

BIENVENIDO



Express.js



mongoDB

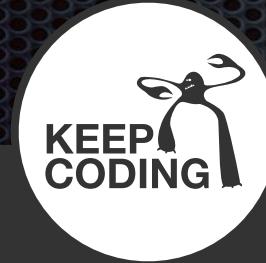


Javier Miguel

@JavierMiguelG

jamg44@gmail.com

CTO & Freelance Developer

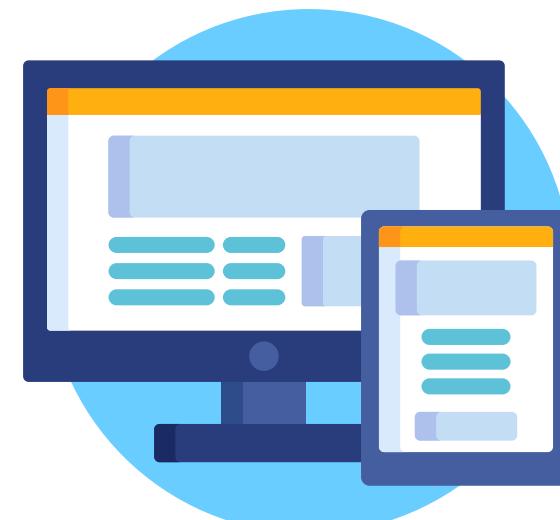


■ Introducción a la web



Cómo funciona la web

NAVEGADOR



PETICIÓN (Request)

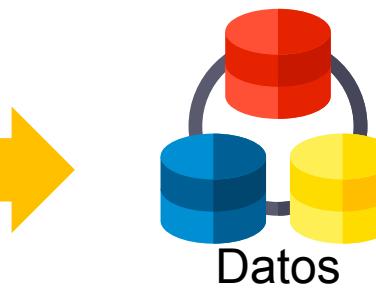
INTERNET

RESPUESTA (Response)

SERVIDOR WEB



Programas



Datos

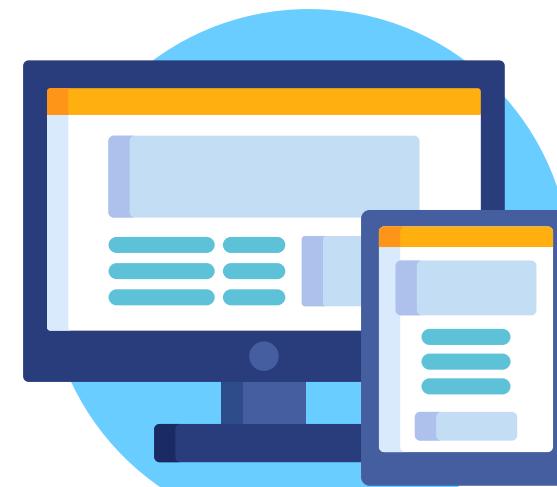


Recursos



■ Esto es FRONT-END

NAVEGADOR

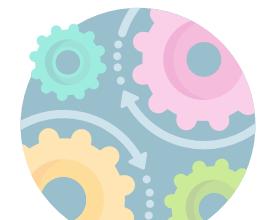


PETICIÓN (Request)

INTERNET

RESPUESTA (Response)

SERVIDOR WEB



Programas



Datos



Recursos

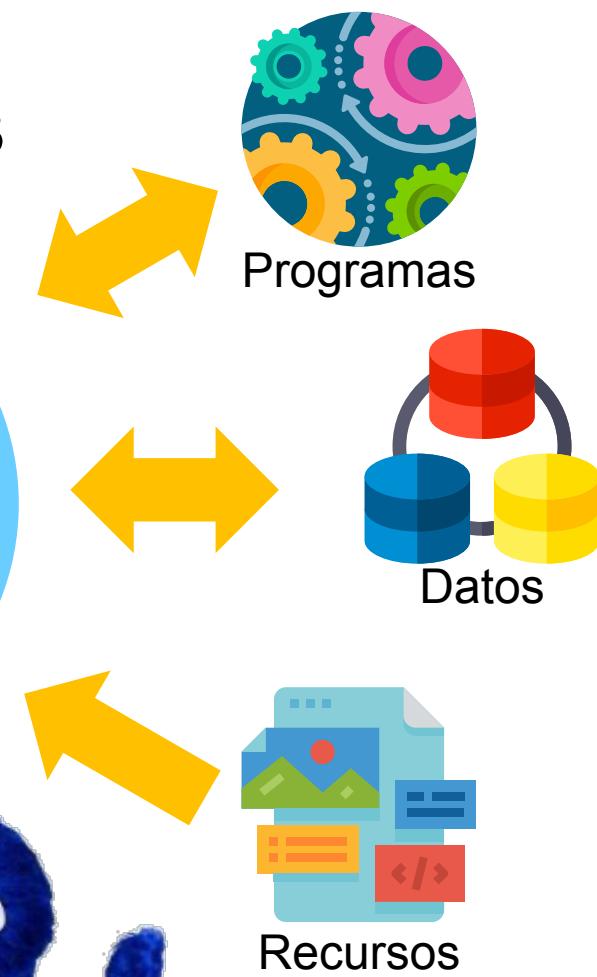


■ Esto es BACK-END

NAVEGADOR



SERVIDOR WEB



■ Esto es BACK-END

NAVEGADOR

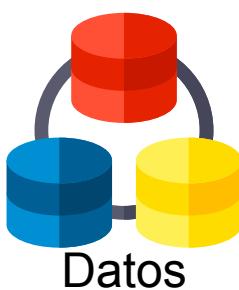


PETICIÓN (Request)

INTERNET

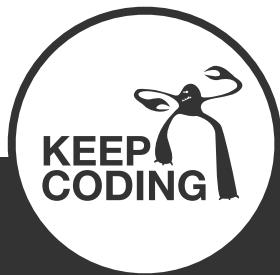
RESPUESTA (Response)

SERVIDOR WEB

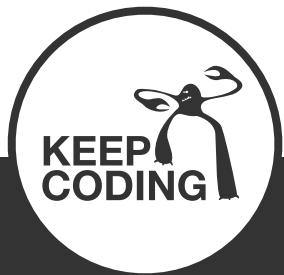




■ Introducción



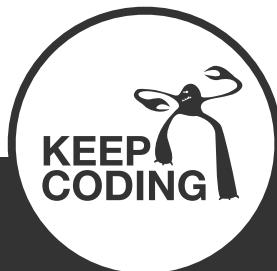
- Es un interprete de Javascript
- Inicialmente diseñado para correr en servidores





Orientado a eventos

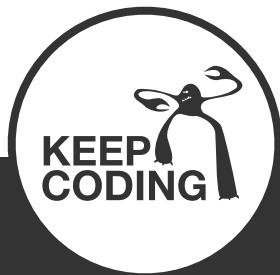
- En la programación secuencial es el programador quien decide el flujo
- En la programación orientada a eventos, el usuario o los programas clientes son quienes definen el flujo





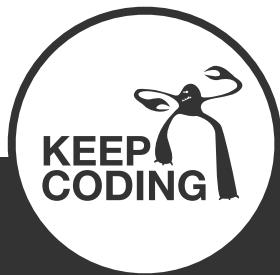
Servidores de aplicaciones

- No necesitamos Tomcat, IIS, etc
- Tampoco necesitamos un servidor web como Apache, nginx, etc
- Nuestra aplicación realiza todas las funciones de un servidor



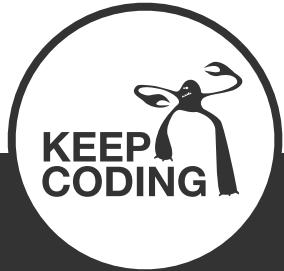
Motor V8

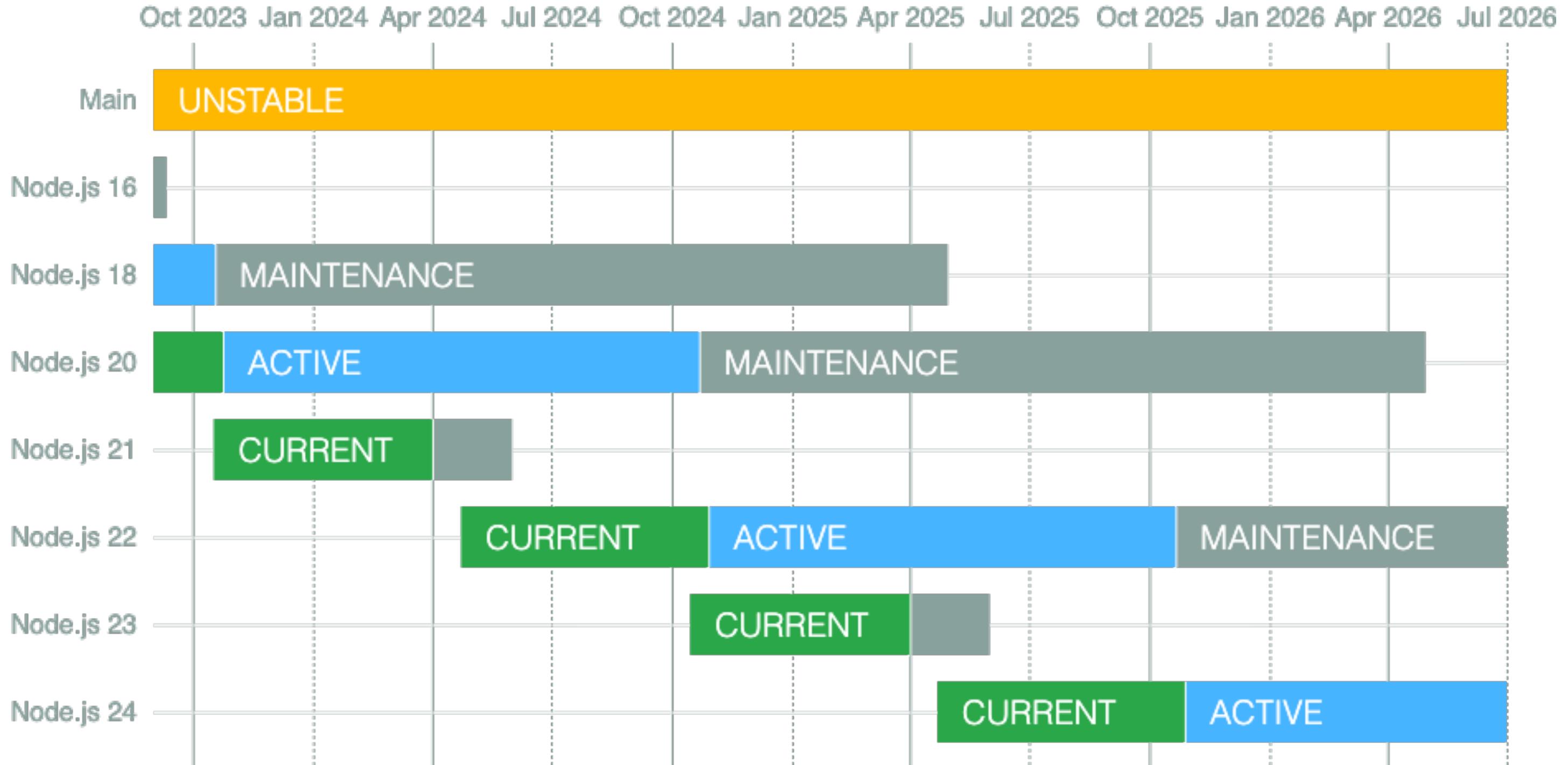
- Motor Javascript creado por Google para Chrome.
- Escrito en C++.
- Multiplataforma (Win, Linux, Mac)





Visiones





KEEP
CODING

Ventajas

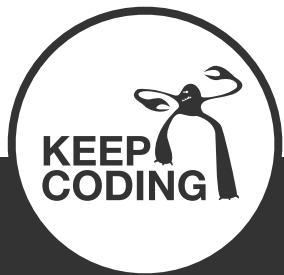
Node.js tiene grandes ventajas reales en:

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



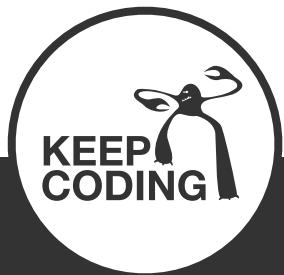


<http://node.green/>





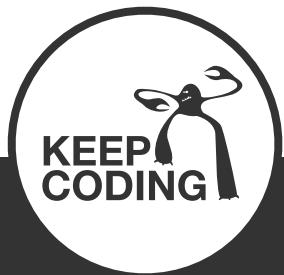
■ Instalando Node.js



■ Distintas formas

Se puede hacer:

- Desde el instalador oficial en nodejs.org
- Con un instalador de paquetes
- Compilando manualmente
- nvm



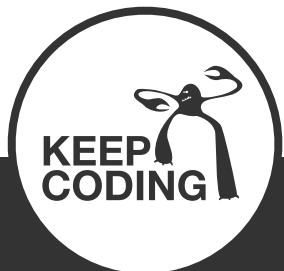
■ Distintas formas - ¿Como decido?

Instalador:

- Más sencillo de instalar

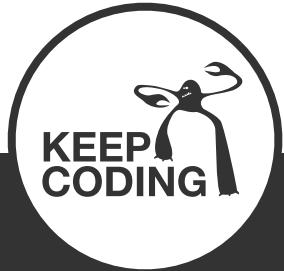
Instalador de paquetes (Homebrew, Chocolatey, apt-get, ...)

- Se instala sin permisos de administrador
- Más fácil de desinstalar





Desde el instalador oficial



A través del instalador

- Ir a <https://nodejs.org/>
- Install → descargar
- Lanzar .pkg|.exe
- Siguiente, siguiente, ...

(Solo OSX y Windows)

The screenshot shows the official Node.js website at https://nodejs.org/. The page has a dark header with the Node.js logo and navigation links for HOME, ABOUT, DOWNLOADS, DOCS, FOUNDATION, GET INVOLVED, SECURITY, and NEWS. Below the header, there is a brief introduction about Node.js being built on Chrome's V8 JavaScript engine and its event-driven, non-blocking I/O model. A large green button labeled "Download for OS X (x64)" is prominently displayed. At the bottom, there are links for Other Downloads, Changelog, and API Docs. The background features a blurred image of a server rack.

node.js®

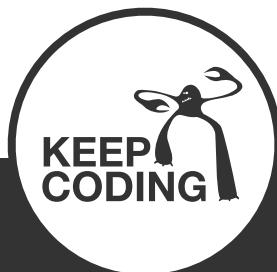
HOME | ABOUT | DOWNLOADS | DOCS | FOUNDATION | GET INVOLVED | SECURITY | NEWS

Node.js® is a JavaScript runtime built on [Chrome's V8 JavaScript engine](#). Node.js uses an event-driven, non-blocking I/O model that makes it [lightweight](#) and [efficient](#). Node.js' package ecosystem, [npm](#), is the largest ecosystem of open source libraries [in the world](#).

Current Version: v4.0.0

Download for OS X (x64)

Other Downloads | Changelog | API Docs



Instalará los ejecutables de node
y npm en la ruta:

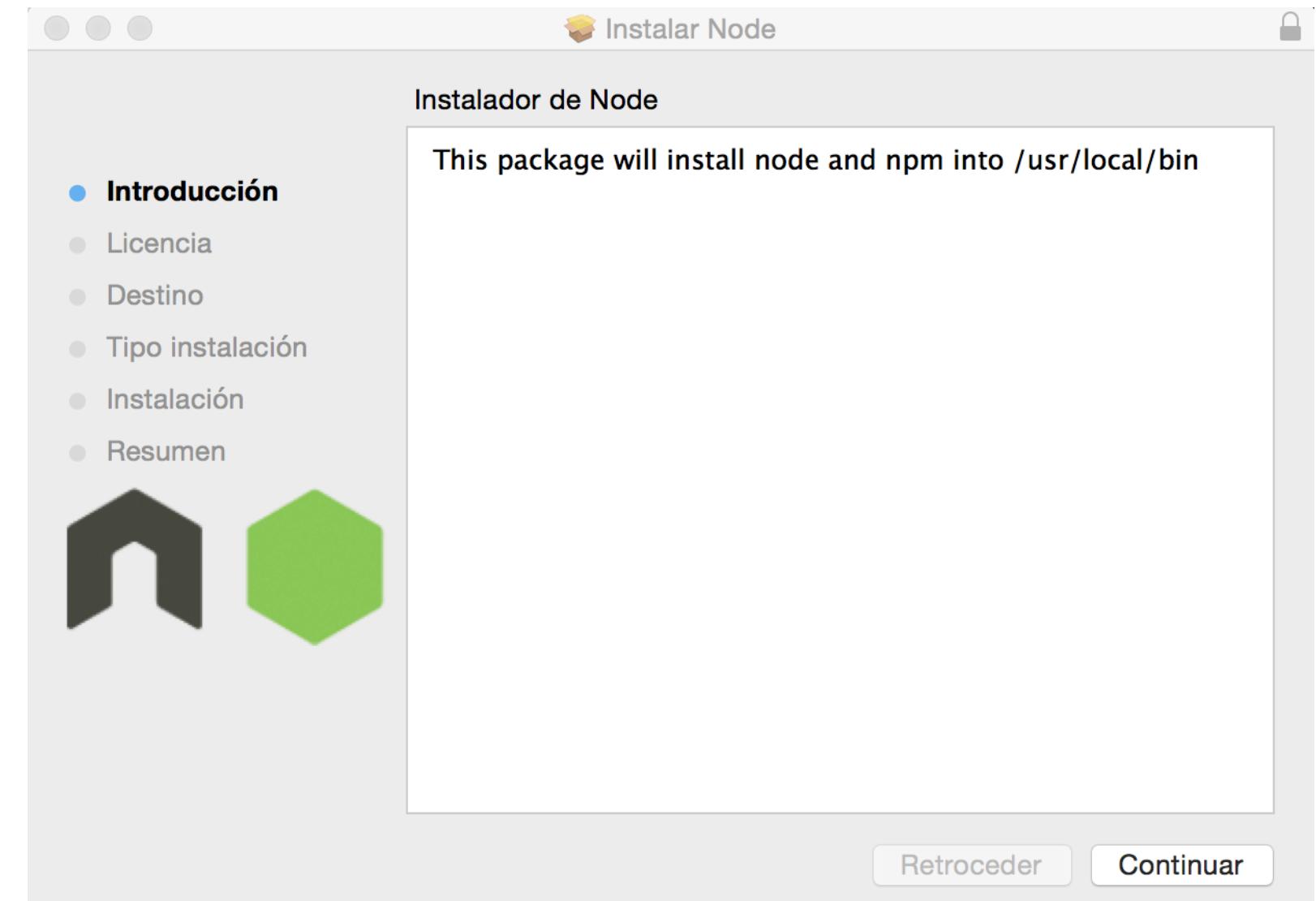
/usr/local/bin

Pondrá la carpeta node_modules
global en:

/usr/local/lib

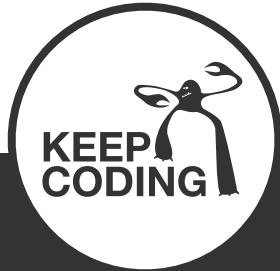
Vínculo dinámico de npm en bin:

```
npm@ -> ../../lib/node_modules/npm/bin/npm-cli.js
```



A partir de aquí ya podremos ejecutar:

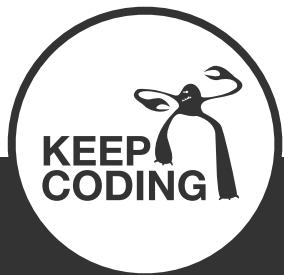
`node -v`



A través del instalador - actualizar

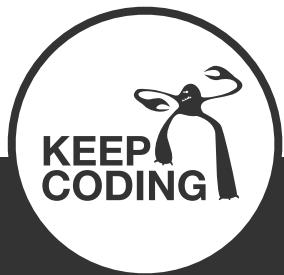
Para actualizar se repite el proceso:

- Descargar el instalador de la versión elegida
- Instalar encima
- Comprobar la nueva versión





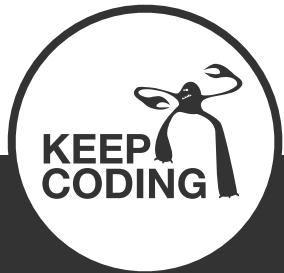
A través de un instalador de paquetes



A través de un instalador de paquetes

Por ejemplo:

- Chocolatey (Windows)
- Homebrew (macOS)
- En Linux: apt en Ubuntu, pacman en arch, etc



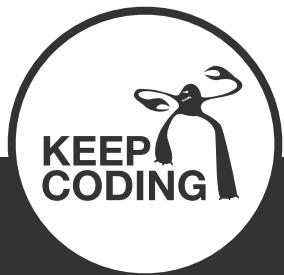
A través de un instalador de paquetes

Instalación:

1. Abre el Terminal
2. Escribe:

```
brew install node
```

Tras la instalación comprueba las versiones. (node -v)



A través de un instalador de paquetes - actualizar

Actualización:

1. Abre el Terminal

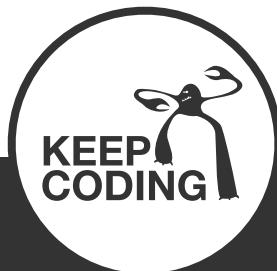
2. Escribe:

`brew update`

3. Luego:

`brew upgrade node`

Tras la actualización comprueba las versiones.

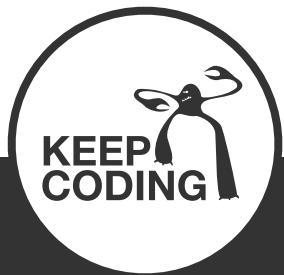


A través de un instalador de paquetes - desinstalar

Desinstalación:

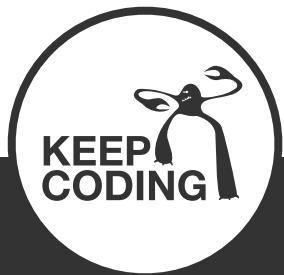
1. Abre el Terminal
2. Escribe:

```
brew uninstall node
```





■ Version managers



Instalar

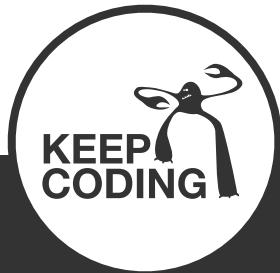
- En linux/mac - <https://github.com/nvm-sh/nvm>
- En windows - <https://github.com/coreybutler/nvm-windows>

```
nvm list
```

```
nvm install <version>
```

```
nvm use <version>
```

```
nvm use system (en linux)
```



nvm - cambio automático

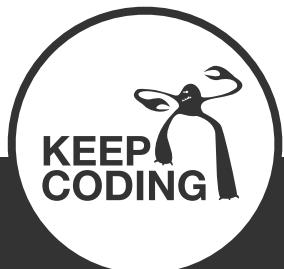
<https://github.com/nvm-sh/nvm#nvmrc>

```
echo "12.13.1" > .nvmrc
```

```
#posteriormente, tras entrar a la carpeta  
nvm use
```

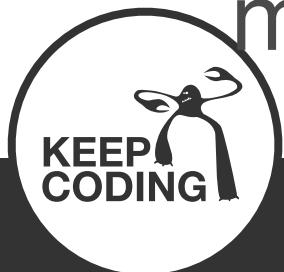
```
#tras salir de la carpeta para volver a la versión default  
nvm use default
```

Tip adicional para usuarios de linux/mac (f en el chat para windows)
<https://github.com/nvm-sh/nvm#deeper-shell-integration>



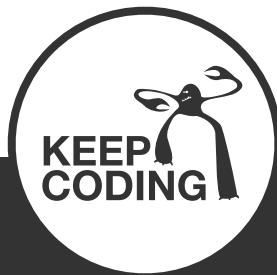
■ Version Managers Alternativos

- 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.



■ Version Managers Alternativos

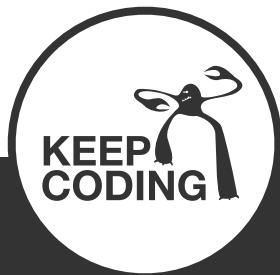
<https://pavel-romanov.com/5-node-version-managers-compared-which-is-right-for-you>





■ Un servidor básico

Ejercicio



Ejecutar

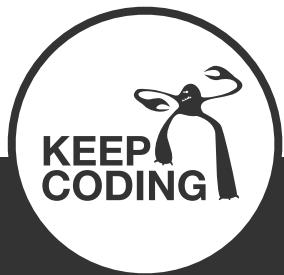
```
$ node index.js
```

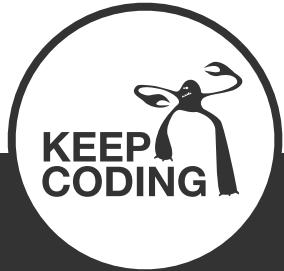
Con nodemon:

```
$ npm install nodemon -g  
$ nodemon
```

Con nodemon y npx:

```
$ npx nodemon
```



 NPM

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

Entre otras cosas nos permite:

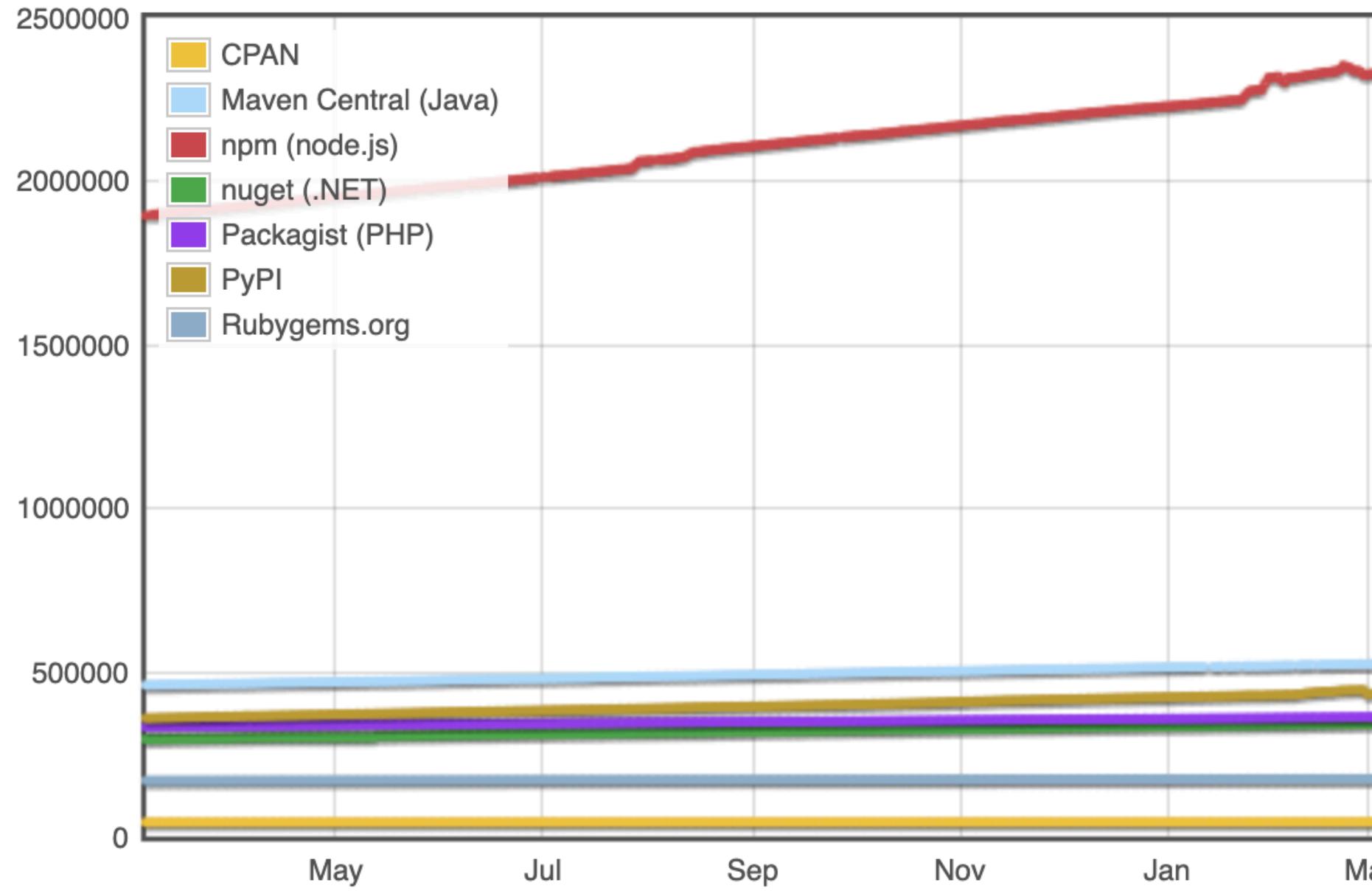
- Instalar librerías o programas de terceros
- Eliminarlas
- Mantenerlas actualizadas

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



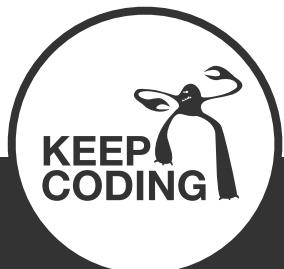
Module Counts

<http://www.modulecounts.com/>



Include

- Clojars (Clojure)
- CPAN
- CPAN (search)
- CRAN (R)
- Crates.io (Rust)
- Crystal Shards
- Drupal (php)
- DUB (dlang)
- Gopm (go)
- Hackage (Haskell)
- Hex.pm (Elixir/Erlang)
- Julia
- LuaRocks (Lua)
- Maven Central (Java)
- MELPA (Emacs)
- Nimble (Nim)
- npm (node.js)
- nuget (.NET)



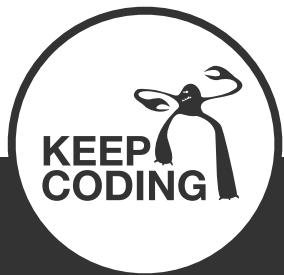
NPM - package.json

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/files/package.json>



NPM - package.json

\$ npm init

This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sane defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.

name: (myapp)

version: (1.0.0)

description: Esta es la descripción

entry point: (index.js) index.js

test command:

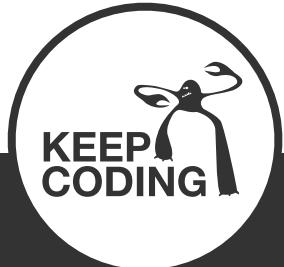
git repository:

keywords:

author: Javier Miguel

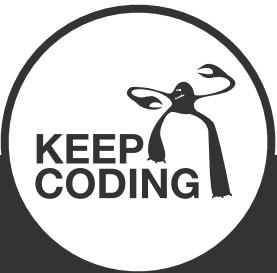
license: (ISC)

About to write to \Users\javi\www\cursonode\keepcoding\myapp\package.json:



NPM - 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": "Javier Miguel",  
  "license": "ISC"  
}
```

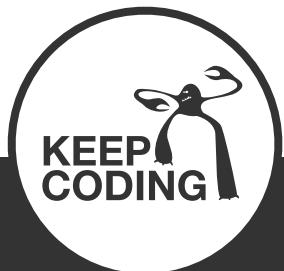


NPM - package.json

Instalar una librería (como por ejemplo chance)

```
npm install chance
```

```
{  
  "name": "myapp",  
  "version": "1.0.0",  
  "description": "Esta es la descripción",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "author": "Javier Miguel",  
  "license": "ISC"  
  "dependencies    "chance": "^0.8.0"  
  }  
}
```



NPM - global o local

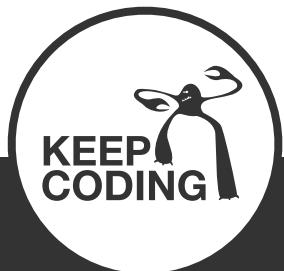
Instalación local, en la carpeta del proyecto

```
npm install <paquete>
```

Instalación global (en */usr/local/lib/node_modules*)

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

Si el paquete tiene ejecutables hará un vínculo a ellos en */usr/local/bin*



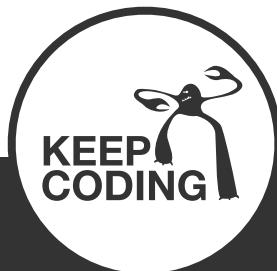
■ NPX - usar paquetes

Usar paquetes locales o remotos.

```
npx <paquete>
```

Si está en local se usa, sino se descarga (~/.npm/_npx), por ejemplo:

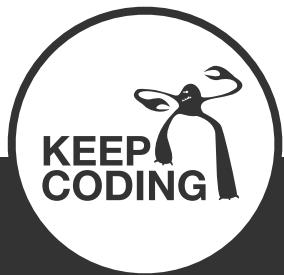
```
npx nodemon
```





■ Instalando un módulo

Ejercicio

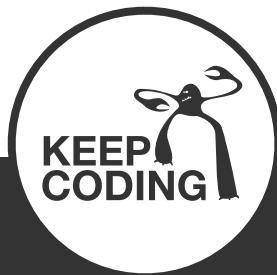


■ Instalando un módulo

Instalar el módulo chance (<https://github.com/victorquinn/chancejs>) y usar su generador de nombres en el servidor básico.

[npm init]

npm install chance

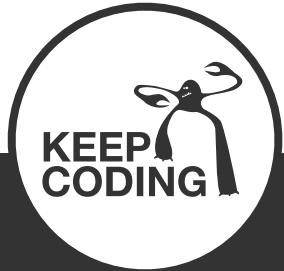




■ JS intermedio & avanzado



■ Hoisting



■ Tipos y variables - Hoisting

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

```
var pinto = 'my value';

function pinta() {
  console.log('pinto', pinto); // my value
  //var pinto = 'local value'; // esto hará hoisting de pinto y será undefined
}

pinta();
```

Si el código es complejo, declarar las variables al principio

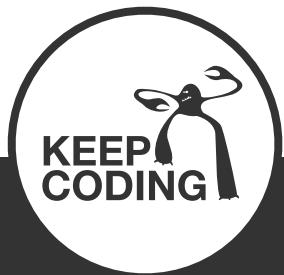
ejemplos/hoisted.js



■ Tipos y variables - Hoisting

Que valores se escribirán en la consola?

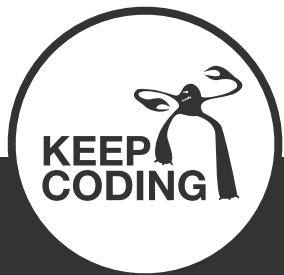
```
var x = 100;  
var y = function () {  
  
    if (x === 20) {  
        var x = 30;  
    }  
    return x;  
};  
  
console.log( x, y() );
```



■ Tipos y variables - Hoisting

Que valores se escribirán en la consola?

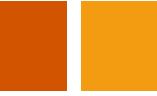
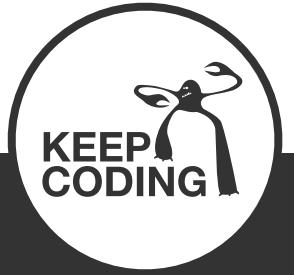
```
var x = 100;  
var y = function () {  
    var x; // -> HOISTING -> x ahora es undefined  
    if (x === 20) {  
        var x = 30;  
    }  
    return x; -> undefined  
};  
  
console.log( x, y() ); // x = 100 | y() = undefined
```



KEEP
CODING

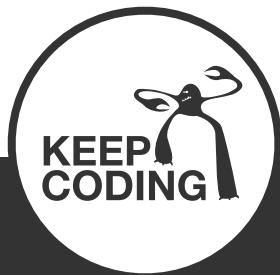


JSON



JavaScript Object Notation

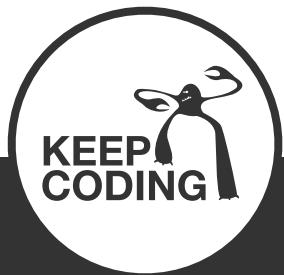
- Es un formato para intercambio de datos, derivado de la notación literal de objetos de Javascript.
- Se usa habitualmente para serializar objetos o estructuras de datos.
- Se ha popularizado mucho principalmente como alternativa a XML, por ser más ligero que este.



■ JSON

Convirtiendo un objeto a JSON:

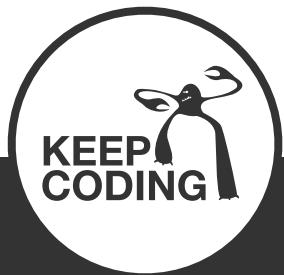
```
var empleado = {  
    nombre: 'Thomas Anderson',  
    profesion: 'Developer'  
};  
  
JSON.stringify(empleado);  
  
// produce un string  
'{"nombre": "Thomas Anderson", "profesion": "Developer"}'
```



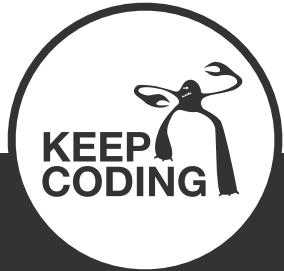
■ JSON

Convirtiendo un texto JSON en un objeto:

```
var textoJSON = '{"nombre": "Thomas  
Anderson", "profesion": "Developer"}';  
  
JSON.parse(textoJSON);  
  
// produce un objeto  
Object {nombre: "Thomas Anderson", profesion: "Developer"}
```



'use strict';



■ 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';
```

O también se puede habilitar solo para una función:

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

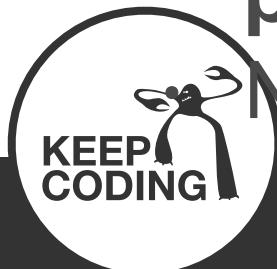


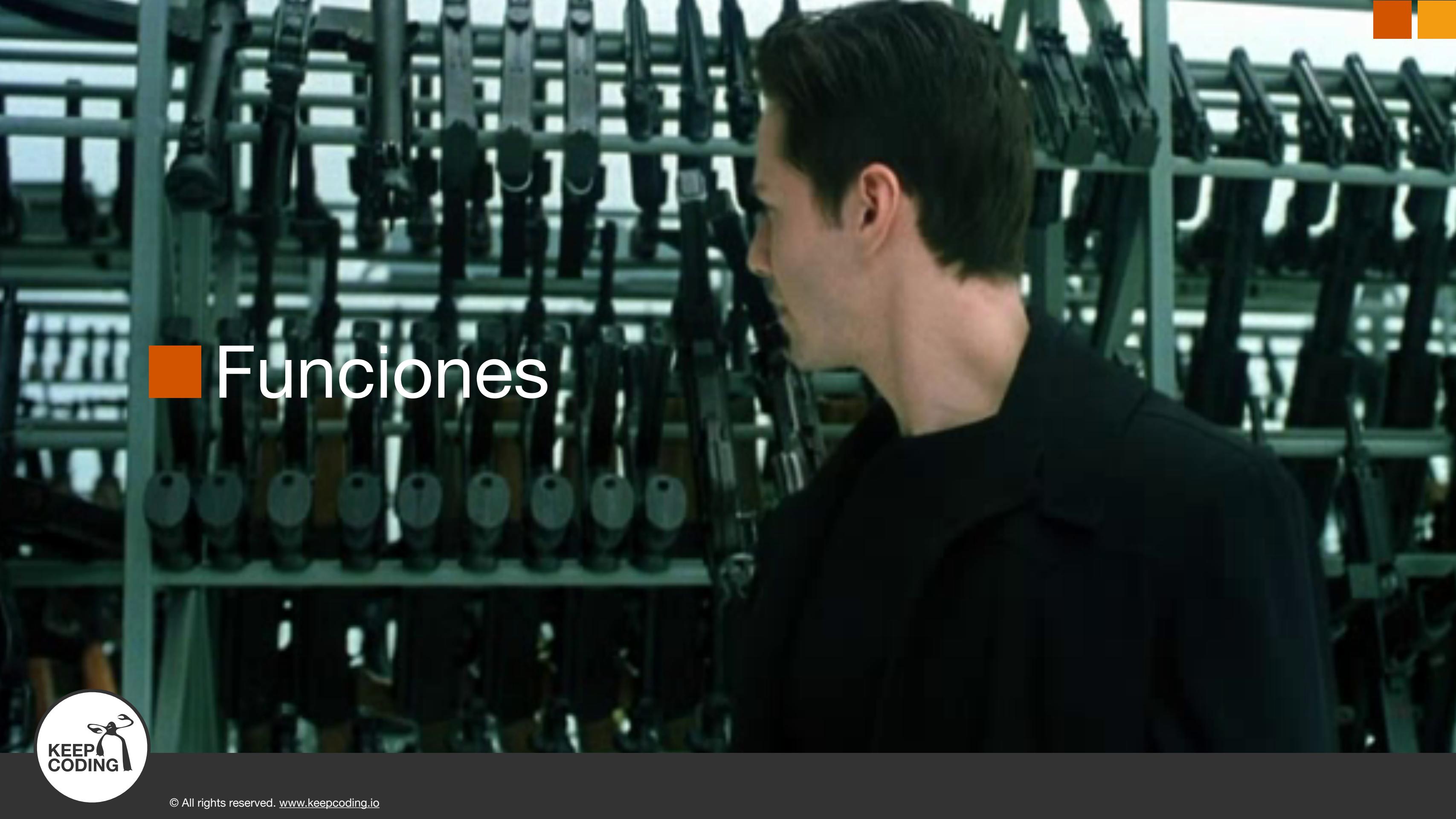
■ Modo estricto

Algunos ejemplos de los beneficios del modo estricto:

- **Las variables deben ser declaradas.** En *slippery 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 *slippery 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.**

No más *with*





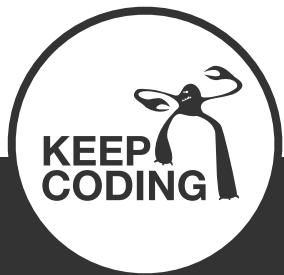
■ Funciones



■ Funciones

Las funciones son objetos

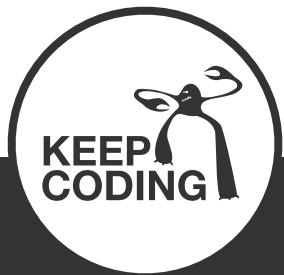
Por tanto también tienen propiedades y métodos



■ Funciones - declaración

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

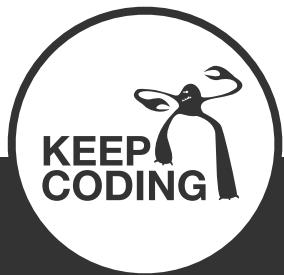
- Requieren un nombre
- Solo a nivel de programa o directamente en el cuerpo de otra función
- Hacen hoisting



■ Funciones - expression

```
var sumar = function(numero1, numero2) {  
    return numero1 + numero2;  
}
```

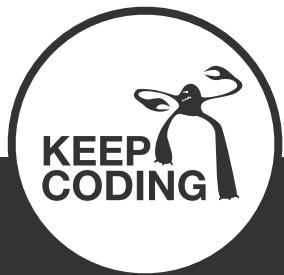
- Como son expresiones, se pueden definir en cualquier sitio donde pueda ir un valor. Por ejemplo, podemos pasarlas como parámetro
- No hacen 'hoisting', se pueden usar solo después de su definición
- Pueden tener un nombre, pero solo sería visible dentro de su cuerpo



Métodos

```
var calculadora = {  
    suma : function(a, b) { return a + b; },  
    mult : function(a, b) { return a * b; }  
}
```

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

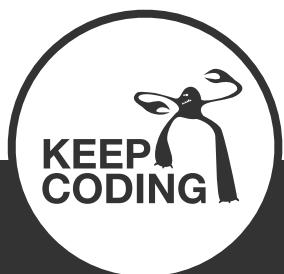


■ Funciones - instancias

```
function Fruta() {  
    var nombre, familia;  
    this.getNombre = function() { return nombre; };  
    this.setNombre = function(value) { nombre = value; };  
}  
  
var limon = new Fruta();  
limon.setNombre("Citrus limon");
```

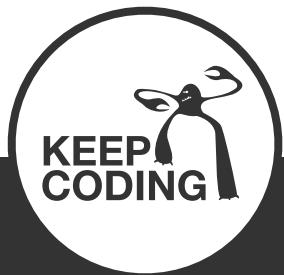
Cuando se usa **new** al invocar una función, se comporta como un constructor de objetos.

ejemplos/instancias.js





■ Callbacks



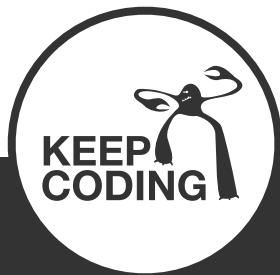
■ Callbacks

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 estar en pausa hasta llamarla.

```
console.log('empiezo');

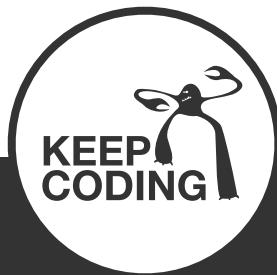
setTimeout(function() {
  console.log('he terminado');
}, 2000);
```



■ Callbacks

En node.js todos los usos de IO (entrada/salida) deberían ser asíncronos.

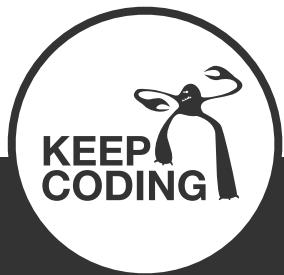
Si tras una llamada a una función asíncrona queremos hacer algo, como comprobar su resultado o si hubo errores, le pasaremos **un argumento más**, una expresión de tipo función (callback), para que la invoque cuando termine.



■Callbacks

```
function suma( n1, n2, callBack) {  
    var resultado = n1 + n2;  
    callBack(resultado);  
}
```

```
suma(1, 5, function(res){ console.log(res); } );
```



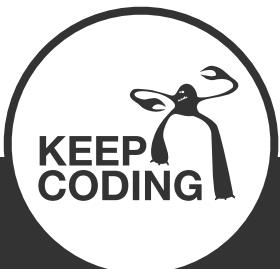
Callbacks

```
function suma( n1, n2, callBack) {  
    var resultado = n1 + n2;  
    callBack(resultado);  
}
```

```
suma(1, 5, function(res){ console.log(res); } );
```



Pasamos una función como tercer parámetro.
Le decimos: "cuando termines haz esto"



Callbacks

```
function suma( n1, n2, callBack ) {  
    var resultado = n1 + n2;  
    callBack(resultado);  
}
```

```
suma(1, 5, function(res){ console.log(res); } );
```

Esa función la recibimos y la llamamos callback



Callbacks

```
function suma( n1, n2, callBack ) {  
    var resultado = n1 + n2;  
    callBack(resultado);  
}
```

```
suma(1, 5, function(res){ console.log(res); } );
```

Cuando hemos terminado llamamos a la
función de callback



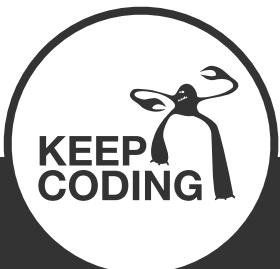
Callbacks

```
function suma( n1, n2, callBack ) {  
    var resultado = n1 + n2;  
    callBack(resultado);  
}
```

```
suma( 1, 5, function(res){ console.log(res); } );
```



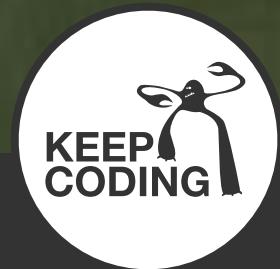
El cuerpo del callback recibe el resultado y lo utiliza





Ejercicio

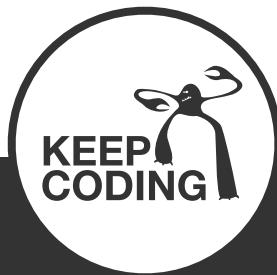
Haciendo una función asíncrona



Ejercicio - función asíncrona

Hacer una función que reciba un texto y tras 2 segundos lo escriba en la consola.

La llamaremos *escribeTras2Segundos*

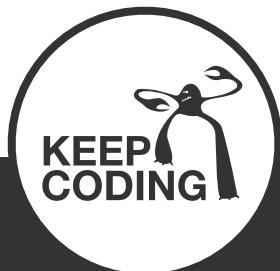


Ejercicio - función asíncrona

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

Al final escribir en la consola "Fin".

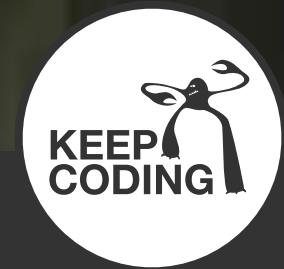
Llamada1 - 2 secs. - **texto1** - llamada2 - 2 secs. - **texto2** - Fin





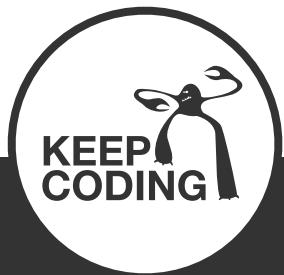
Ejemplo

Haciendo un bucle asíncrono



Ejercicio - bucle asíncrono

Repitamos la llamada a `escribeTras2Segundos` varias veces



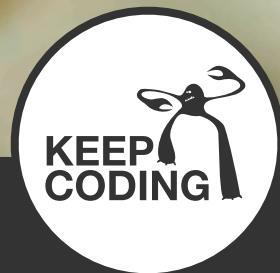


"No intentes doblar la cuchara,
eso es *impossible*..."

En vez de eso procura comprender la verdad

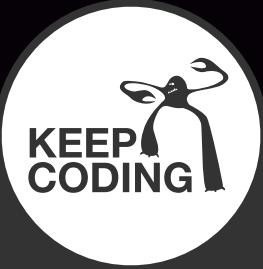
- ¿Que verdad?

Que no hay cuchara"



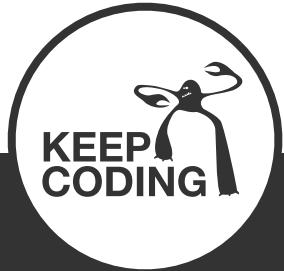


■ Truthy and Falsy



■ Que son Truthy y Falsy

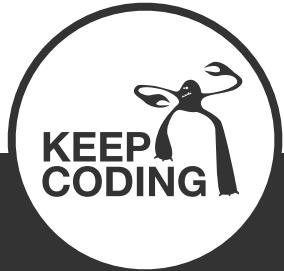
En javascript todo tiene un valor booleano, generalmente conocido como truthy o falsy.



■ Como se produce

```
var variable = "value";
if(variable) {
    console.log("Soy truthy");
}

variable = 0;
if(variable) {
    console.log("...");
} else {
    console.log("Soy falsy");
}
```

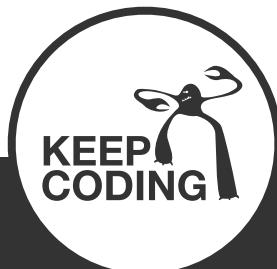


■ Reglas de truthy / falsy

Los siguientes valores siempre son **falsy**:

- **false**
- **0** (cero)
- **""** (cadena vacía)
- **null**
- **undefined**
- **NaN** (valor especial de tipo Number que significa Not-a-Number!)

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



Comparando Falsys

Los valores `false`, `0` (cero), y `""` (cadena vacía) son equivalentes:

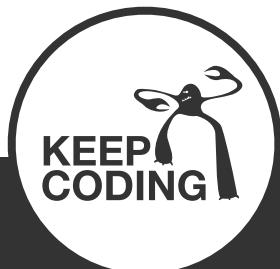
```
var c = (false == 0); // true
var d = (false == ""); // true
var e = (0 == ""); // true
```

Los valores `null` y `undefined` no son equivalentes con nada, excepto con ellos mismos:

```
var f = (null == false); // false
var g = (null == null); // true
var h = (undefined == undefined); // true
var i = (undefined == null); // true
```

Y por último, el valor `NaN` no es equivalente con nada, ni siquiera consigo mismo:

```
var j = (NaN == null); // false
var k = (NaN == NaN); // false
```



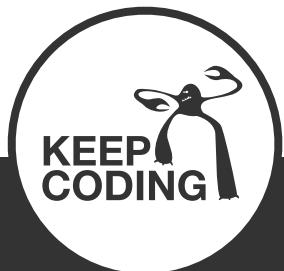
■ La cosa se complica

```
if ( [] ) { /*se ejecuta*/ }
/*Array es instancia de Object, y existe*/
```

```
if ( [] == true ) { /*no se ejecuta*/ }
/*comparamos valores!, [].toString -> "" -> falsy*/
```

```
if ( "0" == 0 ) // true (se convierten a números)
```

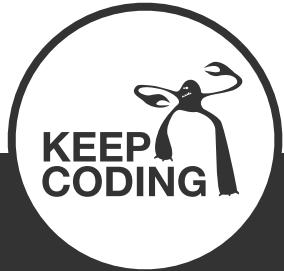
```
if ( "0" ) {console.log('si')} // "si" (se evalúa el string)
```





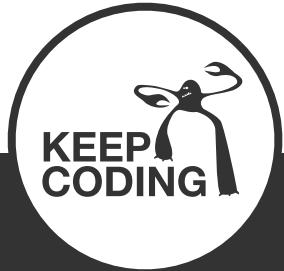
■ La cosa se complica

<https://www.destroyallsoftware.com/talks/wat>





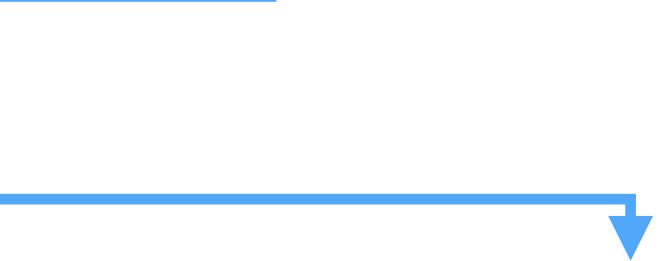
■ Una solución por favor!



■ Truthy and Falsy

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

```
var melio = ( false == 0 ); // true  
  
var seguro = ( false === 0 ); // false
```



Se comparan **primero por su tipo y luego por su valor**





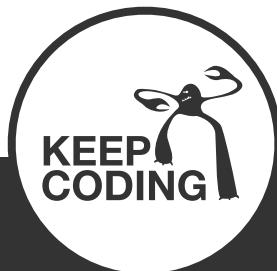
this



this

La palabra clave `this` trata de representar al objeto que llama nuestra función como 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.

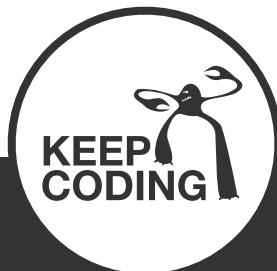


CUIDADO!

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

```
console.log(this); // window en un browser, global en node

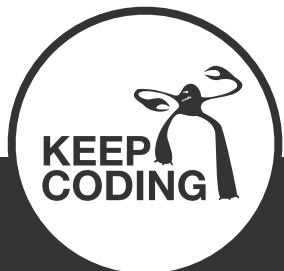
function pinta(){
  console.log(this); // window en un browser, global en node
}
```



Averigua porque...

```
var persona = {  
    name: 'Luis',  
    surname: 'Gomez',  
    fullname: function() {  
        console.log(this.name + ' ' + this.surname);  
    }  
};  
  
persona.fullname(); // Luis Gomez  
setTimeout(persona.fullname, 1000); // undefined undefined  
  
// pista: quien está invocando realmente la función fullName?
```

ejemplos/this.js

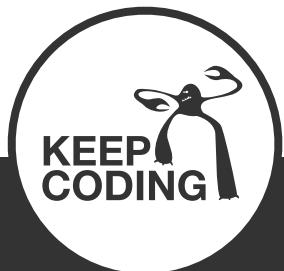


this

Como manejarlo? Le asignamos this con bind

```
var persona = {  
    name: 'Luis',  
    surname: 'Gomez',  
    fullname: function() {  
        console.log(this.name + ' ' + this.surname);  
    }  
};  
  
setTimeout(persona.fullname.bind(persona), 1000); // Luis Gomez
```

ejemplos/this.js



this

Otras formas de asignar this a una función

```
funcion.call(thisArg[, arg1[, arg2[, ...]]])
```

```
persona.iniciales.call(programa, 2, true);
```

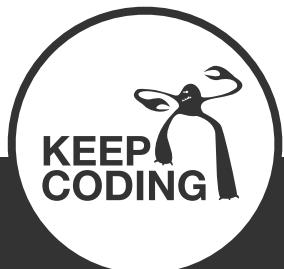
```
funcion.apply(thisArg[, argsArray])
```

```
persona.iniciales.apply(programa, [2, false]);
```

La diferencia es:

- en **call** hay que poner todos los argumentos separados por **comas**
- en **apply** se le pasan como un **array**

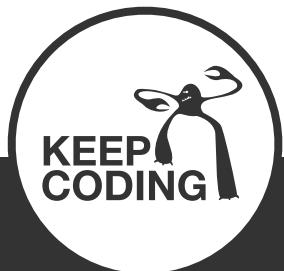
ejemplos/this2.js



this - uso en constructores

Cuando usamos this en un constructor de objetos, este apuntará al objeto returned por el constructor. Pero cuidado con quien llama a los métodos!

```
function Coche() {  
    this.ruedas = 4;  
    this.logRuedas = function() {  
        console.log('tiene ' + this.ruedas);  
    }  
}  
  
var coche = new Coche();  
console.log(coche.ruedas); // 4  
coche.logRuedas(); // tiene 4  
setTimeout(coche.logRuedas, 1000); // tiene undefined
```



A scene from The Matrix featuring Neo (Keanu Reeves) standing in a field of glowing green spheres. He is wearing his signature black trench coat and sunglasses. His right hand is raised, palm facing forward, with several small glowing circles floating around it. The background is a dense field of similar glowing spheres.

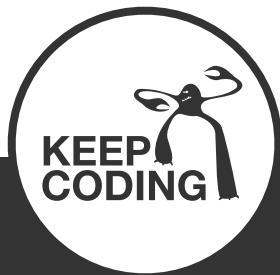
■ Closures



Closures

Un closure 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).

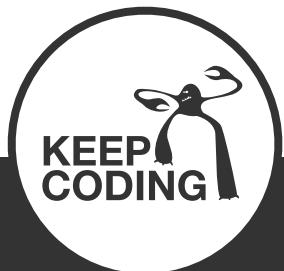


Closures

Ejemplo:

```
function creaClosure(valor) {  
    return function() {  
        return valor;  
    }  
}
```

ejemplos/closure.js



Closures

Algo más elaborado:

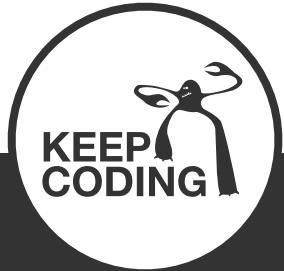
```
function creaAgente(nombre) {
    var edad = 0;
    return {
        ponNombre: function(nuevoNombre) {
            nombre = nuevoNombre;
        },
        leeNombre: function() {
            return nombre;
        },
        ponEdad: function(nuevaEdad) {
            edad = nuevaEdad;
        },
        leeEdad: function() {
            return edad;
        }
    }
}
```

ejemplos/closure2.js





Prototipos



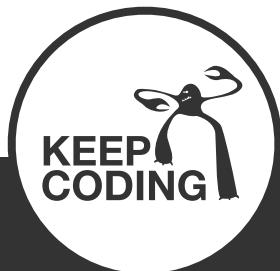
■ Prototype chain

Casi todo en Javascript es un objeto. Cada objeto tiene una propiedad interna llamada prototype que apunta a otro objeto.

Su objeto prototipo tiene a su vez una propiedad prototype que apunta a otro objeto, y así sucesivamente.

A esto se le llama cadena de prototipos.

Si sigues la cadena en algún momento llegarás al objeto Object, cuyo prototipo es null.

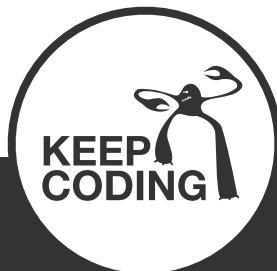


■ Prototipos

Cuando pides una propiedad a un objeto el interprete mira a ver si la tiene ese objeto. Si no la encuentra mira a ver si la tiene su prototipo, y así hasta llegar al final de la cadena.

Si no lo encuentra devuelve el error correspondiente.

Esto nos permitiría hacer algo parecido a clases e implementar herencia.

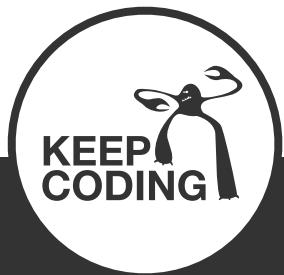


■ Prototipos

```
function Persona(name) {  
    this.name = name;  
}  
  
Persona.prototype.saluda = function() {  
    console.log("Hola, me llamo " + this.name);  
};
```

Esto hará que todas las personas sepan saludar, ¡incluso las que ya estén creadas!

ejemplos/prototipos.js



■ Prototipos - herencia

```
function Agente(name) {  
    Persona.call(this, name);  
    // heredamos el constructor  
    // Esto ejecutará el constructor de Persona sobre el this de Agente  
    // definiendo en el this de Agente sus propiedades.  
    // Es como llamar a "super" en otros lenguajes  
}  
  
// heredamos las propiedades de prototipo  
Agente.prototype = new Persona(); // heredaría name y saluda()
```

Así Agente hereda de Persona.



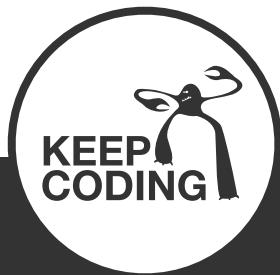
ejemplos/prototipos.js

Extender

Podemos copiar las propiedades de un objeto a otro.

Esto se suele llamar extender un objeto con otro.

```
const config = Object.assign({}, destino, origen);
```

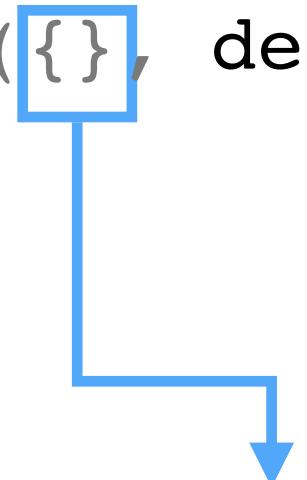


Extender

Podemos copiar las propiedades de un objeto a otro.

Esto se suele llamar extender un objeto con otro.

```
const config = Object.assign({}, destino, origen);
```



El objeto destino **es modificado**



■ Herencia múltiple - mixins

Una forma de conseguir herencia múltiple es usar el patrón mixin:

- Para conseguirlo heredamos del principal (ya lo hemos visto antes)
- Luego extendemos el prototipo del heredado con los mixins:

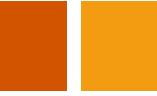
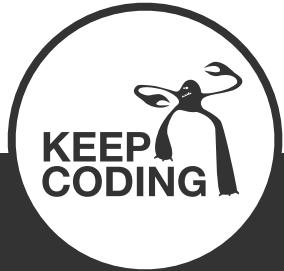
```
Agente.prototype = Object.assign(Agente.prototype, mixin);
```

ejemplos/mixin.js



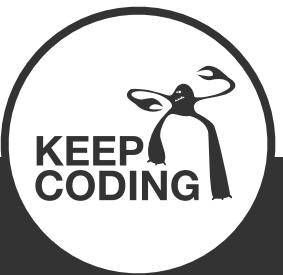


Clases



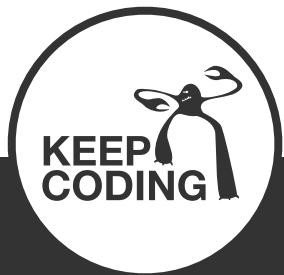
■ Clases

```
class Mascota {  
  
    constructor(nombre) {  
        this.nombre = nombre;  
    }  
  
    saluda() {  
        console.log(`Hola soy ${this.nombre}`);  
    }  
}
```



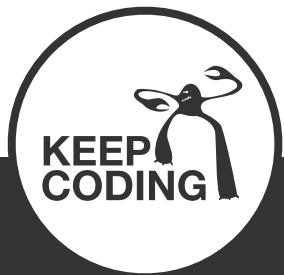
■ Clases

```
const mascota = new Mascota('Toby');  
  
mascota.saluda();
```



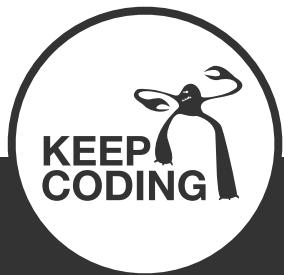
■ Clases

```
class Perro extends Mascota {  
    constructor(nombre) {  
        super(nombre);  
    }  
  
    let perro = new Perro('Niebla');  
  
    perro.saluda();
```





■ Process



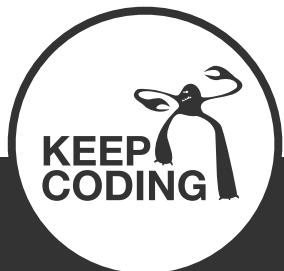
■ Process

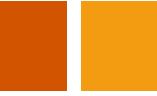
El objeto global process tiene muchas propiedades que nos serán útiles, como `process.platform`, que en OS X nos dirá '`darwin`', en linux '`linux`', etc.

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

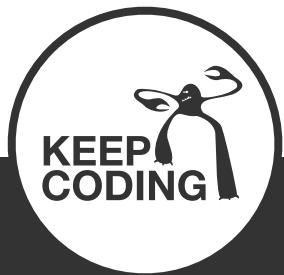
O eventos como `process.on('exit', callback)` donde podemos hacer cosas antes de salir.

[ejemplos/process.js](#)



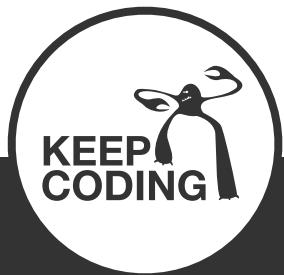


■ Events





■ Event loop

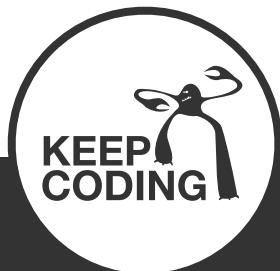


■ Event loop

Node usa un solo hilo.

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

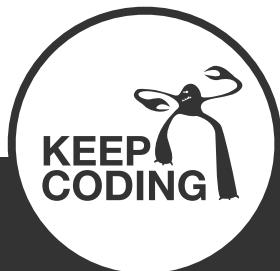
La siguiente vuelta mira a ver si ha terminado algún callback y si es así ejecuta su handlers.



■ Non blocking

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

Por eso todos las llamadas a funciones que usan IO (por ejemplo escritura o lectura en disco, la red, bases de datos, etc) se hacen de forma asíncrona. Se aparcan para que nos avisen cuando terminen.





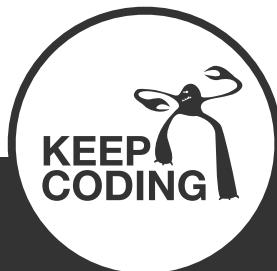
KEEP
CODING

■ Events

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

Usando el EventEmitter podemos colgar eventos de un identificador.

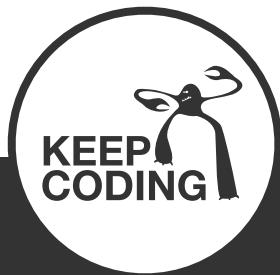
```
eventEmitter.on('llamar telefono', suenaTelefono);
```



■ Events

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

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



■ Events

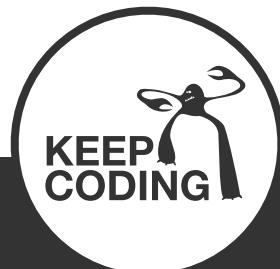
Incluso podemos darle argumentos al identificador para que se los dé a los manejadores.

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

Nuestro manejador recibirá el parametro

```
var suenaTelefono = function(quien) {  
    ...  
}
```

ejemplos/
eventemitter.js



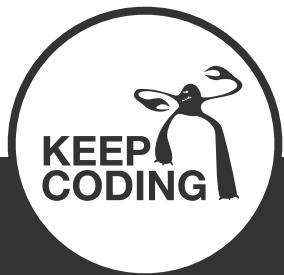
■ Events

Son cómodos para procesar streams.

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

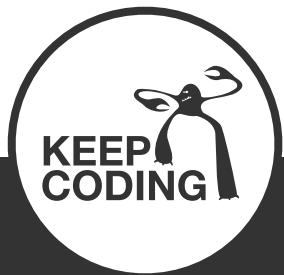
```
process.stdin.on('data', function(data) {  
  file.write(data);  
});
```

[ejemplos/eventedio.js](#)





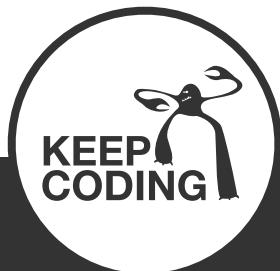
■ Módulos



Módulos

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



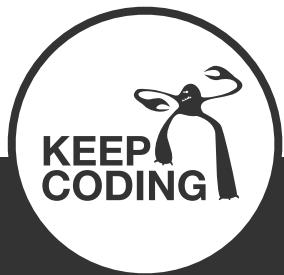
Módulos

Un módulo básico:

```
// modulo.js  
console.log('Hola desde un módulo!');
```

```
// index.js  
require('./modulo.js');
```

ejemplos/modulos/
basico.js



Módulos

Donde busca Node.js los módulos

1. Si es un módulo del core (node:<module>)
2. Si empieza por './' or '/' or '../' va a la ruta calculada desde la situación del fichero actual
3. Módulos en la carpeta node_modules local
4. Módulos en la carpeta node_modules global

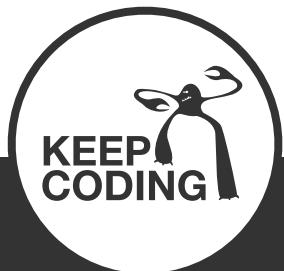


Módulos

¿module.exports o exports?

- exports es un alias de module.exports.
- Node lo crea automáticamente
- Para hacer exports con nombre se pueden usar los dos indistintamente

Si asignamos algo directamente a exports deja de ser un alias.
Solo podemos usarlo con exports.loquesea

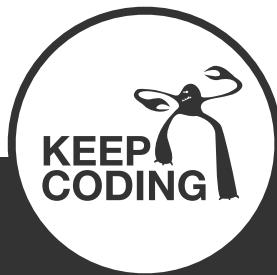


Módulos

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

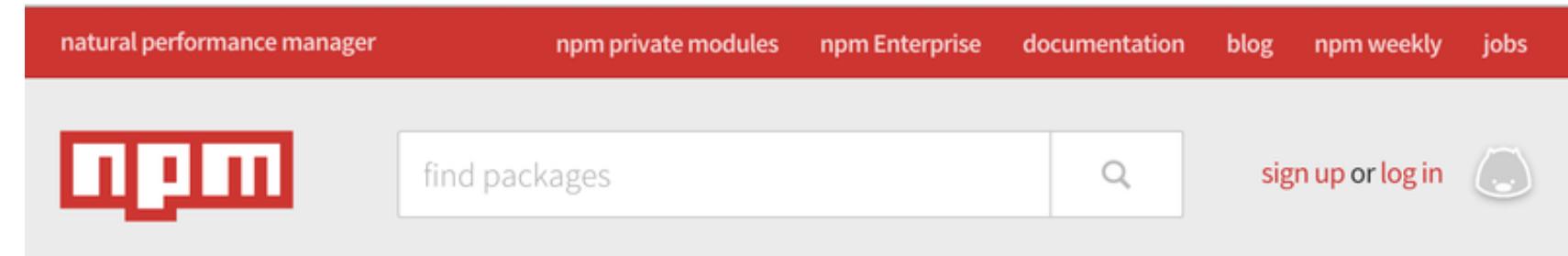
Las siguientes llamadas a require con la misma ruta devolverán el mismo objeto que se creo en la primera llamada.

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



Módulos

Principalmente podemos encontrar módulos de terceros en npmjs.com



npm is the package manager for javascript.

179.374
total packages

37.682.863
downloads in the last day

598.105.505
downloads in the last week

2.198.864.721
downloads in the last month

packages people 'npm install' a lot



■ ES Modules

Los módulos de EcmaScript están disponibles de manera estable en Node.js

<https://medium.com/dailyjs/javascript-module-cheatsheet-7bd474f1d829>

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

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

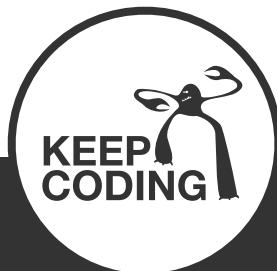


■ ES Modules

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*

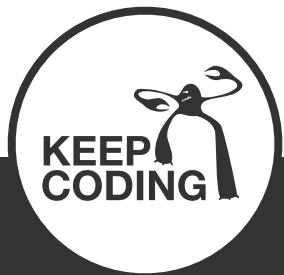
* import.meta.dirname / import.meta.filename están disponibles a partir de node.js v20.11 y v21.2





Ejercicio

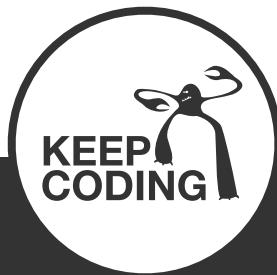
Versión de un módulo



Ejercicio - versión de un módulo

Hacer una función llamada **versionModulo** que reciba un nombre de módulo y un callback.

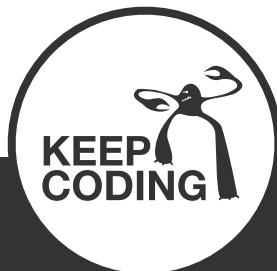
Debe devolver un posible error y la versión del módulo en el callback.



Ejercicio - versión de un módulo

Ingredientes:

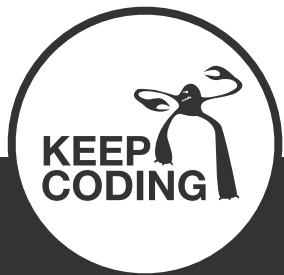
- **fs.readFile** (https://nodejs.org/api/fs.html#fs_fs_readfile_filename_options_callback)
- **path.join** (https://nodejs.org/api/path.html#path_path_join_path1_path2)
- **JSON.parse** (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse)



Ejercicio - versión de un módulo

La probaremos con un código como este:

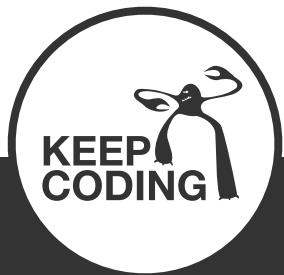
```
versionModulo( 'chance' , function(err, str) {  
  if (err) {  
    console.error('Hubo un error: ', err);  
    return;  
  }  
  console.log('La version del módulo es:', str);  
} );
```





Ejercicio

Versión módulos

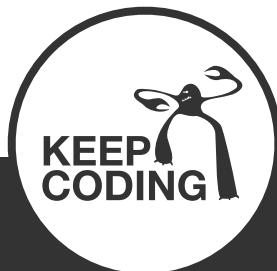


Ejercicio - versión de módulos

Modificar la función **versionModulo** para que consiga la versión de todos los módulos instalados en mi proyecto.

Más ingredientes:

- `fs.readdir` (https://nodejs.org/api/fs.html#fs_fs_readdir_path_callback)
- `async.concat` (<https://github.com/caolan/async#concat>)
- `async common pitfalls` <https://github.com/caolan/async#common-pitfalls-stackoverflow>
- `fs.statSync` (https://nodejs.org/api/fs.html#fs_fs_statsync_path)

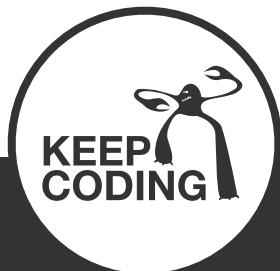


Ejercicio - versión de un módulo

Recibirá un callback. Debe devolver un posible error y un array con los nombres y versiones de los módulos que encuentre.

La probaremos con un código como este:

```
versionModulos(function(err, moduleArr) {  
  if (err) {  
    console.error('Hubo un error: ', err);  
    return;  
  }  
  console.log('Los módulos son:', moduleArr);  
});
```



Express.js

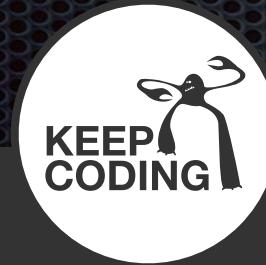


Javier Miguel

@JavierMiguelG

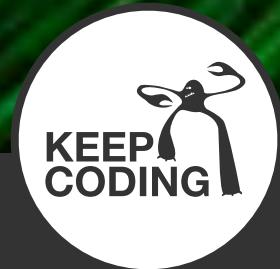
jamg44@gmail.com

CTO & Freelance Developer





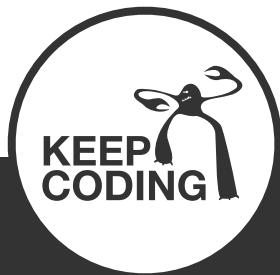
Estructurar nuestra aplicación



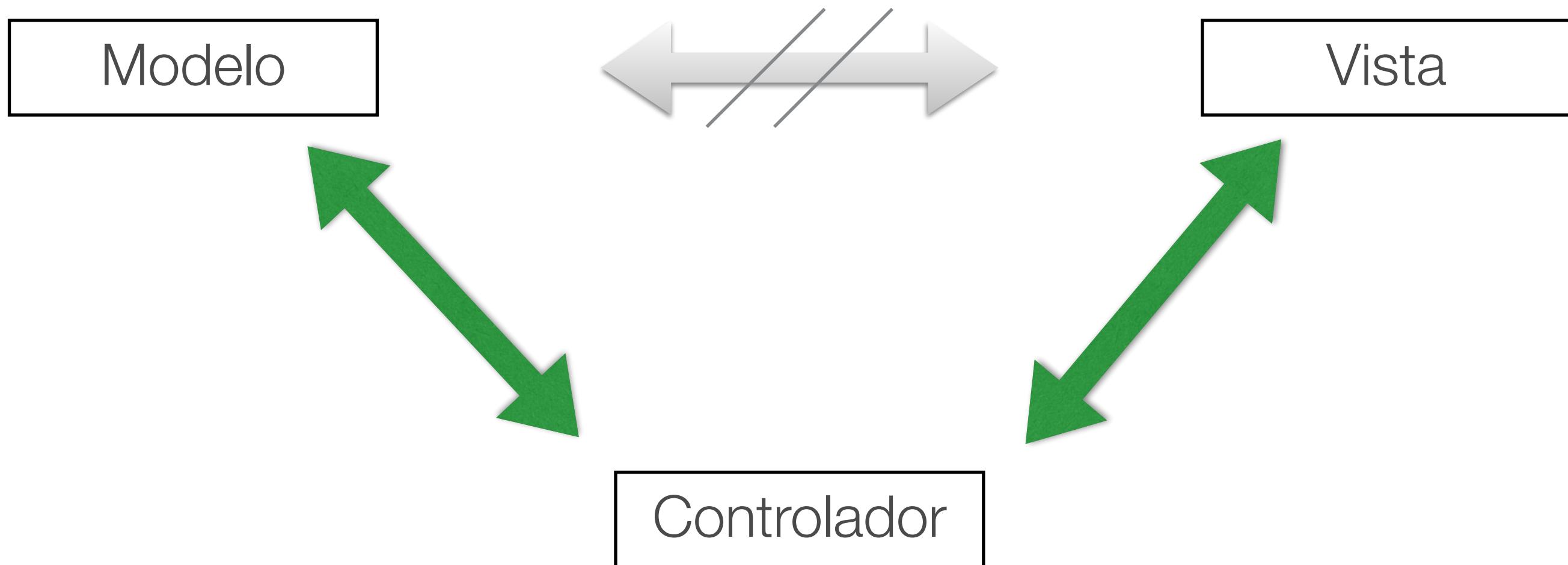
Estructura

En Node.js podemos usar cualquier patrón para estructurar nuestro código.

El patrón MVC es comúnmente usado por muchos desarrolladores por su buen resultado.

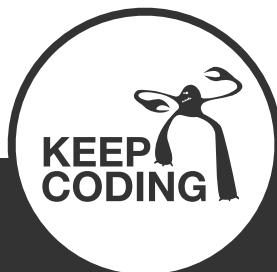


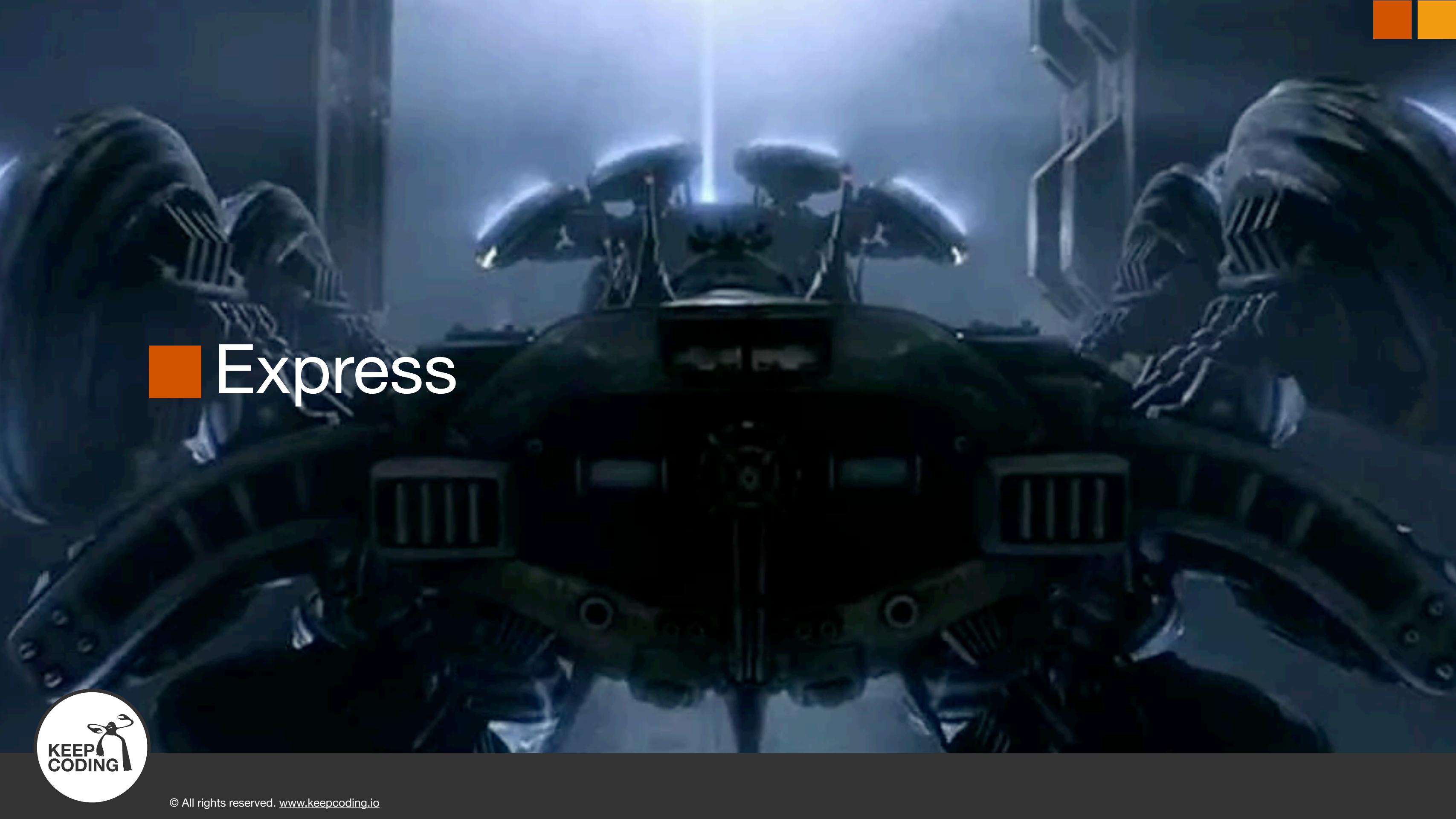
MVC



MVC

- El controlador recoge datos del modelo y los da a la vista para que los represente en su interfaz
- La vista recibe las acciones del usuario y da información al controlador que opcionalmente guardará lo necesario en el modelo
- El modelo avisará el controlador de que hay nuevas versiones de los datos, y este opcionalmente las entregará a la vista





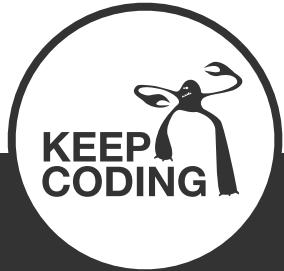
 Express



■ Express

Express es un framework web para Node.js

<http://expressjs.com/>

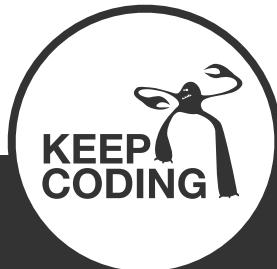


■ Web Frameworks

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

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

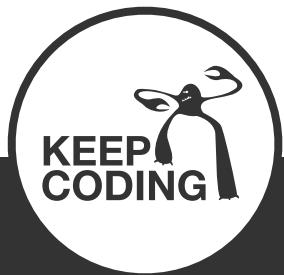
Muchos de ellos extienden la funcionalidad de Express.js, siendo este el más usado.



■ Express

Podemos revisar toda la funcionalidad en:

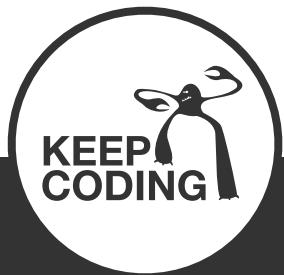
- <http://expressjs.com/api.html>





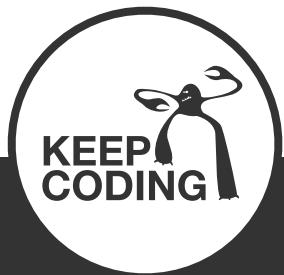
■ Ejercicio

Una app básica con Express





■ Usando Express Generator



■ Usando Express Generator

Express Generator nos crea una estructura **base** para una aplicación.

```
$ [sudo] npm install express-generator -g
```

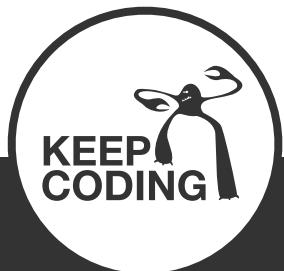
```
$ express -h
```

```
$ express <nombreApp> [--ejs ]
```

```
$ cd <nombreApp>
```

```
$ npm install
```

ejemplos/express/generated

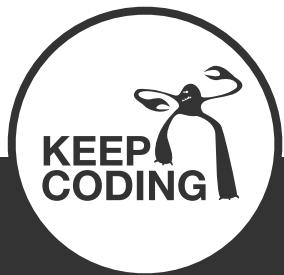


■ Express Generator

Como arrancar nuestra aplicación:

```
$ npm start
```

```
// entorno desarrollo, puerto por defecto (3000)
```

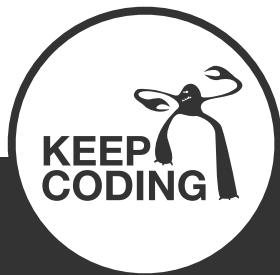


■ Express Generator

Podemos establecer variables de entorno para variar la forma de arranque:

```
$ NODE_ENV=production npm start
```

```
// entorno producción
```

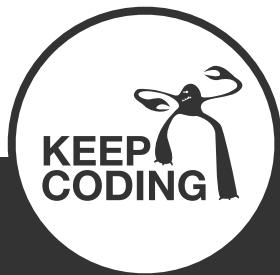


■ Express Generator

Podemos establecer variables de entorno para variar la forma de arranque:

```
$ DEBUG=nombreApp:* PORT=3001 NODE_ENV=production npm start
```

```
// con log debug activado  
// puerto 3001  
// entorno producción
```



■ Express Generator

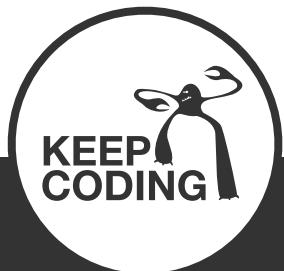
Si queremos podríamos incluir esto en el comando start de npm, especificándolo en el package.json

```
npm install --save-dev nodemon cross-env
```

...

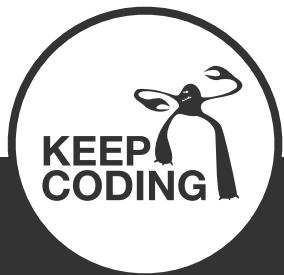
```
"scripts": {  
  "dev": "cross-env DEBUG=nombreApp:* PORT=3000 nodemon"  
},
```

...





■ Middlewares



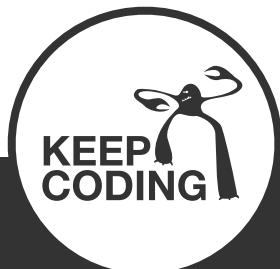
Middlewares

Un Middleware es un handler que se activa ante unas determinadas peticiones o todas.

Debe responder o llamar next().

Podemos poner tantos middlewares como nos haga falta.

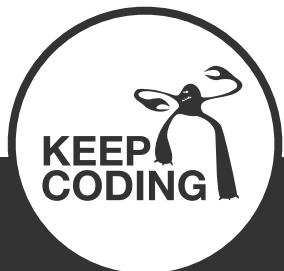
Los middleware's usan callbacks por defecto.



Middlewares

Tiene acceso al objeto de la petición (request), al objeto de la respuesta (response) y al siguiente middleware (next).

Si un middleware no quiere responder en una llamada debe llamar al siguiente con `next()` para pasarle el control, de lo contrario la petición quedará sin respuesta, y el cliente que la hizo (por ejemplo un browser) se quedará esperando hasta su tiempo límite de time-out.

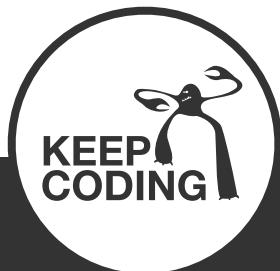


Middlewares - app

Se conecta al objeto instancia de la aplicación (app) con app.use o app.METHOD, por ejemplo:

```
var app = express();

app.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
});
```

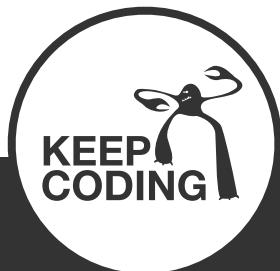


Middlewares - router

Se conecta a un router con router.use o router.METHOD, por ejemplo:

```
var router = express.Router();

router.use(function (req, res, next) {
  req.user = userModel.find(req.body.userId);
  next();
});
```

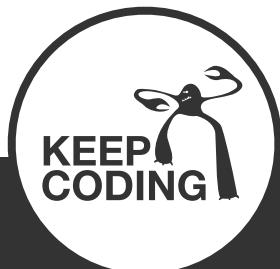


Middlewares - error

Los pondremos los últimos, tras todas nuestras rutas.
Reciben un parámetro más err.

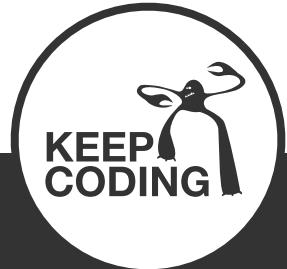
```
app.use(function(err, req, res, next) {  
  console.error(err.stack);  
  res.status(500).send('Something broke!');  
});  
  
// en una ruta anterior podemos haber hecho next(err);
```

Podemos devolver lo que nos convenga, un JSON con un error, una página de error, un texto, etc.





■ Rutas



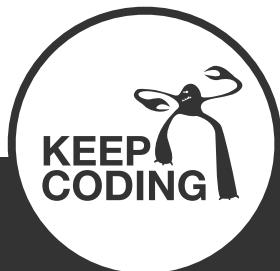
Rutas

Express define las rutas para tener la siguiente estructura:

app .METHOD (PATH , HANDLER)

donde:

- app es la instancia de express
- METHOD es el método de la petición HTTP (GET, POST, ...)
- PATH es la ruta de la petición
- HANDLER es la función que se ejecuta si la ruta coincide.



Rutas

HTTP pone a nuestra disposición varios métodos, de los cuales generalmente hacemos distintos usos:

- **GET** para pedir datos, es idempotente (p.e. listas)
- **POST** para crear un recurso (p.e. crear un usuario)
- **PUT** para actualizar, es idempotente (p.e. guardar un usuario existente)
- **DELETE** eliminar un recurso, es idempotente (p.e. eliminar un usuario)

* idempotente: si lo ejecutas varias veces los resultados no cambian

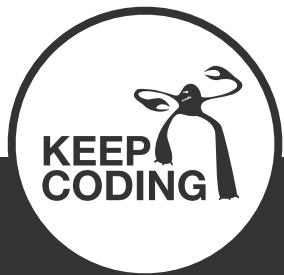


Rutas

Por lo tanto podríamos construir rutas como estas:

```
app.get('/', function (req, res) {  
  res.send('Hello World!');  
});
```

```
app.post('/', function (req, res) {  
  res.send('Guardado!');  
});
```

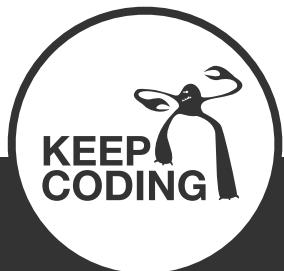


■ Orden de las rutas

El orden es importante!

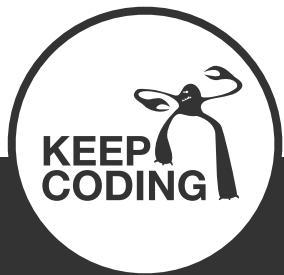
En el orden que carguemos nuestras rutas a express es el orden en que las interpretará.

Si ponemos los estáticos después de nuestras rutas podremos '*sobre-escribir*' un fichero estático con una ruta, por ejemplo para comprobar si el usuario tiene permisos para descargarlo.





Servir ficheros estáticos

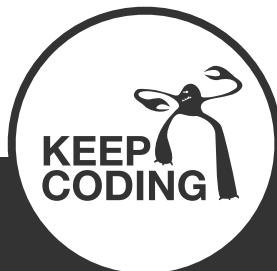


Servir ficheros estáticos

Servir estáticos como CSS, imágenes, ficheros javascript, etc, se especifica con un middleware llamado `express.static`

```
app.use( express.static( path.join(__dirname, 'public') ) );
```

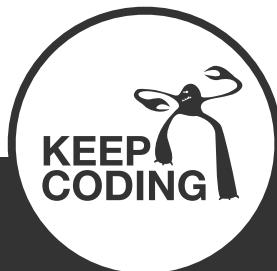
Con esto serviremos lo que haya en la carpeta `public` como estáticos de la raíz de la ruta.



Servir ficheros estáticos

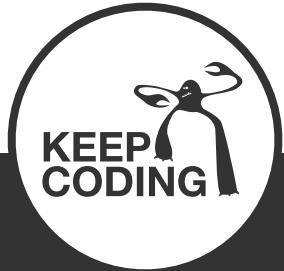
Si queremos añadir otras carpetas de estáticos tenemos que especificar en qué rutas colgarlos.

```
// la ruta virtual '/otros' servirá la carpeta '/otros'  
app.use('/otros', express.static(path.join(__dirname, 'otros')));
```





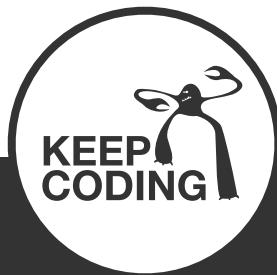
Template engines



■ Templates

Además de servir html estático podemos usar sistemas de plantillas.

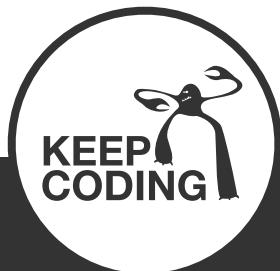
Express-generator por defecto monta Jade, podemos cambiarlo fácilmente, por ejemplo a EJS que es uno de los más usados.



■ Templates

Hay muchos sistemas de templates en Javascript, pero los que cumplen con el estándar de Express están listados en:

<https://www.npmjs.com/package/consolidate#supported-template-engines>

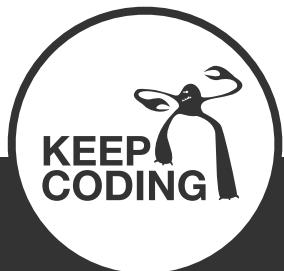


■ Templates

Para instalar un sistema de templates lo instalaremos con **npm install ejs -save** y nos aseguramos de que nuestra aplicación lo usa con un par de settings:

- **views**, el directorio donde estarán nuestras plantillas, por ejemplo:
`app.set('views', './views')`
- **view engine**, el template engine a usar, por ejemplo:
`app.set('view engine', 'jade')`

Express lo carga automáticamente y ya podremos usarlo.

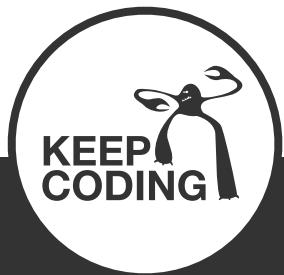


■ Templates - Jade

```
app.set('view engine', 'jade');
```

```
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    p esto es un párrafo
```

ejemplos/express/jade



■ Templates - Jade

```
<!DOCTYPE html>
<html>
  <head>
    <title>Express</title>
    <link rel="stylesheet" href="/stylesheets/style.css">
  </head>
  <body>
    <p>esto es un párrafo</p>
  </body>
</html>
```

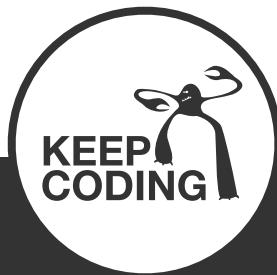
ejemplos/express/jade



■ Templates - EJS

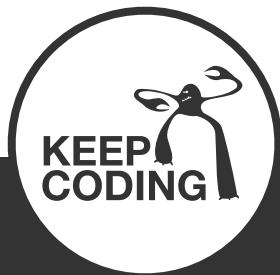
EJS añade su funcionalidad sobre HTML estándar.

Esto puede hacer que nos sea más fácil depurar errores o integrar y mantener una maquetación realizada por un maquetador especializado.



■ Templates

Para proporcionar variables a las vistas tenemos 3 opciones



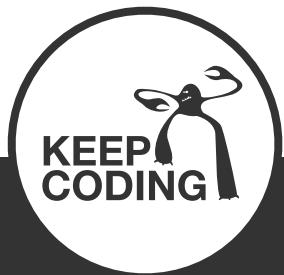
■ Templates

Variables globales a toda la app

```
app.locals.titulo = 'Anuncios';
```

```
...
```

```
res.render('index');
```

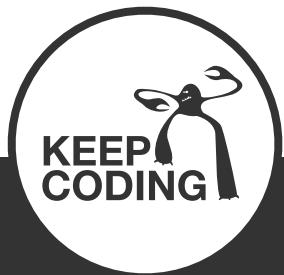


■ Templates

Variables locales a la respuesta

```
res.locals.titulo = 'Anuncios';
```

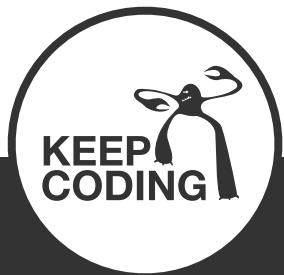
```
res.render('index');
```



■ Templates

Variables locales a la respuesta

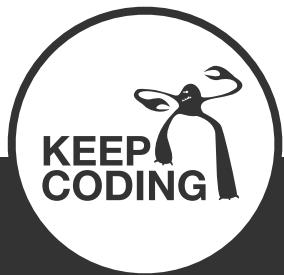
```
res.render('index', { titulo: 'Anuncios' });
```



■ Templates

En la vista tendremos disponibles esas variables.

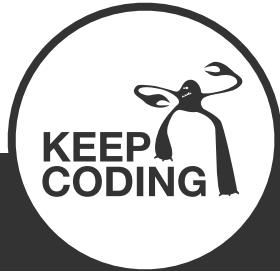
```
<title><%= titulo %></title>
```



■ Templates - sin escapar

El valor será escapado para evitar la inyección de código. Si queremos incluir html usaremos <%- %>

```
<p><%- sinEscapar %></p>
```



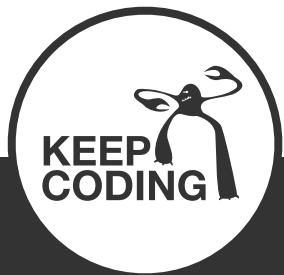
■ Templates - include

Podemos incluir el contenido de otras plantillas.

```
<% include otra/plantilla %>
```

views

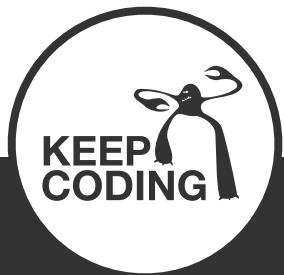
- index.js
- **otra**
- **plantilla.ejs**



■ Templates - condiciones

Sacar bloques de forma condicional es sencillo.

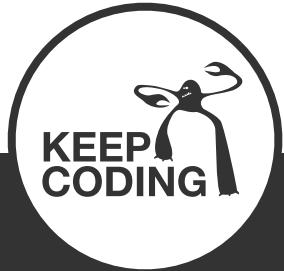
```
<% if (condicion.estado) { %>
    <p><%= condicion.segundo %> es par</p>
<% } else { %>
    <p><%= condicion.segundo %> es impar</p>
<% } %>
```



■ Templates - iterar

O por ejemplo iterar bucles.

```
<% users.forEach(function(user) { %>
  <p><%= user.name %></p>
<% } ) %>
```

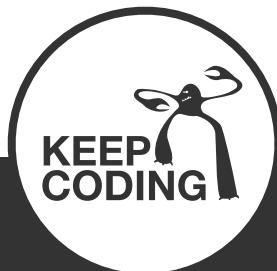


■ Templates - código

En resumen, entre los tags <% y %> (sin usar '=' o '-') podemos colocar cualquier estructura de código válida en Javascript.

Pero sin pasarnos. Esto es una vista, y meter código aquí significa que tenemos funcionalidad en las vistas.

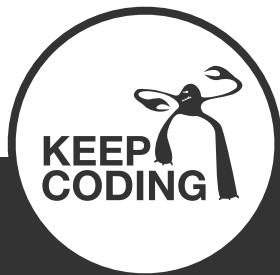
Es recomendable tener toda o la mayor parte de la funcionalidad en los modelos (o al menos en los controladores).



■ Templates - html

Si quisiéramos, podríamos tener las vistas con extensión .html

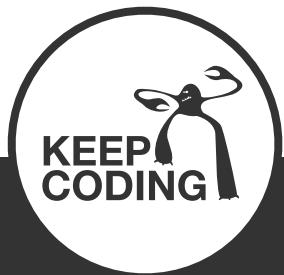
```
app.set('view engine', 'html');  
app.engine('html', require('ejs')).__express);
```



■ Templates

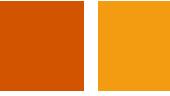
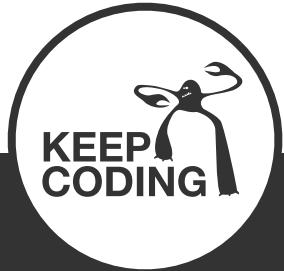
Podemos encontrar su documentación y más ejemplos en:

<https://github.com/mde/ejs>





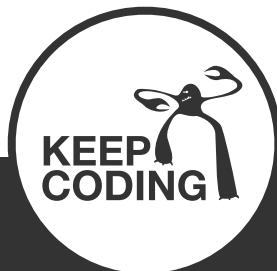
Recibiendo parámetros



■ Recibiendo parámetros

Habitualmente recibiremos parámetros en nuestros controladores de varias formas:

- En la ruta (/users/**5**)
- Con parámetros en query string (/users?**sort=name**)
- En el cuerpo de la petición (POST y PUT generalmente)
- También podemos recibirlas en la cabecera, pero esta zona solemos dejarla para información de contexto, como autenticación, formatos, etc.



■ Recibiendo parámetros - en la ruta

Lo definimos en el argumento PATH de la ruta

```
router.put('/ruta/:id', function(req, res) {  
  console.log('params', req.params);  
  var id = req.params.id;  
});
```

PUT http://localhost:3000/ruta/55

Podemos combinarlo con los otros



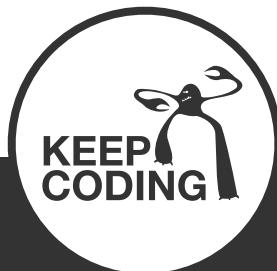
■ Recibiendo parámetros

En el paso por ruta podemos usar expresiones regulares en los parámetros, incluir varios o hacerlos opcionales.

```
router.put('/ruta/:id?', function); // parámetro opcional  
// params { id: 'dato'}
```

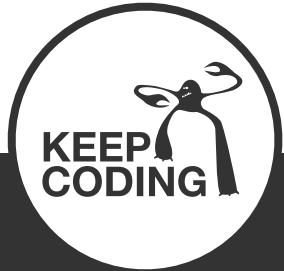
```
router.put('/ruta/:id([0-9]+)', function); // parámetro con regexp  
// params { id: '26'}
```

```
router.put('/ruta/:id([0-9]+)/piso/:piso(A|B|C)', function); // varios  
// params { id: '26', piso: 'A' }
```



■ Recibiendo parámetros - Regexp

<https://regexone.com/>



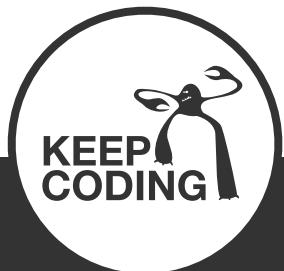
■ Recibiendo parámetros - en query string

Lo definimos en la query string

```
router.put('/ruta', function(req, res) {  
  console.log('query-string', req.query);  
  var id = req.query.id;  
});
```

PUT http://localhost:3000/ruta?id=66

Podemos combinarlo con los otros

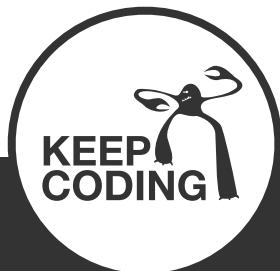


■ Recibiendo parámetros - en el body

Los recibimos en req.body. Esta forma no la podemos usar en GET ya que no usa body.

```
router.put('/ruta', function(req, res) {  
  console.log('body', req.body); // body  
  var nombre = req.body.nombre;  
});
```

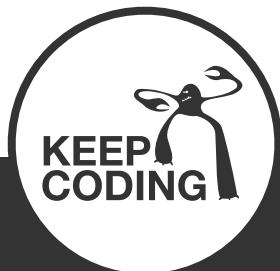
**PUT http://localhost:3000/ruta
{nombre: 'Pepe'}**



■ Recibiendo parámetros - en el body

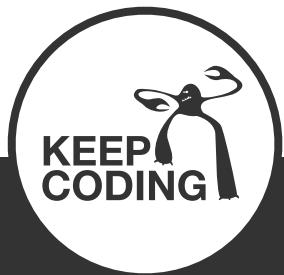
Formatos de codificación del cuerpo de la petición (body).

- Content-Type: "application/x-www-form-urlencoded" **(default)**
- Content-Type: "multipart/form-data"
- Raw
 - Content-Type: "application/json"
 - Content-Type: "text/plain"
 - Content-Type: "image/jpeg"
 - ...





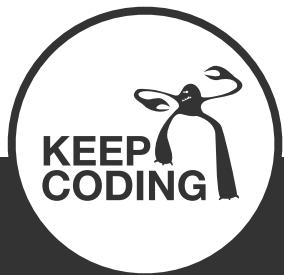
■ Validaciones



■ Validaciones

<https://github.com/ctavan/express-validator>

`npm install express-validator`



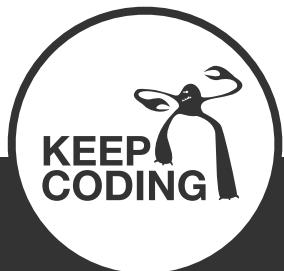
■ Validaciones

En el middleware:

```
const { query, validationResult } = require('express-validator');

...
router.get('/', [
  query('age').isNumeric().withMessage('must be numeric')
], (req, res, next) => {
  validationResult(req).throw(); // excepción si hay errores de validación

  // todo validado!
  res.send('ok');
});
```



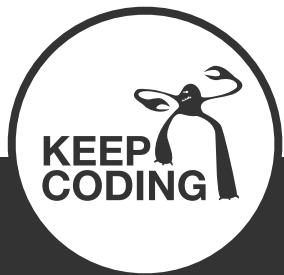
■ Validaciones

En app.js:

```
// error handler
app.use(function(err, req, res, next) {

  if (err.array) { // validation error
    err.status = 422;
    const errInfo = err.array({ onlyFirstError: true })[0];
    err.message = `Not valid - ${errInfo.param} ${errInfo.msg}`;
  }

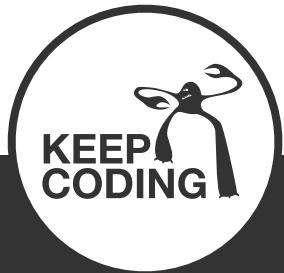
  res.status(err.status || 500);
  ...
})
```



■ Validaciones

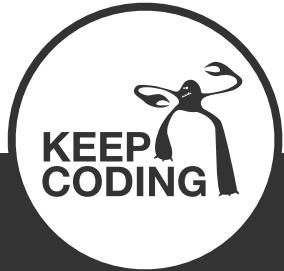
Express.js middleware for validating requests against JSON schema:

<https://github.com/vacekj/express-json-validator-middleware>





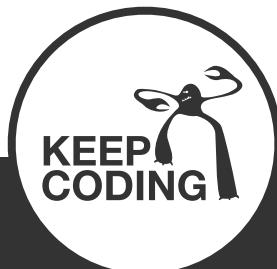
Métodos de respuesta



Métodos de respuesta

res.download()	Prompt a file to be downloaded.
res.end()	End the response process.
res.json()	Send a JSON response.
res.jsonp()	Send a JSON response with JSONP support.
res.redirect()	Redirect a request.
res.render()	Render a view template.
res.send()	Send a response of various types.
res.sendFile	Send a file as an octet stream.
res.sendStatus()	Set the response status code and send its string representation as the response body.

Podemos ver su [documentación](#). Veamos los más usados...



Métodos de respuesta - send

Para responder a una petición podemos usar el método genérico res.send().

El cuerpo de la respuesta puede ser un buffer, un string, un objeto o un array.

```
res.send(new Buffer('whoop'));  
res.send({ some: 'json' } );  
res.send('<p>some html</p>');  
res.status(404).send('Sorry, we cannot find that!');  
res.status(500).send({ error: 'something blew up' });
```

Express detecta el tipo de contenido y pone el header Content-Type adecuado.

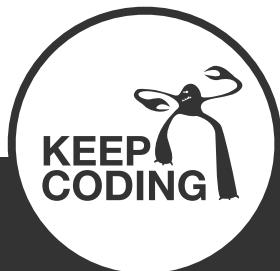
Si es un array o un objeto devuelve su representación en JSON



Métodos de respuesta - json

Podemos usar res.json, que ajusta un posible null o undefined para que salga bien en JSON

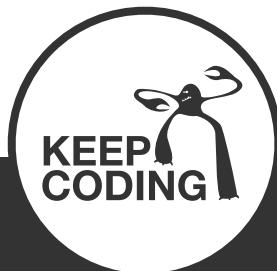
```
res.json(null)
res.json({ user: 'tobi' })
res.status(500).json({ error: 'message' })
```



■ Métodos de respuesta - download

Transfiere el fichero especificado como un attachment. El browser debe solicitar al usuario que guarde el fichero.

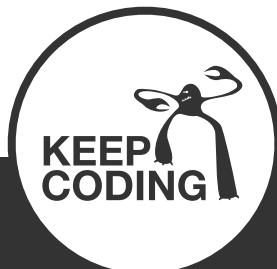
```
// el nombre del fichero es opcional  
res.download('/report-12345.pdf', 'report.pdf');
```



Métodos de respuesta - redirect

Devuelve una redirección con el status code 302 por defecto (found).

```
res.redirect('/foo/bar'); // relativa al root host name  
res.redirect('http://example.com'); // absoluta  
res.redirect(301, 'http://example.com'); // con status  
res.redirect('../login'); // relativa al path actual  
res.redirect('back'); // vuelve al referer
```

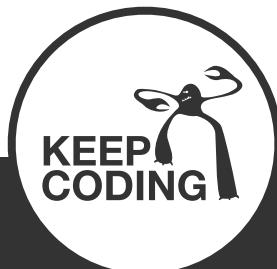


Métodos de respuesta - render

Renderiza una vista y envia el HTML resultante. Acepta un parámetro opcional 'locals' para dar variables locales a la vista.

```
// render de la vista index  
res.render('index');
```

```
// render de la vista user con el objeto locals  
res.render('user', { name: 'Tobi' });
```

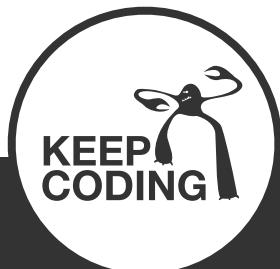


Métodos de respuesta -.sendFile

Envía un fichero como si fuera un estático.

Además de la ruta del fichero acepta un objeto de opciones y un callback para comprobar el resultado de la transmisión.

```
var options = {  
  headers: {  
    'x-timestamp': Date.now(),  
    'x-sent': true  
  }  
};  
  
res.sendFile(fileName, options);
```



Middlewares de terceros

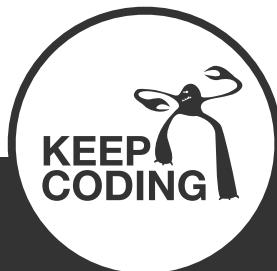
Podemos instalarlos con npm y cargarlos como los anteriores.

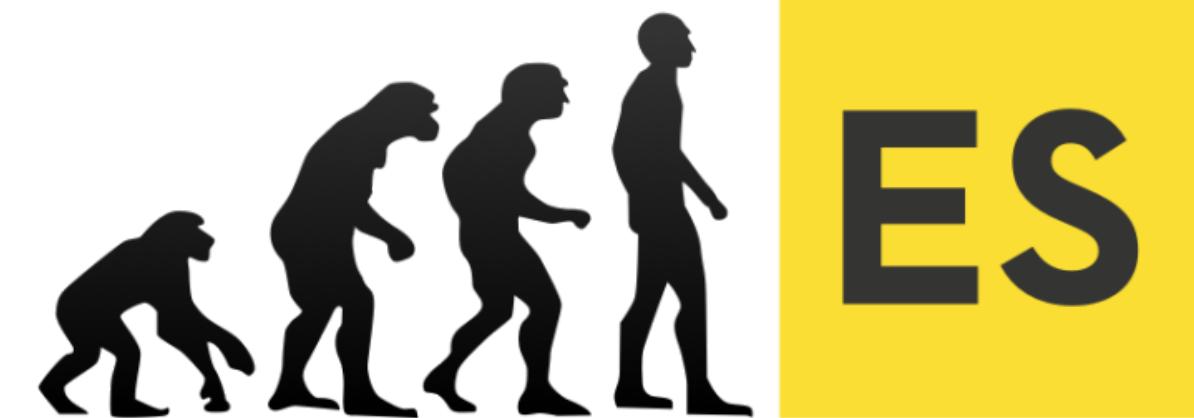
```
$ npm install cookie-parser
```

```
var cookieParser = require('cookie-parser');
```

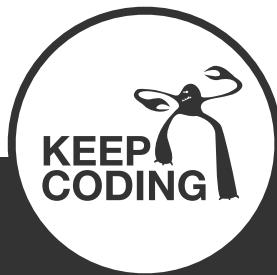
```
// load the cookie parsing middleware
app.use(cookieParser());
```

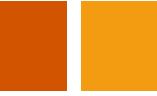
Hay una lista de los más usados en <http://expressjs.com/resources/middleware.html>



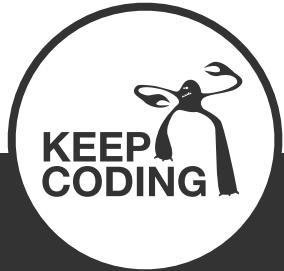


■ Evolución de la asincronía



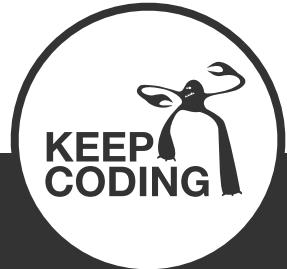


■ Promesas





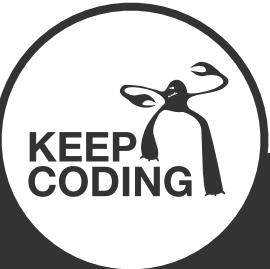
Callback Hell



```
// esto no es vida
carga(function(err, valor0) {
  valida(valor0, function(err, valor1) {
    procesa(valor1, function(err, valor2) {
      guarda(valor2, function(err, valor3) {
        comprueba(valor3, function(err, valor4) {
          junta(valor4, function(err, valor5) {
            limpia(valor5, function(err, valor6) {
              avisa(valor6, function(err, valor6) {
                res.json({piramide: true});
              } );
            } );
          } );
        } );
      } );
    } );
  } );
});
```



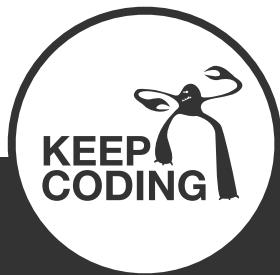
```
174
175     async.series([
176         function(callback) {
177             if (todos == "true") {
178                 models.PromocionesBase.contarTodos(todos, function(err, resultado) {
179                     flag = 1;
180                     resultado = result;
181                     procesarSegmento(segmento_id, function(segmento) {
182                         var id = parseInt(segmento_id);
183                         models.segmentacion.find(id).success(function(segmento) {
184                             if (segmento) {
185                                 var obj = {
186                                     cp: segmento.cp,
187                                     edad: segmento.edad,
188                                     sexo: segmento.sexo,
189                                     hijo: segmento.hijo,
190                                     favoritos: segmento.favoritos
191                                 };
192                                 models.PromocionesBase.contarSimilares(segmento_id, function(err, resultado) {
193                                     if (err) return callback(err);
194                                     return callback(null, resultado);
195                                 });
196                             }
197                         });
198                     });
199                 });
200             },
201         },
202     ],
203     function(error, results) {
204         if (error) return callback(error);
205         else return callback(null, results);
206     });
207 }
```



Promesas

Una promesa es un objeto que representa una operación que aún no se ha completado, pero que se completará más adelante.

Antes de ES2015 podíamos usarlas con librerías, pero estas librerías tienen ligeras (o no tan ligeras) diferencias entre ellas. Ahora ya forman parte del estándar y el lenguaje y no necesitamos estas librerías.



Promesas

Tiene tres estados posibles (<https://promisesaplus.com/>)

1. Pending
2. Fullfilled(value)
3. Rejected(reason)

2.1. Promise States

A promise must be in one of three states: pending, fulfilled, or rejected.

2.1.1. When pending, a promise:

 2.1.1.1. may transition to either the fulfilled or rejected state.

2.1.2. When fulfilled, a promise:

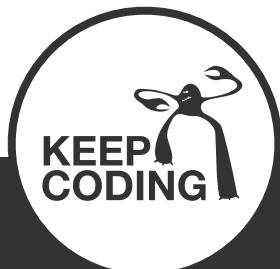
 2.1.2.1. must not transition to any other state.

 2.1.2.2. must have a value, which must not change.

2.1.3. When rejected, a promise:

 2.1.3.1. must not transition to any other state.

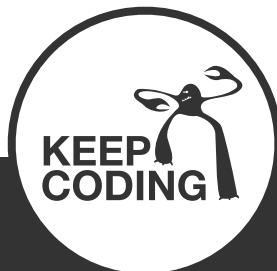
 2.1.3.2. must have a reason, which must not change.



Promesas

Cuando una promesa está en uno de los dos estados fulfilled o rejected se le llama settled.

Si la promesa se hubiera cumplimentado (fulfilled) o rechazado (rejected) antes de asignarle un then o catch, cuando se le asignen serán llamados con el resultado o el error.



■ Promesas

Como se hace

```
var promesa = new Promise(function(resolve, reject) {  
    // llamo a resolve con el resultado  
    // o llamo a reject con el error  
});
```

```
promesa.then( function(resultado) {  
}).catch( function(error) {  
});
```

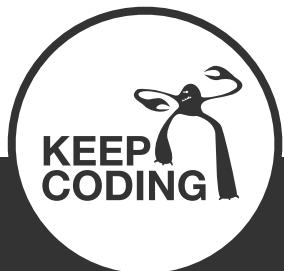


■ Promesas

```
promesa.then( function(resultado) {  
} ).catch( function(error) {  
} );
```

Es simplemente azúcar sintáctico para la forma:

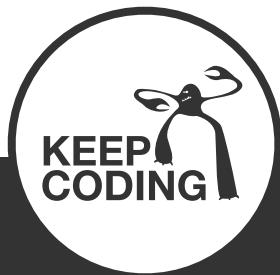
```
promesa.then(  
  function(resultado) { },  
  function(error) { }  
);
```





■ Ejercicio

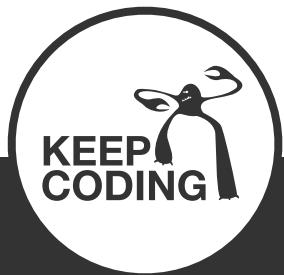
Hacer el comando sleep(milisegundos)



■ Promesas

Podemos encadenar promesas.

```
promesa1
  .then( () => promesa2)
  .then( () => promesa3)
  .then( function(data) { // final
    console.log(data); })
  .catch( function(err) {
    console.log('ERROR', err);
});
```



■ Promesas

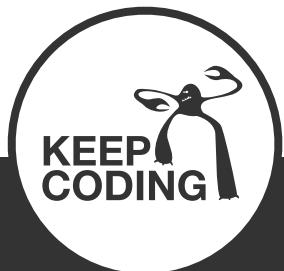
```
var ingredientes = [ 'sal', 'pimienta', 'conejo', 'gambas' ];  
  
// echar() recibe un string y retorna una promesa  
var promisedTexts = ingredientes.map(echar);  
  
Promise.all(promisedTexts)  
  .then(function (texts) {  
    console.log(texts); // han acabado todas  
  })  
  .catch(function (reason) {  
    // llegaremos aqui con el primero que falle  
  });
```



Promesas

Si `Promise.all` esperaba a que estuvieran todas cumplidas, `Promise.race` lo hace cuando cumpla la primera, devolviendo su resultado.

```
Promise.race([p1, p2, p3])
  .then(function (textoDelMasRapido) {
    // llegaremos aqui con el primero que acabe
  })
  .catch(function (reason) {
    // llegaremos aqui con el primero que falle
});
```



Promesas

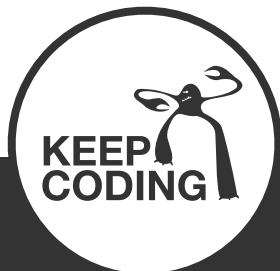
El objeto Promise tiene también un par de métodos estáticos que pueden ser útiles:

`Promise.resolve(valor);`

Devuelve una promesa resuelta con el valor proporcionado.

`Promise.reject(razon);`

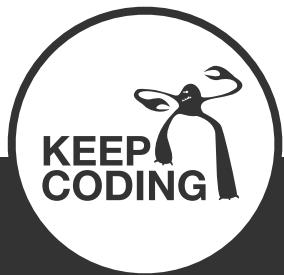
Devuelve una promesa resuelta con la razón suministrada. La razón debería ser un error (generalmente una instancia de objeto Error).



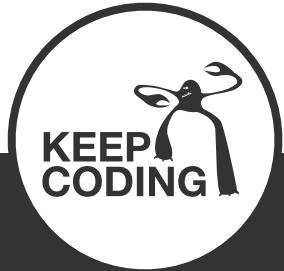
■ Promesas

```
Promise.resolve("bien!").then(function(value) {  
  console.log(value); // "Prueba resolve"  
, function(reason) {  
  // not called  
});
```

```
Promise.reject(new Error("chungo...")).then(function(value) {  
  // not called  
, function(error) {  
  console.log(error); // Stacktrace  
});
```

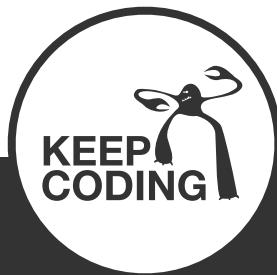


■ async / await



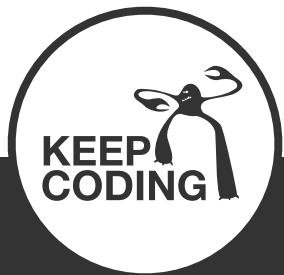
■ ECMAScript 2015 (ES6) - `async` / `await`

async hace que una función devuelva una promesa.



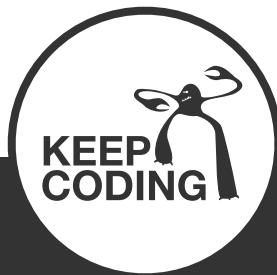
■ ECMAScript 2015 (ES6) - async / await

```
async function saluda() {  
    return 'hola';  
}  
  
console.log(saluda()); // Promise { 'hola' }  
  
saluda().then(res => console.log(res)); // hola
```



■ ECMAScript 2015 (ES6) - `async / await`

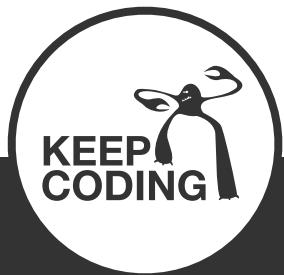
await consume una promesa



■ ECMAScript 2015 (ES6) - async / await

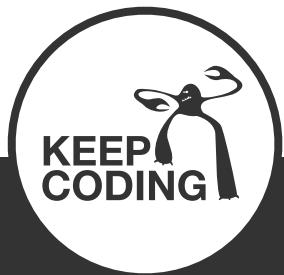
```
async function saluda() {  
  const nombre = await row.findName();  
  return nombre;  
}
```

ejemplos/async-await



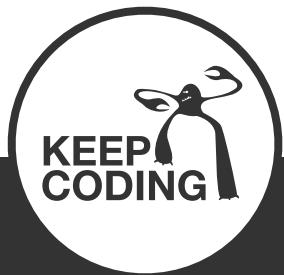
■ ECMAScript 2015 (ES6) - async / await

```
async function bucleAsincronoEnSerie() {  
    for (let i = 0; i < 5; i++) {  
        const info = await row.findNext();  
        console.log(info.name);  
    }  
}
```



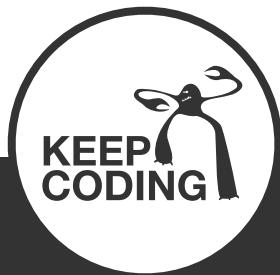
■ ECMAScript 2015 (ES6) - async / await

```
async function asincronoEnParalelo() {  
    const prom1 = row.findNext();  
    const prom2 = row.findNext();  
    const prom3 = row.findNext();  
    const list = await Promise.all([prom1, prom2, prom3]);  
}
```



■ ECMAScript 2015 (ES6) - `async / await`

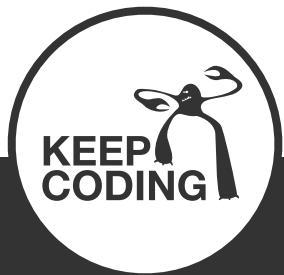
Como usarlo en Express



■ ECMAScript 2015 (ES6) - async await + Express

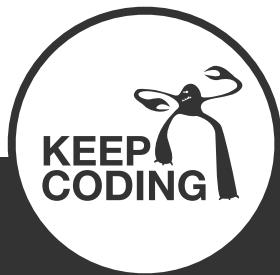
```
router.get(async (req, res, next) => {  
  const list = await Orders.find(...);  
  // si falla quién gestiona el error?  
})
```

* una pista... nadie



■ ECMAScript 2015 (ES6) - async await + Express

```
router.get(async (req, res, next) => {
  try {
    const list = await Orders.find(...);
  } catch (err) {
    next(err);
  }
})
```

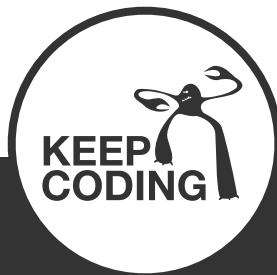


■ ECMAScript 2015 (ES6) - async await + Express

OPCIONAL

<https://github.com/Abazhenov/express-async-handler>

```
$ npm install express-async-handler
```

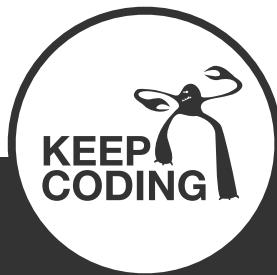


■ ECMAScript 2015 (ES6) - async await + Express

<https://github.com/Abazhenov/express-async-handler>

```
const asyncHandler = require('express-async-handler')

router.get(asyncHandler(async (req, res, next) => {
  const list = await Orders.find(...);
}))
```

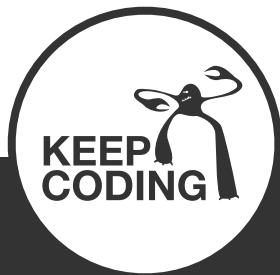


■ ECMAScript 2015 (ES6) - async await + Express

<https://github.com/davidbanham/express-async-errors>

```
const express = require('express');
require('express-async-errors');

app.get('/users', async (req, res) => {
  const users = await User.findAll();
  res.send(users);
});
```



Bases de datos



Javier Miguel

@JavierMiguelG



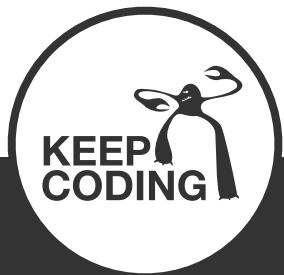
PostgreSQL



mongoDB



■ Bases de datos



Bases de datos

Node.js, a través de módulos de terceros, se puede conectar casi con cualquier base de datos del mercado.

Basta con cargar el driver (módulo) adecuado y establecer la conexión.

```
$ npm install mysql
```

```
$ npm install pg
```

```
$ npm install mongodb
```





SQL



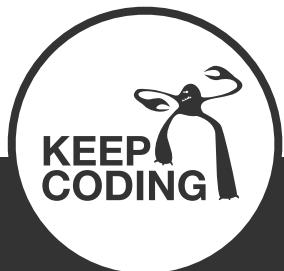
Bases de datos - MySQL (docker)

Desplegar BD MySQL local con Docker:

```
docker run -d --name mariadb-kc \
-v $(PWD)/data:/var/lib/mysql \
-e MARIADB_ROOT_PASSWORD=my-secret-pw \
-p 3306:3306 \
mariadb:latest
```

Abrir shell al contenedor:

```
docker exec -it mariadb-kc bash
mariadb -p
```



Bases de datos - MySQL

Iniciar BD MySQL:

```
CREATE DATABASE cursonode;
```

```
SHOW DATABASES;
```

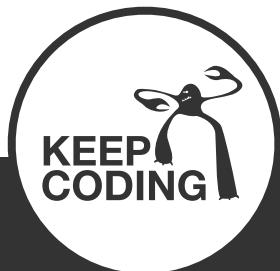
```
use cursonode;
```

```
CREATE TABLE agentes (agenteid int NOT NULL AUTO_INCREMENT, name varchar(255) NOT NULL,  
age int, PRIMARY KEY (agenteid) );
```

```
INSERT INTO agentes (name, age) VALUES ('Brown', 21);
```

```
INSERT INTO agentes (name, age) VALUES ('Smith', 33);
```

```
INSERT INTO agentes (name, age) VALUES ('Jones', 45);
```



KEEP
CODING

Bases de datos - MySQL

```
const mysql = require('mysql2/promise');

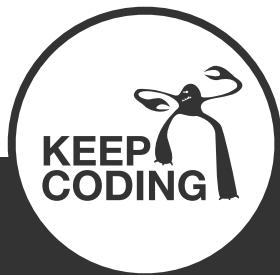
const CONN_STR = 'mysql://root:my-secret-pw@localhost:3306/cursoronode';

async function main(params) {

  const connection = await mysql.createConnection(CONN_STR);

  const [rows, _] = await connection.execute('select * from agentes');

  console.log(rows);
}
```

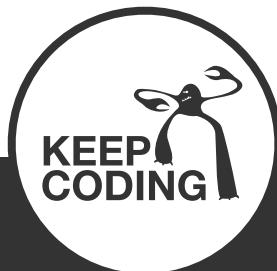


Bases de datos

Para refrescar conceptos de SQL:

Mastering Relational Database Design

<https://dev.to/louaiboumediene/mastering-relational-database-design-a-comprehensive-guide-3jh8>



Bases de datos - SQL ORMs

Un ORM (Object Relational Mapping) se encarga principalmente de:

- Convertir objetos en consultas SQL para que puedan ser persistidos en una base de datos relacional.
- Traducir los resultados de una consulta SQL y generar objetos.

Esto nos resultará útil si el diseño de nuestra aplicación es orientado a objetos (OOP).



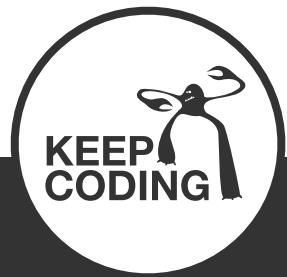
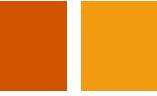
Bases de datos - SQL ORMs

ORMs usados para bases de datos SQL:

- TypeORM <https://github.com/typeorm/typeorm>
- Prisma <https://www.prisma.io/>
- Sequelize <http://docssequelizejs.com/en/latest/>
- Mikro-ORM <https://github.com/mikro-orm/mikro-orm>

Otras alternativas son Knex (query builder) y Bookshelf.





Bases de datos - MongoDB

MongoDB es una base de datos no relacional sin esquemas, esto significa principalmente que:

- No tenemos JOIN, tendremos que hacerlo nosotros
- Cada registro podría tener una estructura distinta
- Mínimo soporte a transacciones

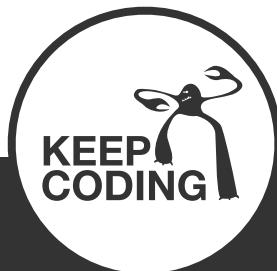
A la hora de decidir qué base de datos usar para una aplicación debemos pensar cómo vamos a organizar los datos para saber si nos conviene usar una base de datos relacional o no relacional.



Bases de datos - MongoDB

Usar una base de datos como MongoDB puede darnos más rendimiento principalmente por alguna de estas razones:

- No tiene que gestionar transacciones
- No tiene que gestionar relaciones
- No es necesario convertir objetos a tablas y tablas a objetos (Object-relation Impedance Mismatch)

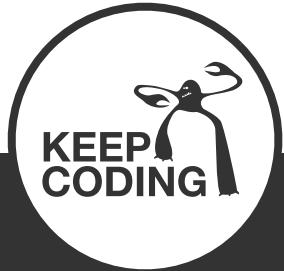


Bases de datos - MongoDB

```
$ npm install mongodb

const { MongoClient } = require('mongodb');

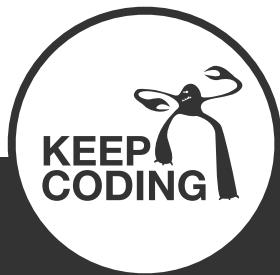
MongoClient.connect('mongodb://localhost', function(err, client) {
  if (err) throw err;
  const db = client.db('cursoronode');
  db.collection('agentes').find({}).toArray(function(err, docs) {
    if (err) throw err;
    console.dir(docs);
    client.close();
  });
});
```



Bases de datos - MongoDB shell basics

Para acceder a la shell usaremos:

```
~/master/cursoronode/mongodb-server/bin/mongo  
MongoDB shell version: 3.0.4  
connecting to: test  
>
```

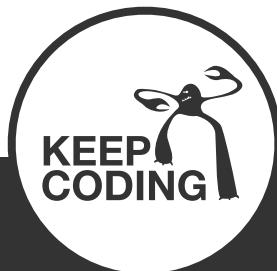


Bases de datos - MongoDB shell basics

```
show dbs
use <dbname>
show collections
show users
db.agentes.find().pretty()
db.agentes.insertOne({name: "Brown", age: 37})
db.agentes.deleteOne({_id: ObjectId("55ead88991233838648570dd")})
db.agentes.updateOne({_id: ObjectId("55eadb4191233838648570de")}, {$set: {age: 38}})
----- cuidado con el $set! -----
```

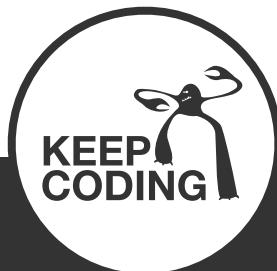
```
db.coleccion.drop()
db.agentes.createIndex({name:1, age:-1})
db.agentes.getIndexes()
```

Mas operaciones en la referencia rápida a la shell de MongoDB



Bases de datos - MongoDB queries

```
db.agentes.find({ name : 'Smith' })
db.agentes.find({ _id : ObjectId("55eadb4191233838648570de") })
db.agentes.find({ age: { $gt: 30 } }) // $lt, $gte, $lte, ...
db.agentes.find({ age: { $gt: 30, $lt: 40 } });
db.agentes.find({ name: { $in: [ 'Jones', 'Brown' ] } }) // $nin
db.agentes.find({ name: 'Smith', $or: [
  { age: { $lt: 30 } },
  { age: 43 } // 'Smith' and ( age < 30 or age = 43 )
] })
```

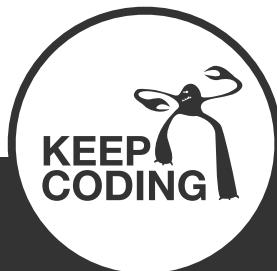


Bases de datos - MongoDB queries

```
// subdocuments  
db.agentes.find({ 'producer.company': 'ACME' })  
  
// arrays  
db.agentes.find({ bytes: [ 5, 8, 9 ] }) // array exact  
db.agentes.find({ bytes: 5 }) // array contain  
db.agentes.find({ 'bytes.0': 5 }) // array position
```

[http://docs.mongodb.org/manual/reference/method/db.collection.find](http://docs.mongodb.org/manual/reference/method/db.collection.find/#db.collection.find)

<http://docs.mongodb.org/manual/tutorial/query-documents/>



Bases de datos - MongoDB queries

Ordenar:

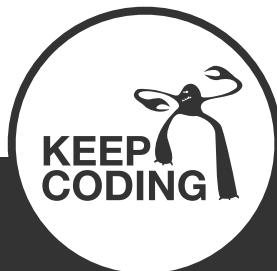
```
db.agentes.find().sort({age: -1})
```

Descartar resultados:

```
db.agentes.find().skip(1).limit(1)  
db.agentes.findOne({name: 'Brown'}) // igual a limit(1)
```

Contar:

```
db.agentes.find().count() // db.agentes.count()
```



Bases de datos - MongoDB queries

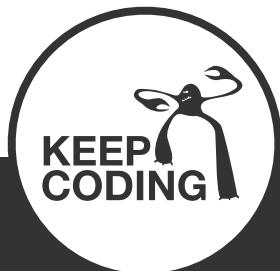
Full Text Search

Crear índice por los campos de texto involucrados:

```
db.agentes.createIndex({titulo: 'text', subtitle: 'text'});
```

Para hacer la búsqueda usar:

```
db.agentes.find({$text:{$search:'smith jones'}});
```



Bases de datos - MongoDB queries

Full Text Search

Frase exacta:

```
db.agentes.find({$text:{$search:'smith jones "el elegido"'}});
```

Excluir un término:

```
db.agentes.find({$text:{$search:'smith jones -mister'}});
```

Más info:

<https://docs.mongodb.com/v3.2/text-search/>

<https://docs.mongodb.com/v3.2/tutorial/specify-language-for-text-index/>

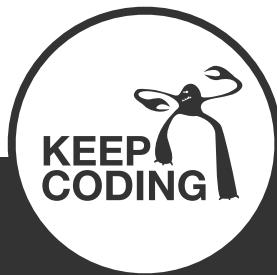


Bases de datos - MongoDB Geo

<https://docs.mongodb.com/manual/applications/geospatial-indexes/>

```
db.productos.createIndex({ubicacion: '2dsphere'})
```

```
db.productos.insert({
  "ubicacion": {
    "coordinates": [ -73.856077, 40.848447 ],
    "type": "Point"
  }
})
```



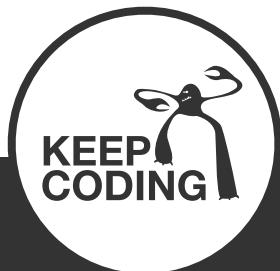
Bases de datos - MongoDB Geo

<https://docs.mongodb.com/manual/applications/geospatial-indexes/>

```
db.productos.createIndex( {ubicacion: '2dsphere' } )
```

```
db.productos.insert( {
  "ubicacion": {
    "coordinates": [ -73.856077, 40.848447 ],
    "type": "Point"
  }
})
```

Es necesario crear un índice geoespacial



KEEP
CODING

Bases de datos - MongoDB Geo

<https://docs.mongodb.com/manual/applications/geospatial-indexes/>

```
db.productos.createIndex({ubicacion: '2dsphere'})
```

```
db.productos.insert({
  "ubicacion": {
    "coordinates": [-73.856077, 40.848447],
    "type": "Point"
  }
})
```



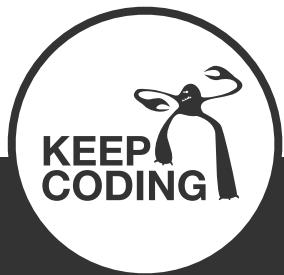
El orden es longitud, latitud



Bases de datos - MongoDB Geo

```
const meters = parseFloat(req.params.meters); // 105 * 1000
const longitude = parseFloat(req.params.lng); // -73
const latitude = parseFloat(req.params.lat); // 40

db.productos.find({
  ubicacion: {
    $nearSphere: {
      $geometry: {
        type: 'Point',
        coordinates: [longitude, latitude]
      },
      $maxDistance: meters
    }
  }
})
```

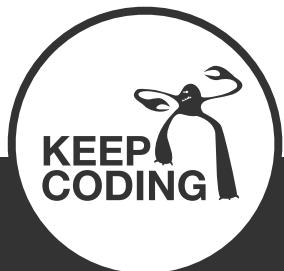


Bases de datos - MongoDB transacción

`findAndModify` es una operación atómica.

```
db.agentes.findAndModify( {  
    query: { name: "Brown" },  
    update: { $inc: { age: 1 } }  
})
```

Lo busca y si lo encuentra lo modifica, no permitiendo que otro lo cambie antes de modificarlo.

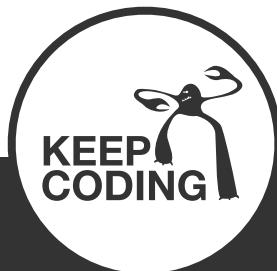


Bases de datos - MongoDB transacción

A partir de la versión 4 hay mejor soporte a transacciones (distributed & multi-document).

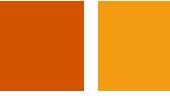
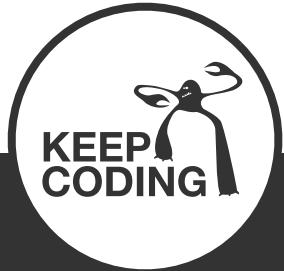
<https://www.mongodb.com/docs/manual/core/transactions/>

Paper detallado: https://webassets.mongodb.com/MongoDB_Multi_Doc_Transactions.pdf





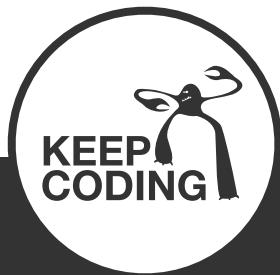
mongoose



Mongoose

Mongoose es una herramienta que nos permite persistir objetos en MongoDB, recuperarlos y mantener esquemas de estos fácilmente.

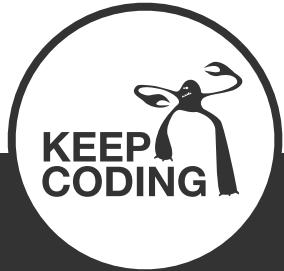
Este tipo de herramientas suelen denominarse ODM (Object Document Mapper).



Mongoose

Instalación como siempre:

```
npm install mongoose
```



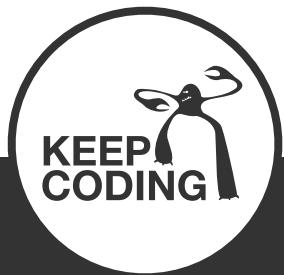
Mongoose

Conectar a la base de datos:

```
var mongoose = require('mongoose');
var conn = mongoose.connection;

conn.on('error', (err) =>
  console.error('mongodb connection error', err));
conn.once('open', () =>
  console.info('Connected to mongodb.'));

mongoose.connect('mongodb://localhost/diccionario');
```



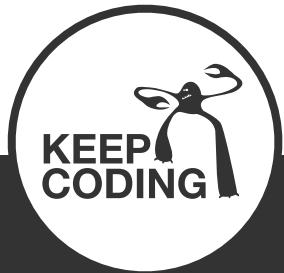
Mongoose

Crear un modelo:

```
var mongoose = require('mongoose');

var agenteSchema = mongoose.Schema({
  name: String,
  age: Number
});

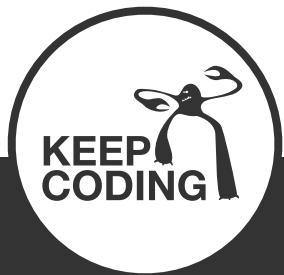
mongoose.model('Agente', agenteSchema);
```



Mongoose

Guardar un registro:

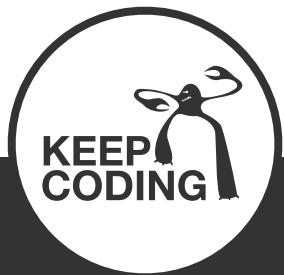
```
var agente = new Agente({name: 'Smith', age: 43});  
  
agente.save(function (err, agenteCreado) {  
  if (err) throw err;  
  console.log('Agente ' + agenteCreado.name + ' creado');  
});
```



Mongoose

Eliminar registros:

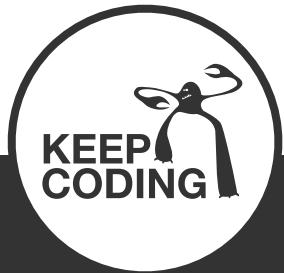
```
Agente.deleteMany( { [filters] } , function(err) {  
    if (err) return cb(err);  
    cb(null);  
} );
```



Mongoose

Crear un método estático a un modelo:

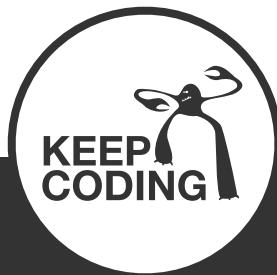
```
agenteSchema.statics.deleteAll = function(cb) {  
    Agente.remove({}, function(err) {  
        if (err) return cb(err);  
        cb(null);  
    });  
};
```



Mongoose

Crear un método de instancia a un modelo:

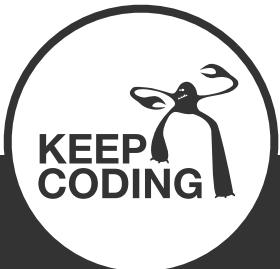
```
agenteSchema.methods.findSimilarAges = function (cb) {  
  return this.model('Agente').find({ age: this.age }, cb);  
}
```



Mongoose

Listando registros:

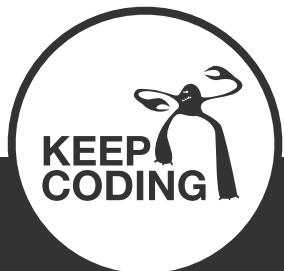
```
agenteSchema.statics.list = function(cb) {  
  var query = Agente.find({});  
  query.sort('name');  
  query.skip(500);  
  query.limit(100);  
  query.select('name age');  
  return query.exec(function(err, rows) {  
    if (err) { return cb(err); }  
    return cb(null, rows);  
  });  
};
```



Mongoose

Crear un modelo con datos geoespaciales:

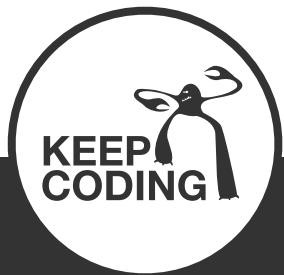
```
var productoSchema = mongoose.Schema( {  
    name: String,  
    location: {  
        type: { type: String},  
        coordinates: [ Number ]  
    }  
} );  
  
schema.index({location: '2dsphere'});  
  
mongoose.model('Producto', productoSchema);
```



Mongoose

Buscar por cercanía:

```
Product.find({ location: {  
    $nearSphere: {  
        $geometry: {type: 'Point', coordinates: [longitude, latitude]},  
        $maxDistance: meters  
    }  
} } );
```





Autenticación



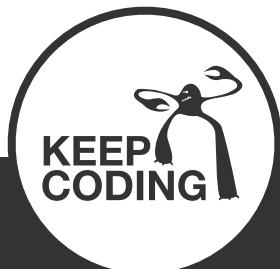
■ Autenticación HTTP

El protocolo HTTP es un protocolo sin estado: cada petición del navegador a un servidor es independiente de las anteriores.

Sin embargo en las aplicaciones web a veces necesitamos mantener el estado para poder relacionar peticiones.

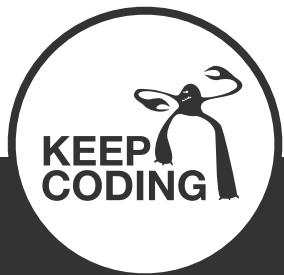
Ejemplo:

- Petición 1: el usuario añade un producto al carrito de compra
- Petición 2: el usuario va a pagar en el proceso de checkout





HTTP Basic Auth

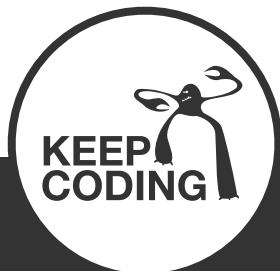


■ Autenticación - basic auth

Basic HTTP Auth es un mecanismo sencillo y muy usado en APIs.

```
npm install basic-auth
```

Implementaríamos un middleware que comprobara las credenciales.



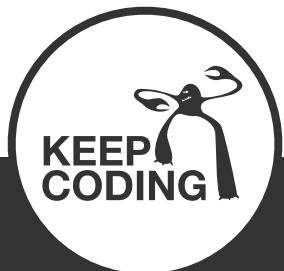
■ Autenticación - basic auth

Ventajas:

- Lo soportan todos los navegadores
- Una persona puede realizar la autenticación manualmente de forma sencilla, por ejemplo para explorar el API

Desventajas:

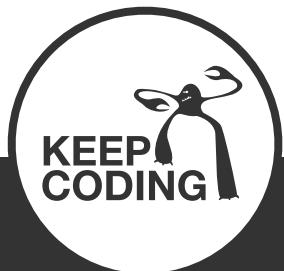
- Hay métodos más seguros



■ Autenticación - basic auth

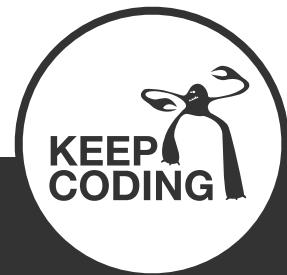
```
npm install basic-auth
```

```
router.use(function(req, res, next) {
  var user = basicAuth(req);
  if (!user || user.name !== 'user' || user.pass !== 'pass') {
    res.set('WWW-Authenticate', 'Basic realm=Authorization
Required');
    return res.sendStatus(401);
  }
  next();
});
```

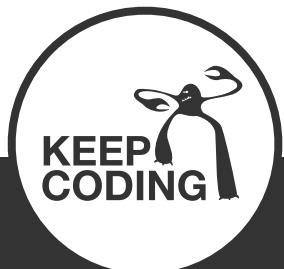
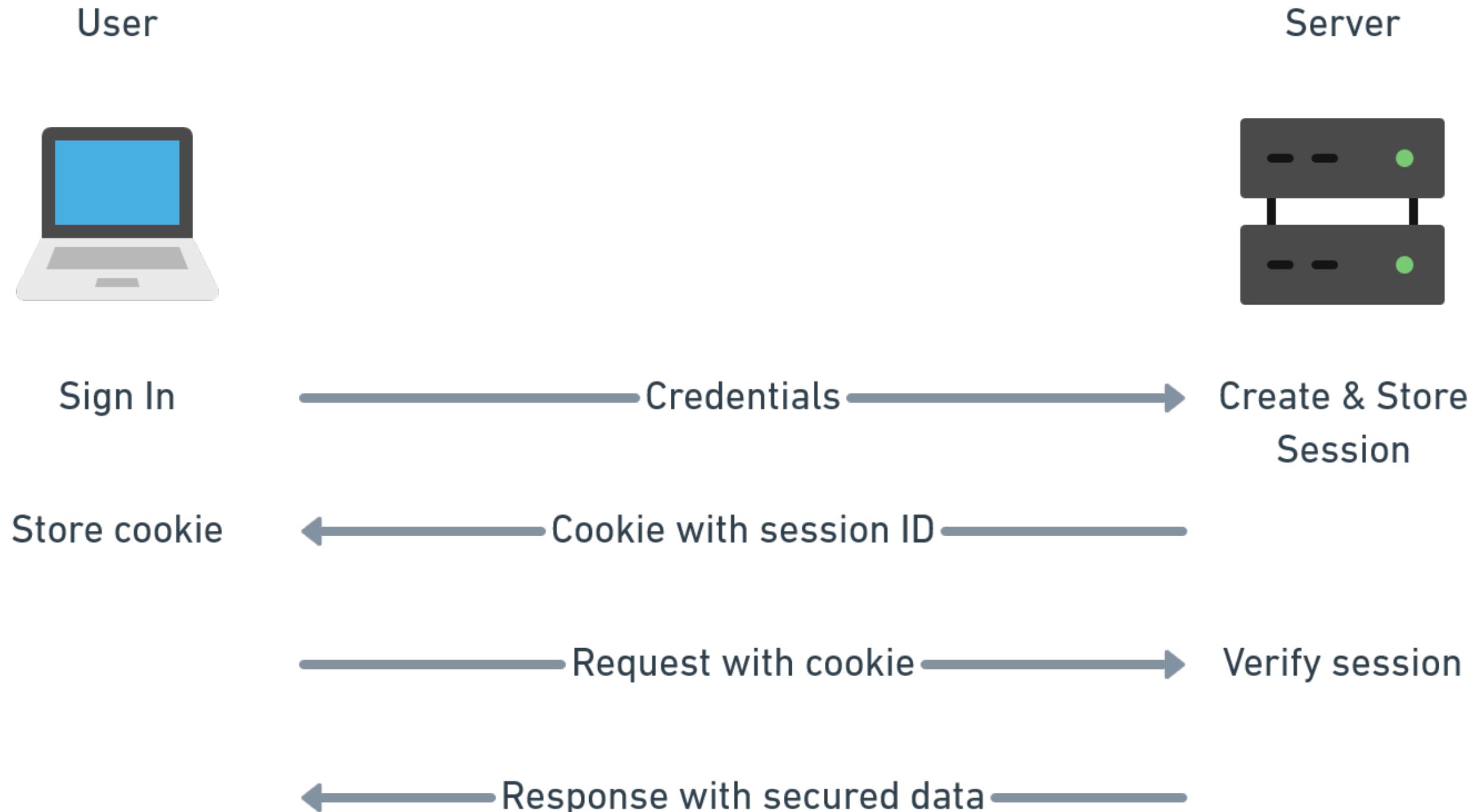




Sesión



■ Autenticación por sesión



■ Autenticación por sesión

La autenticación por sesión se basa en el uso de **cookies**.

Las cookies son **bloques de información** enviados en las cabeceras HTTP en las respuestas de los servidores a los navegadores.

Los **navegadores** almacenan estos bloques de información y los **envían automáticamente a los servidores** en las siguientes peticiones.



■ Autenticación por sesión

Cliente

Servidor

Almacén de sesiones

POST /add/product/to/cart/

{ id: **SESSION_A**, cart_products: [...] }

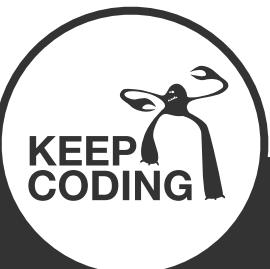
Set-Cookie: sessionid=**SESSION_A**

GET /products/in/cart/
Cookie: sessionid=**SESSION_A**

{ id: **SESSION_A** } ???

Set-Cookie: sessionid=**SESSION_A**

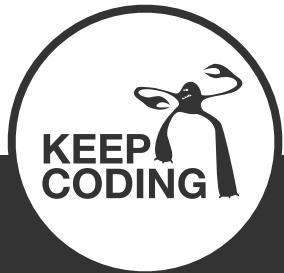
{ id: **SESSION_A**, cart_products: [...] }



Anatomía de una cookie

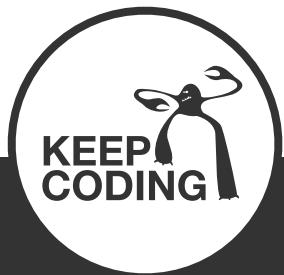
Las cookies están compuestas por los siguientes elementos:

- **domain:** dominio del servidor que ha puesto la cookie
- **path:** ruta opcional de la página que ha puesto la cookie. Si existe, sólo se enviará la cookie de vuelta a esa ruta.
- **expires:** fecha de expiración de la cookie
- **secure:** flag opcional para indicar que la cookie sólo se enviará por protocolo HTTPS
- **httpOnly:** flag opcional que indica que solo la puede leer el servidor
- **value:** separados por ; pueden tener nombre.



Anatomía de una cookie

```
Set-Cookie: name=darth; role=admin;  
domain=deathstar.com; path=/home; secure;  
expires=Tue, 29 Jan 1985 01:02:03 GMT+2
```



■ Hash de contraseñas

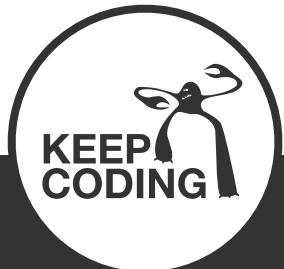
- **PBKDF2**: <https://keepcoding.io/blog/que-es-pbkdf2/>
- **bcrypt**: <https://codahale.com/how-to-safely-store-a-password/>
- **scrypt**: <https://github.com/Tarsnap/scrypt>
- **argon2**: <https://medium.com/@mpreziuso/password-hashing-pbkdf2-scrypt-bcrypt-and-argon2-e25aaf41598e>

Como elegir? (spoiler: PBKDF2 está antiguo, cualquiera de los otros tres es bueno)

<https://stytch.com/blog/argon2-vs-bcrypt-vs-scrypt/>

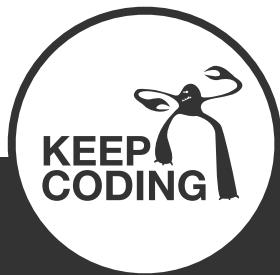
<https://hashing.dev/about/>

<https://npmtrends.com/argon2-vs-bcrypt-vs-scrypt-js>





■ Consumir APIs de terceros

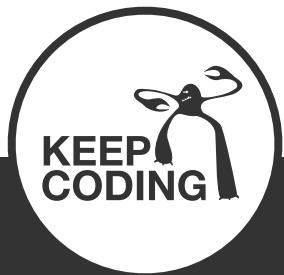


■ Consumir APIs

Uno de los módulos más usados para esto es *request*.

<https://github.com/request/request>

npm install request



■ Consumir APIs

```
var options = {  
  method: 'GET',  
  url: 'https://api.punkapi.com/v2/beers/random',  
  //headers: {'User-Agent': '...'},  
  json: true  
};  
  
request(options, function (err, response, body) {  
  if (err || response.statusCode >= 400) {  
    console.error(err, response.statusCode);  
    return;  
  }  
  // body tendrá nuestro contenido  
}) ;
```

