



DECODED

CODE

FULL STACK DEVELOPER BOOTCAMP

Desarrollamos
{ talento }

NODE JS

Introducción a NODEJS

Es un entorno de ejecución multi-plataforma completamente escalable
creado por Ryan Lienhart Dahl

Introducción a NODEJS

Su primera versión se publicó el 27 de mayo de 2009 con licencia MIT, desarrollado en C/C++, JavaScript y basado en el motor [JavaScriptV8](#), actualmente es la plataforma para desarrollo de diferentes proyectos muy conocidos.

- [Phonegap/Cordova](#)
- [Ionic](#)
- [Bower](#)
- [Grunt/Gulp](#)
- [WebPack](#)

Introducción a NODEJS

Actualmente estamos en la versión 10.15.0 LTS y es uno de los servidores con mayor rendimiento en producción.

Introducción a NODEJS

Si tenéis un poco de tiempo podéis ver un [vídeo donde el creador de NODE explica porque lo creo y porque selecciono dichas tecnologías.](#)

Introducción a NODEJS

Algunas de características:

- Rápido
- Escalable
- Manejo de Eventos.
- Asíncrono
- Ligero
- Eficiente.
- Funciona en sistemas distribuidos.
- Curva de aprendizaje corta.
- Tiene una comunidad muy activa.

Rendimiento

Node ha sido diseñado para optimizar el rendimiento y la escalabilidad en aplicaciones web y es un muy buen complemento para muchos problemas comunes de desarrollo web (ej, aplicaciones web en tiempo real).

Javascript

JavaScript es un lenguaje de programación relativamente nuevo y se beneficia de los avances en diseño de lenguajes cuando se compara con otros lenguajes de servidor web tradicionales (ej, Python, PHP, etc.) Muchos otros lenguajes nuevos y populares se compilan/convierten a JavaScript de manera que puedes también usar CoffeeScript, ClosureScript, Scala, LiveScript, etc.

NPM

El gestor de paquetes de Node (NPM del inglés: Node Packet Manager) proporciona acceso a miles de paquetes reutilizables. Además tiene uno de los mejores sistemas de gestión de dependencias y puede usarse para automatizar la mayor parte de la cadena de herramientas de compilación.

Multiplataforma

Es portable, con versiones que funcionan en Microsoft Windows, OS X, Linux, Solaris, FreeBSD, OpenBSD, WebOS, y NonStop OS. Además, está bien soportado por muchos de los proveedores de hospedaje web, que proporcionan infraestructura específica y documentación para hospedaje de sitios Node. Tiene un ecosistema y comunidad de desarrolladores de terceros muy activa, con una gran cantidad de gente deseando ayudar.

USO

Ejecutar javascript en NODE.JS

```
$ node ruta/al/nombreDelArchivo.js
```

Acceso a los parámetros de ejecución

```
$ node nombreDelArchivo.js parametro1 parametro2 etc  
// nombreDelArchivo.js  
let parametrosDeEjecucion = process.argv.slice(2); // ['parametro1',  
'parametro2', 'etc']
```

Basado en módulos

```
// doble.js
// exportación:
module.exports = function(a) {
    return 2 * a;
};

// importación:
var doblaNumero = require('./doble.js');
dobraNumero(2); // 4
```

Instalación

Instalando Node.js

Para instalar **NODE.JS** en caso de windows o MAC podemos bajarnos su instalador desde su página web.

En el caso de linux podemos realizar la instalación desde el terminal

```
$ curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
$ sudo apt-get install -y nodejs
$ sudo apt-get install -y build-essential //herramientas de compilación.
```

The background of the slide features a light gray polygonal grid pattern. Overlaid on this are several large, semi-transparent orange and red triangles. One triangle is positioned in the upper left quadrant, another is at the bottom left, and a smaller one is near the center. A single horizontal orange line segment is located in the lower right quadrant.

npm

Node Package Manager (npm)

Que es NPM

Npm es el gestor de paquetes por excelencia de Node.js, los dos han ido acompañados de la mano desde que apareció node.js ya en la versión 0.06

Es un gestor que utiliza el bash para realizar las tareas de instalación de las dependencias

Actualmente dispone de más de 700.000 paquetes instalables para ser utilizados en nuestras aplicaciones.

Que tareas podemos realizar con NPM

- Gestión de dependencias.
- Ejecución de comandos asociados a una tarea
- Manejo de versiones de las dependencias.
- A la vez podemos crear nuestra distribución de paquetes.

Instalando NPM

Viene por defecto en la instalación de NODEJS.

Comprobación de la instalación

Desde un terminal ejecutamos la siguiente línea de comandos

```
npm -v
```

Empezando con NPM

Podemos comprobar todos los paquetes que dispone NPM en su [web](#)

La misma página nos da toda la información necesaria sobre los paquetes disponibles por ejemplo [BootStrap](#)

Instalando dependencias con NPM

Para utilizar npm abrimos una consola de nuestro SO y ejecutamos.

```
$ npm install [nombre del paquete]
```

Instalando dependencias con NPM

Inicialmente no es necesario disponer de tener el archivo package.json , ya que las instalaciones podemos realizarlas desde:

- Un directorio que contiene el package.json
- Mediante una URL que contenga el paquete.
- Mediante el nombre del paquete y la versión a instalar
paquete@version
- Mediante la url git del paquete.
- Mediante el nombre del paquete y un tag **paquete@tag** O simplemente con el nombre del paquete y nos instalará la última versión.

Creando el package.json

Una manera de trabajar es utilizar el package.json donde almacenamos las dependencias que vamos instalando.

Para ello ejecutamos el comando

```
$ npm init
```

Creando el package.json

El mismo nos creará el archivo **package.json** una vez contestada las preguntas que nos marca el shell

```
{  
  "name": "slides-bootcamp-fullstack",  
  "version": "1.0.0",  
  "description": "",  
  "main": "",  
  "scripts": {  
    "start": "reveal-md -w",  
    "pdf": "reveal-md content/git/git.md --print content/git/git.pdf",  
    "test": "echo \"Error: no test specified\" && exit 1",  
    "postinstall": "# npm install -g reveal-md"  
  },  
  "author": "",  
  "license": "ISC",  
  "dependencies": {  
    "reveal-md": "^2.0.0"  
  },  
  "devDependencies": {  
    "live-server": "^2.0.0"  
  }  
}
```

Creando el package.json

Para realizar la instalación y que lo almacene en nuestro package.json debemos utilizar

```
$ npm install [nombre del paquete] --save
```

Si queremos guardarlo en el entorno de desarrollo debemos utilizar la siguiente instrucción

```
$ npm install [nombre del paquete] --save-dev
```

Creando el package.json

Ciertos paquetes son necesarios instalarlos de forma global en nuestro SO, para ello ejecutaremos:

```
$ npm install [nombre del paquete] -g
```

Otras instrucciones interesantes de NPM

Podemos listar los paquetes que tenemos instalados en nuestra aplicación mediante:

```
//Si queremos obtener todo el listado.  
$ npm ls  
// Si queremos obtener el listado detallado.  
$ npm ls -l  
// Si queremos obtener el listado de paquetes instalados de forma global.  
$ npm ls -g
```

Otras instrucciones interesantes de NPM

Para desinstalar paquetes instalados usamos

```
$ npm uninstall [nombre del paquete ]
```

Para actualizar

```
// Para actualizar todos los paquetes instalados.  
$ npm update  
// Para actualizar un paquete determinado.  
$ npm update [nombre del paquete]
```

Tareas / Scripts

Otras de las posibilidades que nos brinda NPM es la definición de tareas o script que podemos lanzar desde la ventana shell

Por defecto tenemos las siguientes tareas predefinidas

```
"scripts": {  
  "test": "echo \\\"Error: no test specified\\\" && exit 1"  
}
```

Tareas / Scripts

Para definir nuevas tareas o script debemos modificar el archivo package.json y añadirlo en el apartado scripts

```
// dentro de package.json
"scripts": {
  "produccion": "node app.js"
}
//Para ejecutarlo desde la terminal
$ npm run produccion
```

Node environment

Core modules

- Documentación oficial
- Listado core modules

Assertion Testing

C/C++ Addons - N-API

Crypto

Events

HTTPS

Query Strings

String Decoder

TTY

V8

Buffer

Child Processes

DNS

File System

Net

REPL

Timers

URL

VM

C++ Addons

Cluster

Domain

HTTP

Path

Stream

TLS/SSL

Utilities

ZLIB

Module scope

- Documentación

`_dirname`
`clearInterval()`
`exports`
`process`
`setInterval()`
`URLSearchParams`

`_filename`
`clearTimeout()`
`global`
`require()`
`setTimeout()`

`clearImmediate()`
`console`
`module`
`setImmediate()`
`URL`

Primeros pasos con NODE.JS

Creando nuestro primer servidor.

Ejecutamos un nuevo proyecto mediante **npm init**

Creamos un .js con nuestra aplicación y le insertaremos el siguiente código:

```
//Cargarmos las librerías necesaria del core de node.js
const http = require('http');

//Definimos variables a usar.
const hostname = '127.0.0.1';
const port = 3000;
```

Creando nuestro primer servidor.

```
//Creamos el objeto servidor donde pasamos los dos parámetros.  
const server = http.createServer((req, res) => {  
    res.statusCode = 200;  
    res.setHeader('Content-Type', 'text/plain');  
    res.end('Hello World\n');  
});
```

Creando nuestro primer servidor

```
//Activamos nuestro servidor.  
server.listen(port, hostname, () => {  
  console.log(`Servidor arrancado en http://${hostname}:${port}/`);  
}) ;
```

Creando nuestro primer servidor.

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

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
}) ;

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
}) ;
```

Gestionando Rutas directamente desde NODE.JS

Para ello debemos utilizar un nuevo módulo del core de node.js `url`

```
const http = require('http');
const url = require('url');
```

Gestionando Rutas directamente desde NODE.JS

Obtenemos el path name de la petición mediante el módulo url

```
const server = http.createServer((req, res) => {  
  let pathname = url.parse(req.url).pathname;
```

Gestionando Rutas directamente desde NODE.JS

Tratamos la ruta obtenida

```
if (pathName === '/') {  
  console.log(pathName);  
  res.statusCode = 200;  
  res.setHeader('Content-Type', 'text/plain');  
  res.end('Hello World\n');  
}
```

Gestionando Rutas directamente desde NODE.JS

```
const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
    let pathName = url.parse(req.url).pathname;
    if(pathName === '/') {
        res.statusCode = 200;
        res.setHeader('Content-Type', 'text/plain');
        res.end('Hello World\n');
    }
    else if (pathName === '/About') {
        console.log(pathName);
        res.statusCode = 200;
        res.setHeader('Content-Type', 'text/plain');
        res.end('Quienes somos\n');
    }
}) ;
}) ;

server.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname}:${port}/`);
}) ;
```

Multi Hilo con Workers

The background of the slide features a light gray polygonal grid pattern. Overlaid on this are several large, semi-transparent orange triangles. One triangle is positioned in the upper left quadrant, pointing downwards. Another is at the bottom left, pointing upwards. A third is located in the lower right quadrant. A single, thin red horizontal line segment is placed near the center-right of the slide.

Requisitos para su funcionamiento

Necesitamos tener como mínimo la versión de node 10.5, en esta versión se ha añadido en versión experimental el procesamiento asíncrono mediante Workers.

```
node -v; // >= 10.5
```

¿Para qué sirven?

Hasta ahora Node ha tenido una desventaja a la hora del procesamiento de grandes cantidades de datos u operaciones computacionales que tengan una gran carga de procesador, los Workers vienen para darnos una solución a este problema, aunque aún en modo experimental, para las áreas como AI, Machine Learning, Data Science etc.

Iniciando Node con Workers en modo experimental

Para ello vamos a crear un script en el que tenga por defecto el parámetro de Workers experimentales activado.

```
"start": "node --experimental-worker app.js"
```

Importando el paquete para trabajar con workers

Tenemos que importar al inicio de nuestro archivo .js el paquete de 'worker_threads', del cual vamos a coger los elementos que nos interesan.

```
const worker = require('worker_threads');  
const { Worker, isMainThread, workerData } = worker;
```

Y creamos unas variables que utilizaremos más adelante.

```
let valorActual = 0;  
let intervalos = [100, 1000, 500];
```

¿Qué son Worker, isMainThread y workerData?

- Worker: es un objeto del cual podemos crear instancias, las cuales de ejecutan en un hilo de procesamiento diferente al actual.
- isMainThread: es un Boolean que nos indica si estamos en el hilo de ejecución desde el que hemos arrancado mediante Node.
- workerData: es una clonación de otro objeto para crear comunicación entre workers, ya que al pasar un objeto por parámetros en JavaScript se hace por referencia, cosa que se evita con este método.

Creando código para el hilo principal

Utilizando la variable `isMainThread`, la cual nos indica si estamos en el hilo principal de procesamiento vamos a determinar el comportamiento de cada una de las partes dentro de un "if"



```
if (isMainThread) {  
    // Unicamente en el hilo principal  
    console.log('Este es el hilo principal');  
    for (let i = 0; i < 2; i++) {  
        // iteramos dos veces  
        // creamos un worker pasando como datos el valor del iterador  
        let w = new Worker(__filename, { workerData: i });  
    }  
  
    setInterval(  
        a => {  
            // Aumentamos el contador cada  
            valorActual = contador(a, valorActual + 1);  
            return valorActual;  
        },  
        intervalos[2],  
        'MainThread'  
    );  
}
```


Creando código para el Worker

Si no estamos en el hilo principal vamos a coger el ID del iterador a través de workerData para determinar el intervalo de tiempo entre iteraciones que le corresponde y vamos a aumentar su valor. Tendremos como resultado en la consola una serie de logs en los que imprimimos el contador, si es el mainThread o en caso de ser un Worker, imprimiremos su ID a través de la función "contador".

```
else {  
    console.log('Este no lo es');  
  
    setInterval((a) => {  
        valorActual = contador(a, valorActual + 1)  
        return valorActual;  
    }, intervalos[workerData], workerData);  
}
```

Simulando un caso de gran uso computacional

En este caso vamos a crear comunicación entre el hilo principal y el Worker, además de crear archivos diferentes para cada uno de ellos, en el archivo app.js necesitaremos las siguientes librerías:

```
// app.js
const { Worker } = require('worker_threads');
const request = require('request');

// Además vamos a poner un console log para reconocer de dónde vienen los
// datos.
console.log('Este es el hilo principal');
```

Comunicación entre hilos

Para poder comunicarnos entre hilos de procesamiento necesitamos atrapar ciertos eventos, por suerte, los propios Workers disponen del método `.on(nombreEvento, callback)` desde el que podemos atraparlos sin necesidad de código extra.

Además vamos a aislar la creación de nuevos workers en una función reutilizable.

```
// app.js
function empezarWorker(path, callback) {
  let w = new Worker(path, { workerData: null });
  w.on('message', msg => {
    // Mensaje desde el worker
    callback(null, msg);
  });
  w.on('error', callback); // Explicacion del error
  w.on('exit', code => {
    // Codigo de salida del worker diferente a 0 significa que ha habido
    un error
    if (code != 0)
      console.error(
        new Error(`El Worker se ha parado, código de error: ${code}`));
  });
}
```

});

return w;

}

Haciendo peticiones desde el mainThread

Para comprobar que tenemos acceso a peticiones a servidores desde el archivo donde creamos los Workers, imprimiremos por consola el resultado de una petición get a google:

```
// app.js
request.get('http://www.google.com', (err, response) => {
  if (err) {
    return console.error(err);
  }

  // response.body[0] como array sirve para leer letra a letra
  // el contenido de la pagina web solicitada
  console.log('Total de bytes recibidos: ', response.body.length);
});
```

Intanciando un Worker

Esta vez el Worker va a ser una instancia de un archivo aislado, que nos va a imprimir sus resultados por consola.

```
// app.js
let miWorker = empezarWorker(
  `${__dirname}/codigoWorker.js`,
  (error, resultado) => {
    if (error) return console.error(error);

    console.log('[[Proceso con gran carga computacional terminado]]');
    console.log('El primer valor es: ' + resultado.val);
    console.log('Ha tardado ' + resultado.timeDiff / 1000 + ' '
segundos.');
  }
);
```

Comunicacion desde el Worker



En esta ocasión únicamente necesitamos tener la capacidad de comunicarnos con el main thread a través de "parentPort". La información devuelta será el índice 0 tras ordenar un array de 5.000.000 números aleatorios y el tiempo que ha transcurrido desde el inicio de la operación.

```
// codigoWorker.js
const { parentPort } = require('worker_threads');
const ordenar = require('./ordenar');

function random(min, max) {
  return Math.random() * (max - min) + min;
}

const start = Date.now();
let elGranArray = Array(5000000)
  .fill()
  .map(() => random(2, 10000));

parentPort.postMessage({
  val: ordenar(elGranArray).primerValor,
  timeDiff: Date.now() - start,
});
```

Creando librerías para nuestro worker

Para importar librerías hemos visto que se hace igual que en todos los archivos de Node, con el método `require()` y de esta misma manera para exportar, usaremos `module.exports`

```
// ordenar.js
module.exports = function(arr) {
  arr.sort((a, b) => {
    return a - b;
  });

  return { primerValor: arr[0] };
};
```

Ahora ya podemos ejecutar `app.js` y comprobar su funcionamiento.