

Fullstack Web Bootcamp

Desarrollo Backend con Node.js

20/10/2025

Índice

Introducción	03
Web	09
Node JS	14
JS Pro	37

Introducción





Desarrollador Fullstack, exalumno (reincidente) de KeepCoding, amante de Typescript y todo el ecosistema Javascript.
Entusiasta de la formación y divulgación en todo aquello que esté relacionado con la tecnología y el emprendimiento.

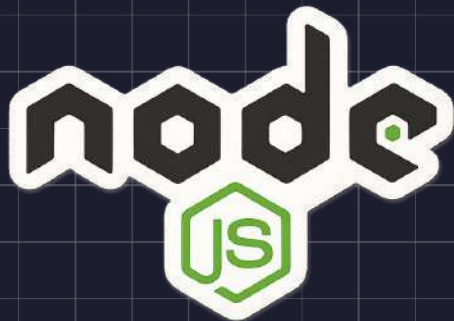
Requisitos

- Conocer VSCode
- Conocer GIT
- Comprender los conceptos del módulo de desarrollo frontend con Javascript
- Ganas de aprender

Objetivos

- Entender cómo funciona la arquitectura básica de un entorno web
- Conocer cómo se comunica nuestro navegador con el entorno
- Entender las peticiones HTTP y sus verbos
- Comprender cómo se persisten los datos
- Aplicar una política correcta de permisos
- Disfrutar por el camino

Stack y Herramientas



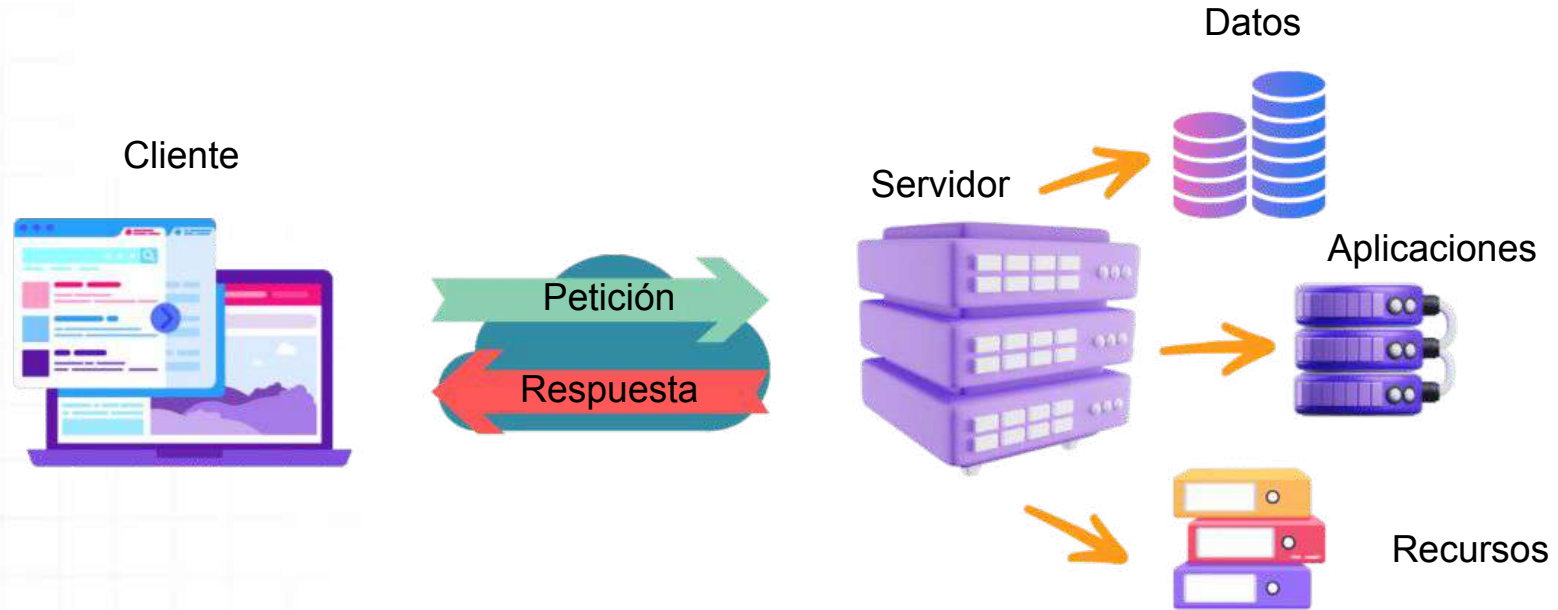
¿Qué construiremos?

Api RestFull con autenticación

Web

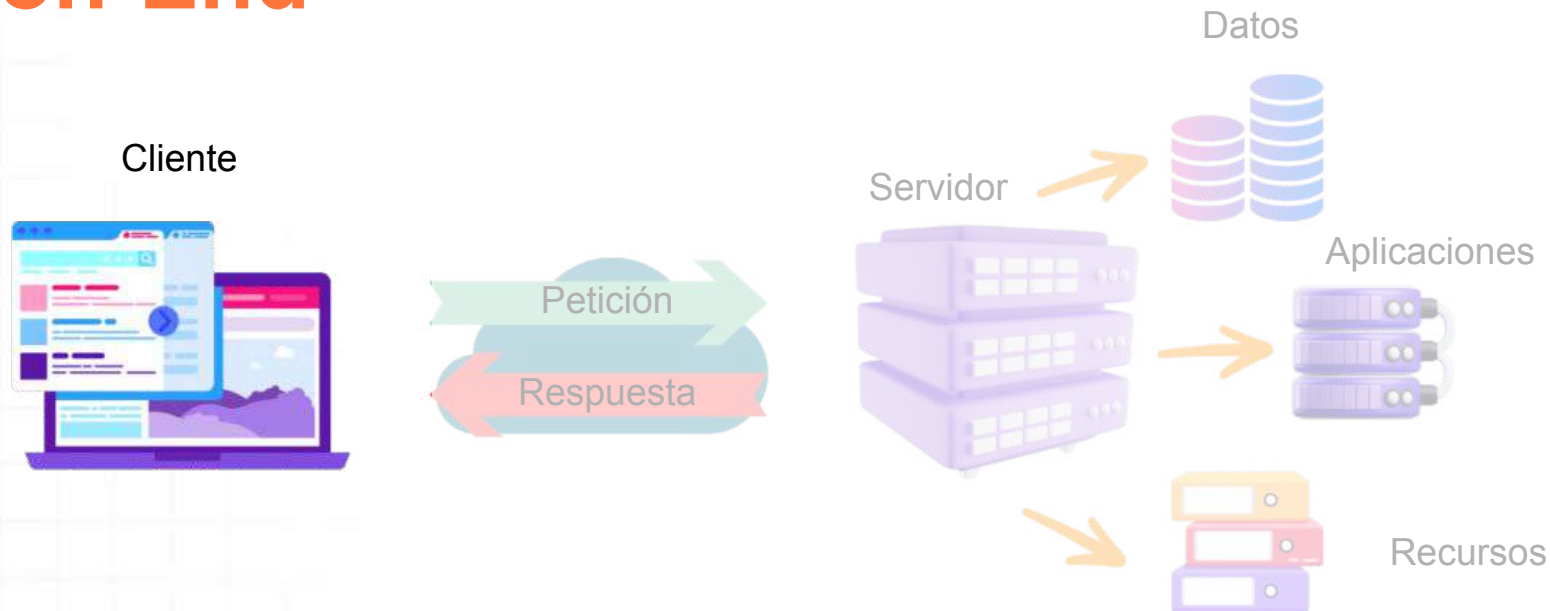


Cómo funciona la web



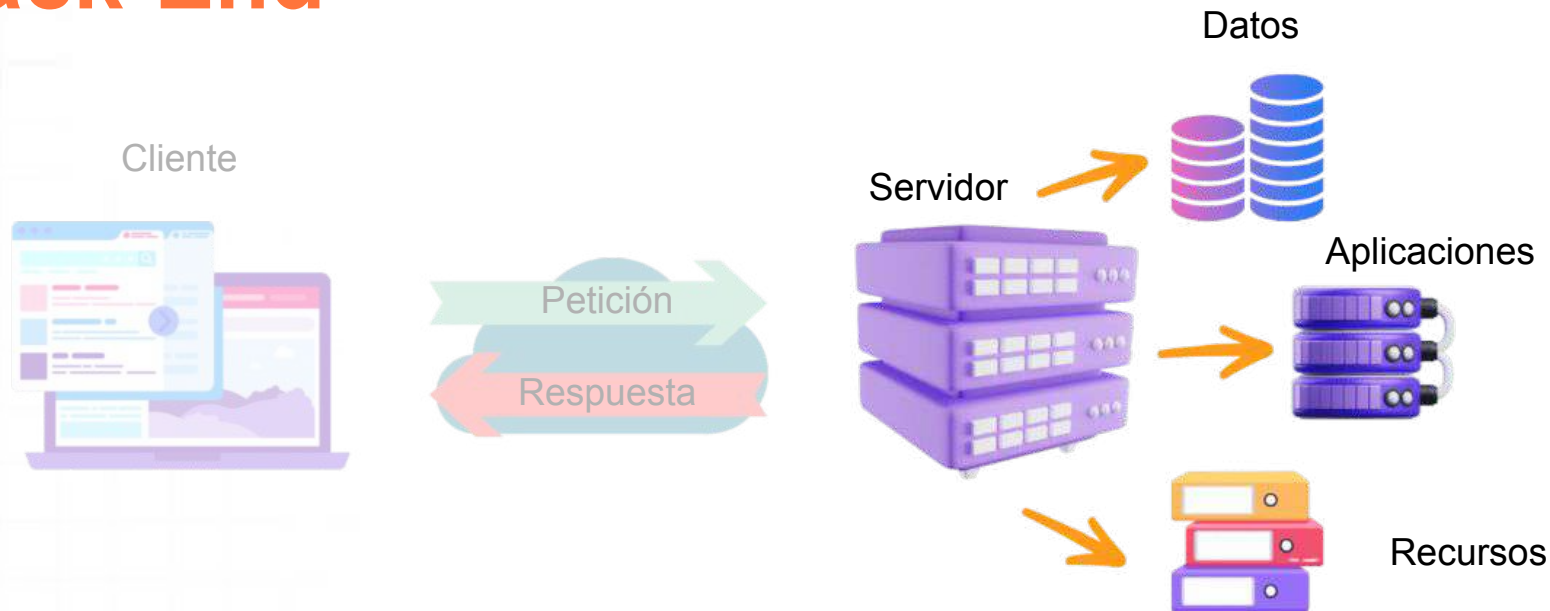
Cómo funciona la web

Fron-End



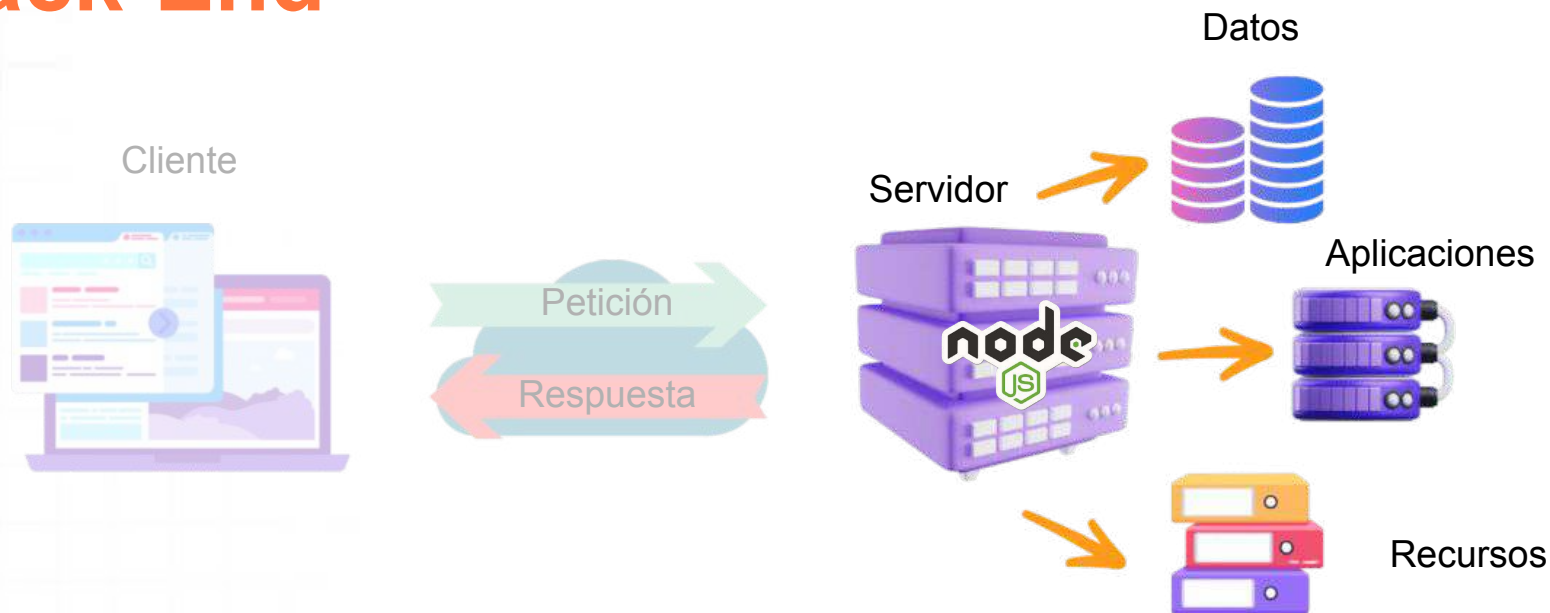
Cómo funciona la web

Back-End



Cómo funciona la web

Back-End



Intro NodeJS



NodeJS

¿Qué es?

- Es un intérprete de Javascript
- Inicialmente diseñado para servidor
- Orientado a eventos*
- Con servidor de aplicación*
- Basado en el motor V8*



NodeJS

Orientado a eventos

- En la programación secuencial, es el programador quien decide el flujo y el orden de ejecución.
- En la programación orientada a eventos, el usuario o los programas cliente son quienes deciden el flujo.

NodeJS

Servidor de aplicación

- No necesitamos un programa que ejecute nuestro código como Tomcat, IIS, etc.
- Tampoco necesitamos un servidor web como Apache, nginx, etc.
- Nuestra aplicación realiza todas las funciones de un servidor.

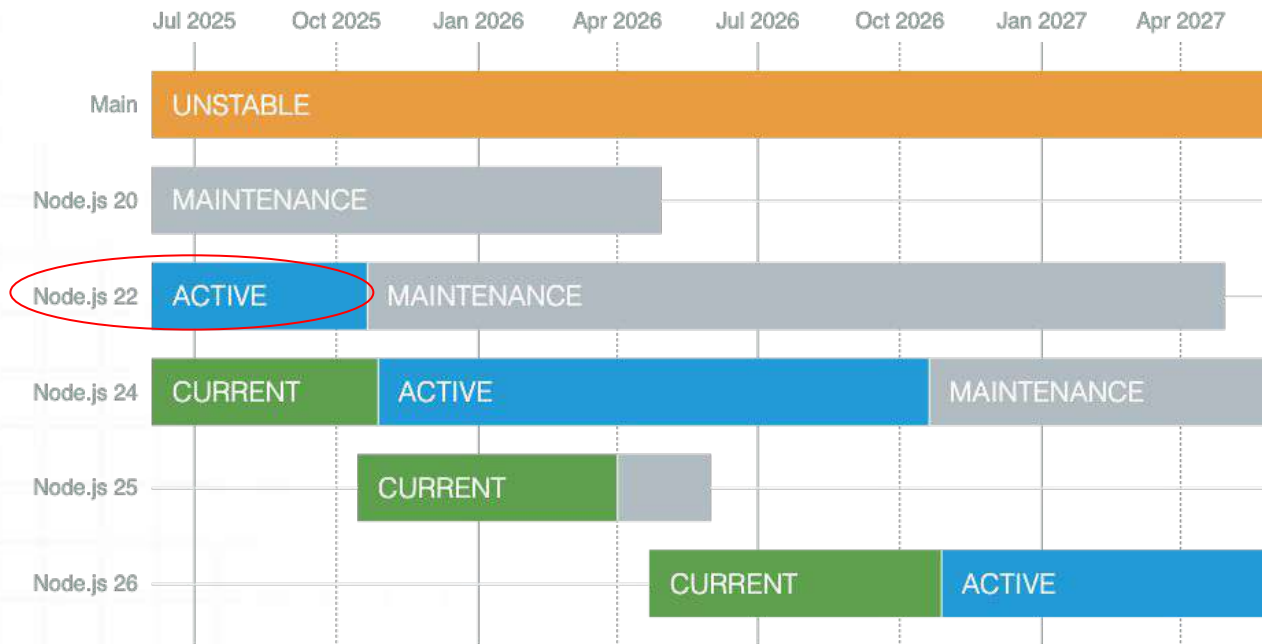
NodeJS

Motor V8

- Motor de Javascript creado por Google para Chrome
- Escrito en C++
- Multiplataforma (Windows, Linux, Mac)

NodeJS

Versiones



NodeJS

Ventajas

Node.js tiene grandes ventajas reales en:

- Aplicaciones de red, como APIs , servicios en tiempo real, servidores de comunicaciones, etc.
- Aplicaciones cuyos clientes están escritos en Javascript, pues compartimos código y estructuras entre el servidor y el cliente.

<https://node.green/>



NodeJS

Instalación

Existen distintas formas de instalar Node.js

- Desde el instalador oficial
 - Más sencillo de instalar
- Con un instalador de paquetes
 - Más fácil desinstalar y mantener actualizado
- Con un gestor de versiones
 - Podemos gestionar distintas versiones

NodeJS

Versions Managers

Un gestor de versiones como NVM nos permite tener distintas instalaciones aisladas de Node.js en nuestro dispositivo e utilizarlas de manera independiente según el proyecto.

Windows: <https://github.com/coreybutler/nvm-windows>

Linux / Mac: <https://github.com/nvm-sh/nvm>

Instalar node.js

```
$ nvm --version
```

```
$ nvm list
```

```
$ nvm install <version>
```

```
$ nvm use <version>
```

```
$ node --version
```



NodeJS

NVM - Cambio automático

<https://github.com/nvm-sh/nvm#nvmrc>

```
$ node --version > .nvmrc
```

Posteriormente, al entrar en la carpeta podemos ejecutar

```
$ nvm use
```

Y tras salir de la carpeta

```
$ nvm use default
```

Para usuarios de Mac y Linux:

<https://github.com/nvm-sh/nvm#deeper-shell-integration>

NodeJS

NVM - Alternativas

- n es una alternativa de nvm de larga data que logra lo mismo con comandos ligeramente diferentes y se instala a través de npm en lugar de un script bash.
- fnm es un administrador de versiones más reciente, que afirma ser mucho más rápido que nvm. (También usa Azure Pipelines).
- Volta es un nuevo administrador de versiones del equipo de LinkedIn que afirma una velocidad mejorada y soporte multiplataforma.
- asdf-vm es una única CLI para varios idiomas, como gvm, nvm, rbenv y pyenv (y más), todo en uno.
- nvs (Node Version Switcher) es una alternativa a nvm multiplataforma con la capacidad de integrarse con VS Code.

Node.js

Servidor Básico

Talk is cheap

Show me the code



Crear un servidor básico

```
$ node index.js
```

```
# Y si actualizamos?
```

```
$ npm i -g nodemon
```

```
$ npx nodemon
```



Node.js

Gestor de paquetes NPM

NodeJS

NPM

Node Package Manager es un gestor de paquetes que nos ayuda a gestionar las dependencias de nuestro proyecto.

Entre otras cosas nos permite:

- Instalar librerías o programas de terceros
- Eliminarlas
- Mantenerlas actualizadas

Generalmente se instala conjuntamente con Node.js de forma automática.

NodeJS

NPM - package.json

Se apoya en un fichero llamado package.json para guardar el estado de las librerías.

```
$ npm init
```

Crea este fichero.

Documentación en <https://docs.npmjs.com/cli/v11/configuring-npm/package-json>

package.json

```
{  
  "name": "myapp",  
  "version": "1.0.0",  
  "description": "Esta es la descripción",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "author": "Nauel Gómez",  
  "license": "ISC"  
}
```



NodeJS

NPM - global vs local

Instalación local, en la carpeta del proyecto

```
$ npm install <paquete>
```

Instalación global, en nuestro sistema, o en la carpeta de instalación de node.

```
$ npm install -g <paquete>
```

Si el paquete tiene ejecutables, se hará un vínculo a ellos en `/usr/local/bin`

NodeJS

NPM - npx

Npx nos permite utilizar paquetes locales o **remotos**, en caso de ser remotos se instalan temporalmente.

```
$ npx paquete
```

Si está en local se utiliza, si no se descarga (~/.npm/_npx), por ejemplo:

```
$ npx nodemon
```

Talk is cheap

Show me the code



Instalar un módulo

Instalar el módulo chance y usar su generador de nombres en el servidor básico.

<https://www.npmjs.com/package/chance?activeTab=readme>

```
[npm init]
```

```
npm install chance
```



JS Pro



JS Pro

Hoisting

JS Pro

Hoisting

Las declaraciones de variables en JS son “hoisted”. Esto significa que el intérprete va a mover al principio de su contexto (función) la declaración, manteniendo la inicialización donde estaba.

```
1  var pinto = "My Value";
2
3  function pinta() {
4      // var pinto; // Declaración hoisted
5      console.log("Pinto: " + pinto); // My Value
6      var pinto = "Local Value"; // Esto hará hoisting de pinto y será undefined
7  }
8
9  pinta();
```

JS Pro

Hoisting

Qué valores se escribirán
en la consola?

```
1  var x = 100;
2
3  var y = function() {
4      if (x === 20) {
5          var x = 30;
6      }
7      return x;
8  };
9
10 console.log(x, y());
```


JS Pro

Hoisting

Qué valores se escribirán en la consola?

```
1  var x = 100;
2
3  var y = function() {
4      var x; // -> Hoisting, ahora es undefined
5      if (x === 20) {
6          // Aquí se mantiene la inicialización
7          x = 30;
8      }
9      return x; // X es undefined
10 };
11
12 console.log(x, y());
13 // Output: 100 undefined
```

JS Pro

JSON

JS Pro

JSON

Javascript Object Notation

- Es un formato para intercambio de datos, derivado de la notación literal de objetos en Javascript.
- Se utiliza habitualmente para serializar objetos o estructuras de datos.
- Se ha popularizado mucho principalmente como alternativa a XML, por ser más ligero que éste.

JS Pro

JSON

Convirtiendo un objeto a JSON

```
1  var empleado = {  
2      nombre: "Thomas Anderson",  
3      profesion: "Developer",  
4  };  
5  
6  JSON.stringify(empleado);  
7  // Output: '{"nombre":"Thomas Anderson","profesion":"Developer"}'
```

JS Pro

JSON

Convirtiendo JSON a un objeto

```
1  var textoJSON = '{"nombre":"Thomas Anderson","profesion":"Developer"}';  
2  
3  var objeto = JSON.parse(textoJSON);  
4  // Output: Object { nombre: 'Thomas Anderson', profesion: 'Developer' }
```

JS Pro

‘use strict’

JS Pro

Modo estricto

El modo Strict habilita más avisos y hace Javascript un lenguaje un poco más coherente. El modo no estricto se suele llamar “sloppy mode”. Para habilitarlo se puede escribir al principio de un fichero:

```
'use strict';
```

También se puede habilitar solo para una función:

```
function estoyEnStrictMode() {  
    'use strict';  
    ...  
}
```

JS Pro

Modo estricto

Algunos ejemplos de los beneficios del modo estricto:

- **Las variables deben ser declaradas. En *sloppy mode*, una variable mal escrita se crearía global, en *strict* falla.**
- Reglas menos permisivas para los parámetros de funciones, por ejemplo no se pueden repetir.
- Los objetos de argumentos tienen menos propiedades (arguments.callee por ejemplo)
- **En funciones que no son métodos, *this* será undefined.**
- Asignar y borrar propiedades inmutables fallará con una excepción, en *sloppy mode* fallaba silenciosamente.
- No se pueden borrar identificadores si cualificar (delete variable; —> delete this.variable;)
- **eval() es más limpio. Las variables que se definen en el código evaluado no pasan al scope que lo rodea.**

JS Pro

Funciones

JS Pro

Funciones

Las funciones son objetos

Pero también tienen propiedades y métodos

JS Pro

Funciones - Declaración

- Requieren un nombre
- Solo a nivel de programa o directamente en el cuerpo de otra función.
- Hacen Hoisting

```
1  function suma(numero1, numero2) {  
2      return numero1 + numero2;  
3  };
```

JS Pro

Funciones - Expresión

- Cómo son expresiones, se pueden definir en cualquier sitio donde pueda ir un valor.
- No hacen hoisting, se pueden utilizar sólo después de su definición.
- Pueden tener un nombre, pero solo sería visible dentro de su cuerpo.

```
1  var suma = function(numero1, numero2) {  
2      return numero1 + numero2;  
3  };
```

JS Pro

Funciones - Métodos

- Cuando una función es una propiedad de un objeto se llama **método**.

```
1 var calculadora = {  
2   suma: function(numero1, numero2) {  
3     return numero1 + numero2;  
4   },  
5   resta: function(numero1, numero2) {  
6     return numero1 - numero2;  
7   }  
8 };
```

JS Pro

Funciones - Instancias

Cuándo se usa **new** al invocar una función se comporta como un constructor de objetos.

```
1  function Fruta() {  
2      var nombre, familia;  
3      this.getNombre = function() {  
4          return nombre;  
5      };  
6      this.setNombre = function(valor) {  
7          nombre = valor;  
8      };  
9  };
```

JS Pro

Callbacks

JS Pro

Callbacks

Un ejemplo es cuando usamos `setTimeout`, que recibe como parámetros:

- Una función con el código que queremos que ejecute tras la espera.
- El número de milisegundos que tiene que esperar para llamarla.

```
1 console.log('Empiezo');
2 setTimeout(function() {
3     console.log('He terminado');
4 }, 3000);
```


JS Pro

Callbacks

En Node.js todos los usos de I/O (entrada / salida) deberían ser asíncronos.

Si tras una llamada a una función asíncrona queremos hacer algo, como comprobar su resultado o si hubo errores, le pasaremos **un argumento más**, una expresión de tipo función (callback), para que la invoque cuando termine.

Talk is cheap

Show me the code



Ejercicio

Hacer una función que reciba un texto y tras dos segundos lo escriba en la consola.

La llamaremos `escribeTras2Segundos`



Ejercicio

Llamarla dos veces (texto1 y texto2. Deben salir los textos con sus pausas correspondientes.

Al final escribir en la consola “Fin”.

Llamada 1 - 2sec. - **texto1** - llamada 2 - 2secs. - **texto2** - **Fin**



Ejercicio TODO

Repitamos la llamada a `escribeTras2Segundos` varias veces ejecutar un total de N veces?



JS Pro

Truthy and Falsy

JS Pro

Truthy & Falsy

En javascript todo tiene un valor booleano, generalmente conocido como **truthy** o **falsy**.

JS Pro

Truthy & Falsy

```
1  var variable = 'value';
2
3  if (variable) {
4      console.log('Soy truthy');
5  }
6
7  variable = 0;
8
9  if (variable) {
10     console.log('...');
11 } else {
12     console.log('Soy falsy');
13 }
```


JS Pro

Truthy & Falsy

Los siguientes valores siempre son **falsy**:

- `false`
- `0` (cero)
- `""` (cadena vacía).
- `null`
- `undefined`
- `NaN`

Todos los demás valores son **truthy**, incluyendo `"0"` (cero entre comillas) y `"false"` (false entre comillas), **empty functions**, **empty arrays** y **empty objects**.

JS Pro

Truthy & Falsy

```
1 // Los valores false, 0 (cero), y "" (cadena vacía) son equivalentes:
2 var c = (false == 0); // true
3 var d = (false == ""); // true
4 var e = (0 == ""); // true
5
6 // Los valores null y undefined no son equivalentes con nada, excepto con ellos mismos:
7 var f = (null == false); // false
8 var g = (null == null); // true
9 var h = (undefined == undefined); // true
10 var i = (undefined == null); // true
11
12 // Y por último, el valor NaN no es equivalente con nada, ni siquiera consigo mismo:
13 var j = (NaN == null); // false
14 var k = (NaN == NaN); // false
```

JS Pro

Truthy & Falsy - se complica

```
1  if ( [] ) { /*se ejecuta*/ }
2  /*Array es instancia de Object, y existe*/
3
4  if ( [] == true ) { /*no se ejecuta*/ }
5  /*comparamos valores!, [].toString -> "" -> falsy*/
6
7  if ( "0" == 0 ) // true (se convierten a números)
8
9  if ( "0" ) {console.log('si')} // "si" (se evalúa el string)
```

JS Pro

Truthy & Falsy - solución?

Usamos el **igual estricto** (===) y el **distinto estricto** (!==)

```
1  var melio = ( false == 0 ); // true
2
3  var seguro = ( false === 0 ); // false
4  // Primero compara el tipo y luego el valor
```

JS Pro

This

JS Pro

This

La palabra clave `this` trata de representar al **objeto que llama** a nuestra función cómo método, no donde está definida.

Por lo general, su valor hace referencia al **objeto propietario** de la función que la está invocando, o en su defecto, al objeto donde dicha función es un método.

JS Pro

This - Cuidado

De forma general, cuando se usa en algo **distinto a un método** su valor es el contexto global (o undefined en modo estricto).

```
1 console.log(this); // window en un browser, global en node
2
3 function pinta() {
4     console.log(this); // window en un browser, global en node
5 }
```

JS Pro

This

Como manejarlo? Le asignamos this con bind

```
1  var persona = {  
2    name: 'Luis',  
3    surnames: 'Gomez',  
4    fullName: function() {  
5      console.log(this.name + ' ' + this.surnames);  
6    }  
7  }  
8  setTimeout(persona.fullName.bind(persona), 1000);
```


JS Pro

Clousures

JS Pro

Closures

Un clousure se construye con una función (A) que devuelve otra (B).

La función devuelta (B), sigue manteniendo el acceso a todas las variables de la función que la creó (A).

JS Pro

Closures - Ejemplo

```
1  function creaClosure(valor) {  
2      return function(){  
3          return valor;  
4      }  
5  }
```

JS Pro

Classes

JS Pro

Classes

```
1  class Mascota {  
2      constructor(nombre) {  
3          this.nombre = nombre;  
4      }  
5      saluda() {  
6          console.log(`Hola soy ${this.nombre}`);  
7      }  
8  }
```

JS Pro

Classes

```
1 let perro = new Perro('Niebla');  
2 perro.saluda();
```

JS Pro Classes

```
1  class Perro extends Mascota {  
2      constructor(nombre) {  
3          super(nombre);  
4      }  
5  }  
6  let perro = new Perro('Niebla');  
7  perro.saluda();
```

Talk is cheap

Show me the code



JS Pro

Process

JS Pro

Process

El objeto global process tiene muchas propiedades que nos serán útiles, como process.platform, que en OS X nos dirá 'darwin', en linux 'linux', etc.

También tiene métodos útiles como process.exit(int) que para node estableciendo un exit code.

O eventos como process.on('exit', callback) donde podemos hacer cosas antes de salir.

JS Pro

Event Loop

JS Pro

Event Loop

Node usa un solo hilo.

Tiene un bucle interno, que podemos llamar 'event loop' donde cada vuelta ejecuta todo lo que tiene en esa 'fase', dejando los callbacks pendientes para otra vuelta.

La siguiente vuelta, mira a ver si ha terminado algún callback y si es así ejecuta sus handlers.

JS Pro

Event Loop - Non blocking

Si node se quedara esperando hasta que termine una query o una petición a Instagram, acumularía demasiados eventos pendientes y dejaría de atender a las siguientes peticiones, ya que **usa un solo hilo**.

Por eso todas las llamadas a funciones que usan IO, se hacen de forma asíncrona. Se aparcan para que nos avisen cuando terminen.

<https://www.builder.io/blog/visual-guide-to-nodejs-event-loop>

JS Pro

Event Loop - Events

Node nos proporciona una forma de manejar IO en forma de eventos.

Usando el EventEmitter, podemos colgar eventos de un identificador.

```
eventEmitter.on('llamar telefono', suenaTelefono);
```

JS Pro

Event Loop - Events

Y podemos emitir el identificador cuando queremos que sus eventos 'salten'.

```
eventEmitter.emit('llamar telefono');
```

JS Pro

Event Loop - Events

Son cómodos para procesar streams.

Mandando a un fichero todo lo que se escriba en la consola.

```
process.stdin.on('data', () => {...})
```


JS Pro

Módulos

JS Pro

Modulos

Los módulos de Node.js se basan en el estándar de **CommonJS**.

- Los módulos usan **exports** para exportar cosas.
- Quien quiere usar un módulo lo carga con **require**.
- La instrucción `require` es **síncrona**.
- Un módulo es un **singleton**.

JS Pro

Modulos

donde busca Node.js los módulos:

1. Si es un módulo del core (node:...=
2. Si es local (la url es relativa).
3. Módulos de la carpeta node_modules local
4. Módulos de la carpeta node_modules global

JS Pro

Modulos

Node carga un módulo una sola vez. En el primer require.

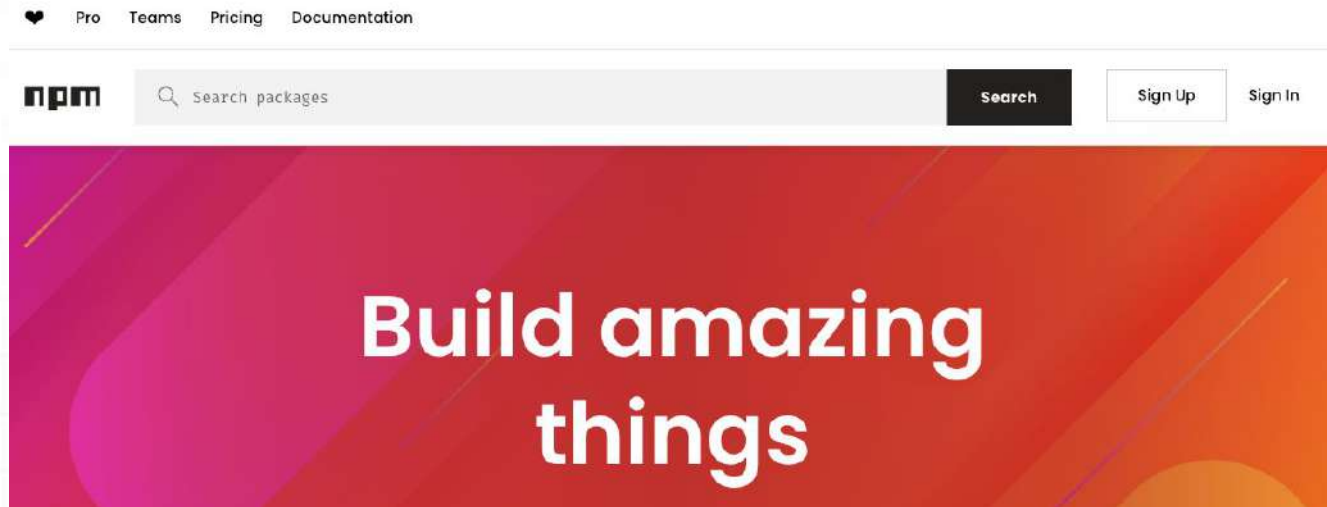
Las siguientes llamadas a require con la misma ruta devolverán el mismo objeto que se creó en la primera llamada.

Esto nos vendrá muy bien con las conexiones a bases de datos, por ejemplo.

JS Pro

Modulos

El principal repositorio de módulos es:



JS Pro

Modulos

Los módulos de **EcmaScript** están disponibles de manera estable en Node.js desde hace algunas versiones.

El proyecto debe ser de tipo módulo.

<https://nodejs.org/api/packages.html#determining-module-system>

JS Pro

Modulos - ESM

En proyectos con "type": "module", o en ficheros .mjs

- No está disponible `require` —> se usa `import`
- No está disponible `__dirname` —> puedes usar `import.meta.dirname*`
- No está disponible `__filename` —> puedes usar `import.meta.filename*`

Express.js



Talk is cheap

Show me the code



JS Pro

Express

Express es un framework web para Node.js

<https://expressjs.com/>

JS Pro

Express

Express es un framework web para Node.js

<https://expressjs.com/>

JS Pro

Express

Existen múltiples frameworks web para node.js y surgen nuevos con frecuencia, por ejemplo:

- Express.js
- Koa
- Hapi
- Restify
- ...

Muchos de ellos extienden la funcionalidad de Express, siendo el más usado.

JS Pro Express

Podemos revisar toda la documentación en:

<https://expressjs.com/en/api.html>

Gracias



ngomez@codiara.com

<https://www.linkedin.com/in/nauelg/>

keep coding



Descanso

Buscar articulo domain events



Descanso

Volvemos 21:05

