies

```
const express = require('express');
const cookieParser = require('cookie-parser');
const app = express();

app.use(cookieParser());

app.get('/', function (req, res) {
    // Descomentar para asignar la cookie, luego comentar para ver. Solo dura un minuto!!
    //res.cookie('remember', 1, { maxAge: 60 * 1000 });
    console.log("Cookies: ", req.cookies);
    res.send('Hola Cookies');
})

app.listen(8080);
```

¿Preguntas?