



neitek_

NodeJS

Carolina Caballero

Julio 2021

Agenda

- Introducción
- Timers
- Programación asíncrona y callbacks
- Promises
- Async & Await
- EventLoop
- Invocaciones HTTP Request

NodeJS



NodeJS

- Es un entorno en tiempo de **ejecución** multiplataforma, es open-source, para la capa del **servidor** basado en el lenguaje de programación **JavaScript**.
- Se ejecuta en el servidor, no en el navegador.
- Basado en el motor V8 de Google.
- V8: es un entorno de ejecución para JavaScript creado para Google Chrome, compila el código fuente JavaScript en código máquina, en vez de interpretarlo en tiempo real

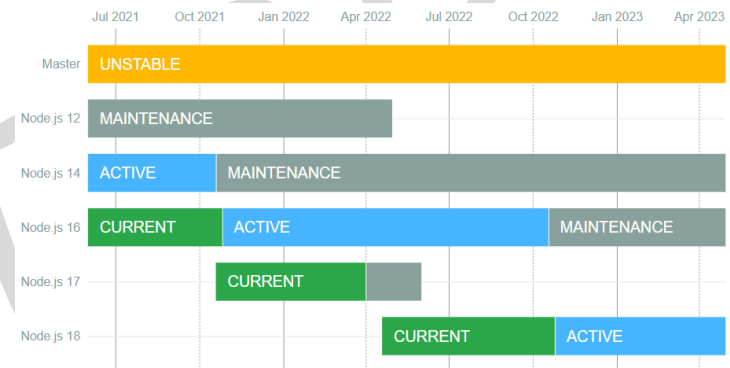
Historia



- Node.js es creado por inicialmente por Ryan Dahl en 2009.
- Debido a las limitaciones en su tiempo de los web servers para manejar muchas conexiones concurrentes (mas de 10mil).
- Actualmente se encuentra la versión Node.js 16

Versiones

- Cada 6 meses sacan releases.
- Los releases impares se dejan de soportar a los 6 meses
- Los releases pares se vuelven LTS (Long-Term Support)






Instalación Nodejs

- ◉ <https://nodejs.org/en/download/>

Downloads

Latest LTS Version: 14.17.3 (includes npm 6.14.13)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS Recommended For Most Users	Current Latest Features
 Windows Installer <small>node-v14.17.3-x64.msi</small>	 macOS Installer <small>node-v14.17.3.pkg</small>
 Source Code <small>node-v14.17.3.tar.gz</small>	

Windows Installer (.msi)

Windows Binary (.zip)

macOS Installer (.pkg)

macOS Binary (.tar.gz)

Linux Binaries (x64)

Linux Binaries (ARM)

Source Code

32-bit	64-bit
32-bit	64-bit
64-bit	
64-bit	
64-bit	
ARMv7	ARMv8
node-v14.17.3.tar.gz	

Instalación Visual Studio Code

- <https://code.visualstudio.com/>

The image shows a screenshot of the Visual Studio Code website and a code editor window. The website header includes the Visual Studio Code logo, navigation links (Docs, Updates, Blog, API, Extensions, FAQ, Learn), a search bar, and a 'Download' button. A banner below the header states 'Version 1.58 is now available! Read about the new features and fixes from June.' The main content area features the text 'Code editing. Redefined.' and 'Free. Built on open source. Runs everywhere.' Below this is a 'Download for Windows' button with a dropdown arrow, and a red arrow points to it. Other platforms and Insiders builds are also mentioned. The code editor window shows the 'EXTENSIONS: MARKETPLACE' sidebar with a list of extensions including Python, GitLens, C/C++, ESLint, Debugger for Chrome, and Language Support. The main editor area displays a JavaScript file named 'serviceWorker.js' with code for registering a service worker.

Visual Studio Code Docs Updates Blog API Extensions FAQ Learn

Search Docs Download

Version 1.58 is now available! Read about the new features and fixes from June.

Code editing. Redefined.

Free. Built on open source. Runs everywhere.

Download for Windows Stable Build

Other platforms and Insiders builds

By using VS Code, you agree to its [license and privacy statement](#).

EXTENSIONS: MARKETPLACE

- @sortinstalls
- Python 2019.6.24221 54.9M 4.5 Linting, Debugging (multi-threaded, ... Microsoft Install
- GitLens — Git sup... 9.8.5 23.1M 5 Supercharge the Git capabilities built... Eric Amodio Install
- C/C++ 0.24.0 23M 3.5 C/C++ IntelliSense, debugging, and ... Microsoft Install
- ESLint 1.9.0 21.9M 4.5 Integrates ESLint JavaScript into VS ... Dirk Baumer Install
- Debugger for Ch... 4.11.6 20.6M 4 Debug your JavaScript code in the C... Microsoft Install
- Language Supp... 0.47.0 18.7M 4.5 Java Linting, Intellisense, formatting, ... Red Hat Install

src > JS serviceWorker.js > register > window.addEventListener('load') callback

```
39 checkValidServiceWorker(swUrl, config);
40 // Add some additional logging to localhost, p
41 // service worker/PWA documentation.
42 navigator.serviceWorker.ready.then(() => {
43   product
44   productSub
45   removeSiteSpecificTrackingException
46   removeWebWideTrackingException
47   requestMediaKeySystemAccess
48   sendBeacon
49   serviceWorker (property) Navigator.serviceWorke...
50   storage
51   storeSiteSpecificTrackingException
52   storeWebWideTrackingException
53   userAgent
54   vendor
55 }
56
57 function registerValidSW(swUrl, config) {
58   navigator.serviceWorker
59   register(swUrl);
60 }
```


Iniciar un proyecto

• npm init

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

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

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

Press ^C at any time to quit.
package name: (cursonodejs)
version: (1.0.0)
description: ejercicios de node.js
entry point: (index.js)
test command:
git repository:
keywords: nodejs, javascript
author: Carolina Caballero
license: (ISC) MIT

Iniciar un proyecto

- Se crea un archivo: package.json

package.json > ...

```
1 {  
2   "name": "cursonodejs",  
3   "version": "1.0.0",  
4   "description": "ejercicios de node.js",  
5   "main": "index.js",  
6   "scripts": {  
7     "test": "echo \"Error: no test specified\" && exit 1"  
8   },  
9   "keywords": [  
10    "nodejs",  
11    "javascript"  
12  ],  
13   "author": "Carolina Caballero",  
14   "license": "MIT"  
15 }  
16
```

Ejecutar un programa

- Para ejecutar un programa con Node.js se realiza de la siguiente manera:
- **node** sample01.js
- **node** sample01

Módulos (modules)

- En Node.js un módulo es una funcionalidad simple o compleja organizada en uno o mas archivos JavaScript que puede ser reutilizada por la aplicación Node.js
- Cada módulo tiene su propio context.
- Es similar lo que viene siendo librerías en JavaScript

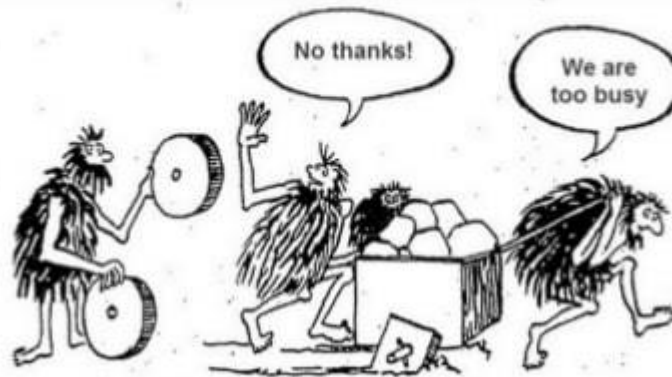
Node.js Built-in Modules

- Node.js viene con un conjunto de módulos incorporados (También son llamados Core Modules), algunos de ellos son:

Core Module	Description
http	http module includes classes, methods and events to create Node.js http server.
url	url module includes methods for URL resolution and parsing.
querystring	querystring module includes methods to deal with query string.
path	path module includes methods to deal with file paths.
fs	fs module includes classes, methods, and events to work with file I/O.
util	util module includes utility functions useful for programmers.

Cartoon time...

Useful Libraries



Cargar módulo Core

- Para usar un módulo, primero se requiere importarlo:

```
var module = require('module_name');
```

Crear y Exportar un módulo local

```
sample02-func.js > ...  
1  function suma (a, b) {  
2    |   return a + b;  
3  }  
4  
5  exports.suma = suma;
```

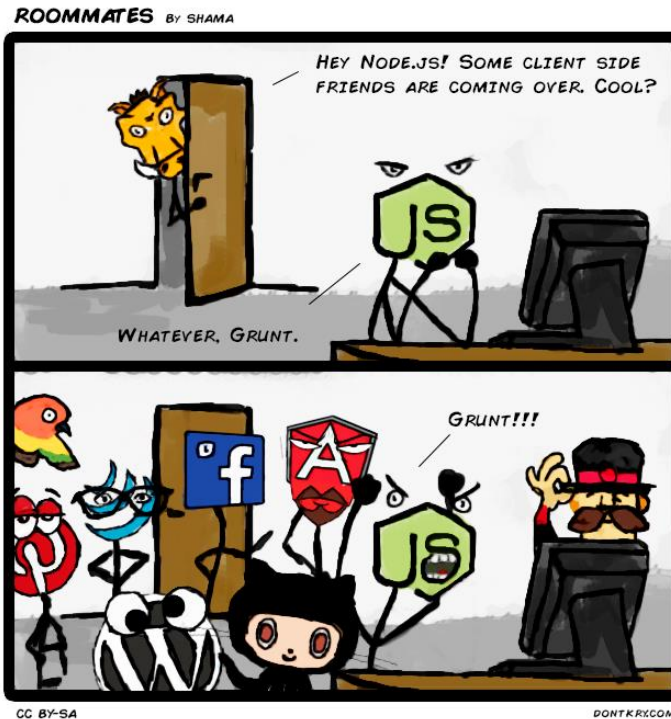
```
sample02.js > ...  
1  const op = require ('./sample02-func');  
2  
3  console.log (op.suma(1 ,5) );
```

node sample02

Crear y Exportar un módulo local

- Ver y ejecutar los siguientes programas:
 - node sample03
 - node sample04
 - node sample05

Cartoon time...



Agenda

- Introducción
- Timers
- Programación asíncrona y callbacks
- Promises
- Async & Await
- EventLoop
- Invocaciones HTTP Request

Timers

- Es un módulo de NodeJS que contiene funciones para ejecutar código después de un cierto periodo de tiempo
- No requiere importarse

Timers

Method	Description
<code>clearImmediate()</code>	Cancels an Immediate object
<code>clearInterval()</code>	Cancels an Interval object
<code>clearTimeout()</code>	Cancels a Timeout object
<code>ref()</code>	Makes the Timeout object active. Will only have an effect if the <code>Timeout.unref()</code> method has been called to make the Timeout object inactive.
<code>setImmediate()</code>	Executes a given function immediately.
<code>setInterval()</code>	Executes a given function at every given milliseconds
<code>setTimeout()</code>	Executes a given function after a given time (in milliseconds)
<code>unref()</code>	Stops the Timeout object from remaining active.

Timers

```
setTimeout(() => {  
  console.log("Hola");  
}, 2000)
```

```
setTimeout(() => {  
  console.log("Mundo");  
}, 50)
```

```
const id = setInterval(() => {  
  // runs every 2 seconds  
  console.log('.')  
}, 500);
```

```
setTimeout( () => {  
  clearInterval(id)  
}, 4000);
```

Ejercicio

- Haz un arreglo de nombres, con al menos 5 nombres.
- Despliega los nombres del arreglo cada segundo.
- El programa debe terminar cuando se desplieguen todos los nombres.

Agenda

- Introducción
- Timers
- Programación asíncrona y callbacks
- Promises
- Async & Await
- EventLoop
- Invocaciones HTTP Request

Asincronicidad en Lenguajes de programación

- **Asíncrono** significa que las cosas sucedan independientemente del flujo de programa principal.
- Las computadoras, de hecho, trabajan en forma asíncrona, cada programa se ejecuta en un tiempo específico y después se detiene, para permitir a otro programa continuar su ejecución. Todo esto pasa en un ciclo muy rápido que es imposible notarlo.



JavaScript es Síncrono

- Javascript, como la mayoría de los lenguajes de programación son Síncronos.
- Cada línea de programación se ejecutan una por una, en serie.
- Lo que lo hace poder ser asíncrono es el ambiente, en este caso el **browser**, que provee un conjunto de APIs que lo permiten.
- A su vez Node.js ha introducido un non-blocking I/O environment

Callbacks

- Un **Callback** es una función que será llamada cuando se termine cierta tarea, esto previene bloqueos y permite que se siga ejecutando otro código al mismo tiempo.

Conversión típica de Callbacks

```
function asyncOperation ( a, b, c, callback ) {  
  // ... lots of hard work ...  
  if ( /* an error occurs */ ) {  
    return callback(new Error("An error has occurred"));  
  }  
  // ... more work ...  
  callback(null, d, e, f);  
}  
  
asyncOperation ( params.., function ( err, returnValues.. ) {  
  //This code gets run after the async operation gets run  
});
```

Diferencia Sync Vs Async

Síncrono

```
function processData () {  
  var data = fetchData ();  
  data += 1;  
  return data;  
}
```

Asíncrono

```
function processData (callback) {  
  fetchData(function (err, data) {  
    if (err) {  
      console.log("An error has occurred. Abort everything!");  
      return callback(err);  
    }  
    data += 1;  
    callback(data);  
  });  
}
```

Callbacks

```
function hola (nombre, callback) {  
  setTimeout(function () {  
    console.log ("Hola " + nombre);  
    callback();  
  }, 1000);  
}  
  
function adios (nombre, callback2) {  
  setTimeout(function() {  
    console.log ('Adios ' + nombre);  
    callback2();  
  }, 500)  
}  
  
console.log("Inicia...");  
hola ("Caro", function() {  
  adios('Caro', function() {  
    console.log ('Termina');  
  } );  
});
```

Callback hell



```
console.log("Inicia...");
hola ("Caro", function(nombre) {
    hablar(function() {
        hablar (function() {
            hablar (function() {
                adios(nombre, function() {
                    console.log ('Termina');
                } );
            } );
        } );
    } );
});
});
});
```

Cartoon time



Agenda

- Introducción
- Timers
- Programación asíncrona y callbacks
- Promises
- Async & Await
- EventLoop
- Invocaciones HTTP Request

Promises

- Estandarizadas e introducidas en ES2015
- A promise is commonly defined as a **proxy for a value that will eventually become available**.
- Once a promise has been called, it will start in a **pending state**.
- The created promise will eventually end in a resolved state, or in a rejected state, calling the respective callback functions (passed to then and catch) upon finishing

Creating a Promise

```
let done = true

const isItDoneYet = new Promise((resolve, reject) => {
  if (done) {
    const workDone = 'Here is the thing I built'
    resolve(workDone)
  } else {
    const why = 'Still working on something else'
    reject(why)
  }
})
```

Leyendo un archivo con Promise

```
const fs = require('fs')

const getFile = (fileName) => {
  return new Promise((resolve, reject) => {
    fs.readFile(fileName, (err, data) => {
      if (err) {
        reject(err) // calling `reject` will cause the promise to be rejected
        return      // and we don't want to go any further
      }
      resolve(data)
    })
  })
}

getFile('/etc/passwd')
  .then(data => console.log(data))
  .catch(err => console.error(err))
```

Consume a Promise

```
const isItDoneYet = new Promise(/* ... as above ... */)
//...

const checkIfItsDone = () => {
  isItDoneYet
    .then(ok => {
      console.log(ok)
    })
    .catch(err => {
      console.error(err)
    })
}
```

Cambio a Promise

```
// solo con Callback  
function hola (nombre, callback) {  
  setTimeout(function () {  
    console.log ("Hola " + nombre);  
    callback(nombre);  
  }, 500);  
}
```

```
// con Promise  
function hola (nombre) {  
  return new Promise (function(resolve, reject) {  
    setTimeout(function () {  
      console.log ("Hola " + nombre);  
      resolve(nombre);  
    }, 500);  
  });  
}
```

Cartoon time...



Agenda

- Introducción
- Timers
- Programación asíncrona y callbacks
- Promises
- Async & Await
- EventLoop
- Invocaciones HTTP Request

Async & Await

- Introducidas en ES2017
- Async/Await estan construidas en promesas
- Ayudan a eliminar código repetitivo (boilerplate code)
- A simple vista parece codigo síncrono, sin embargo es asíncrono.
- El **await** solo es válido dentro de una function **async**

Async & Await

```
const doSomethingAsync = () => {  
  return new Promise(resolve => {  
    setTimeout(() => resolve('I did something'), 3000)  
  })  
}
```

```
const doSomething = async () => {  
  console.log(await doSomethingAsync())  
}
```

```
console.log('Before')  
doSomething()  
console.log('After')
```

When you want to **call** this function (`doSomethingAsync`) you prepend **await**, and the **calling code will stop until the promise is resolved or rejected**. One caveat: the client function must be defined as **async**

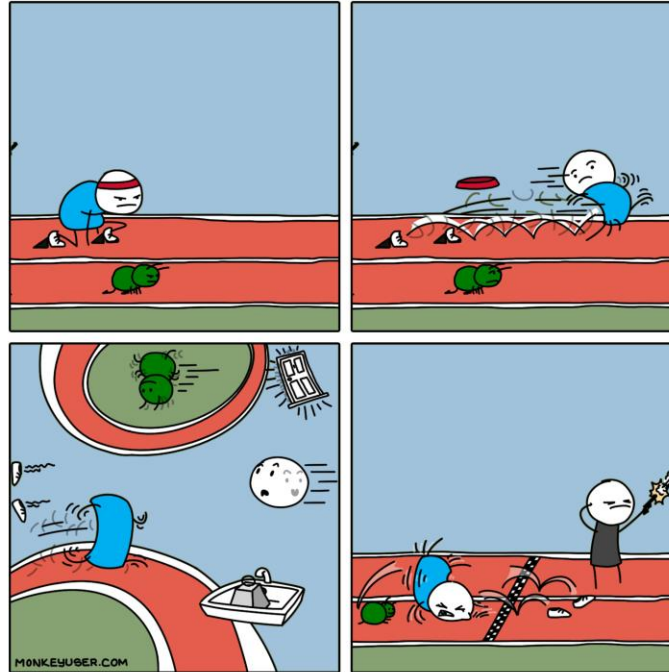
Cambio a Async & Await

```
// con Promise
function hola (nombre) {
  return new Promise (function(resolve, reject) {
    setTimeout(function () {
      console.log ("Hola " + nombre);
      resolve(nombre);
    }, 500);
  });
}
```

```
//async
async function hola (nombre) {
  return new Promise (function(resolve, reject) {
    setTimeout(function () {
      console.log ("Hola " + nombre);
      resolve(nombre);
    }, 500);
  });
}
```

Cartoon time...

ASync



Agenda

- Introducción
- Timers
- Programación asíncrona y callbacks
- Promises
- Async & Await
- EventLoop
- Invocaciones HTTP Request

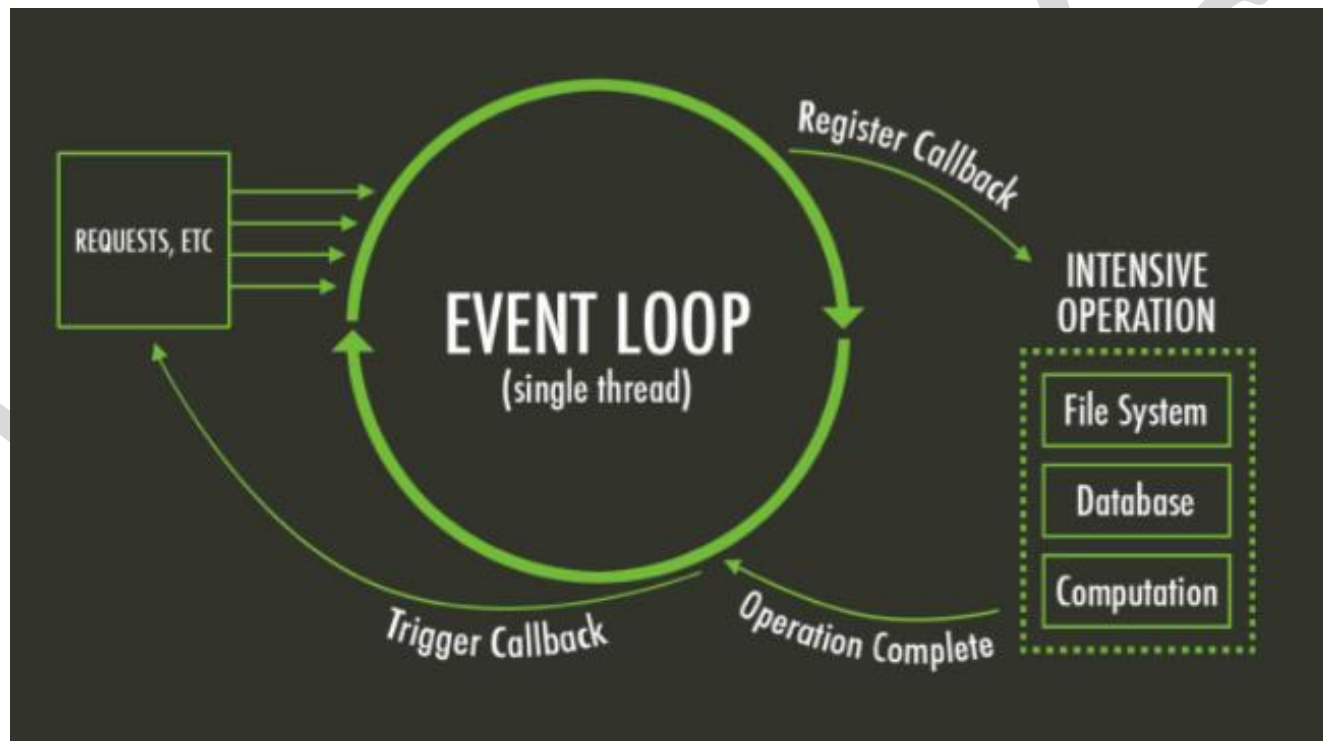
Event loop

- The **Event Loop** is one of the most important aspects to understand about Node.js.
- Explains how Node.js can be asynchronous and have non-blocking I/O.
- The Node.js JavaScript code runs on a single thread. There is just one thing happening at a time.

Event loop

- The call stack is a LIFO (Last In, First Out) stack.
- The event loop continuously checks the **call stack** to see if there's any function that needs to run.

Event loop



Event loop

```
const bar = () => console.log('bar')
```

```
const baz = () => console.log('baz')
```

```
const foo = () => {  
  console.log('foo')  
  bar()  
  baz()  
}
```

```
foo()
```

Event loop



Agenda

- Introducción
- Timers
- Programación asíncrona y callbacks
- Promises
- Async & Await
- EventLoop
- Invocaciones HTTP Request

HttpServer Hello World

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

const port = 3000

const server = http.createServer((req, res) => {
  res.statusCode = 200
  res.setHeader('Content-Type', 'text/html')
  res.end('<h1>Hello, World!</h1>')
})

server.listen(port, () => {
  console.log(`Server running at port ${port}`)
})
```

File: server01.js

Invocaciones HTTP Request

- Vamos a crear un servidor en node.js
- Usaremos el modulo express
- Usaremos el modulo body-parser
- Recibiremos invocaciones Get y Post

Instalación de express

- Modulo para manejar peticiones HTTP
- `npm i express`

```
npm notice created a lockfile as package-lock.json. You should commit this file.
```

```
npm WARN backendnode@1.0.0 No repository field.
```

```
+ express@4.17.1
```

```
added 50 packages from 37 contributors and audited 50 packages in 3.771s  
found 0 vulnerabilities
```

HttpServer con express

```
const express = require ('express');  
  
var app = express();  
  
app.use ('/', function (req, res) {  
  res.send('Hola');  
});  
  
app.listen (3000);  
console.log('La aplicacion esta escuchando en http://localhost:3000')
```

Instalación de body-parser

- Módulo para trabajar con el body de la petición
- npm i body-parser

```
-curso\backendNode>npm install body-parser
npm WARN backendnode@1.0.0 No repository field.

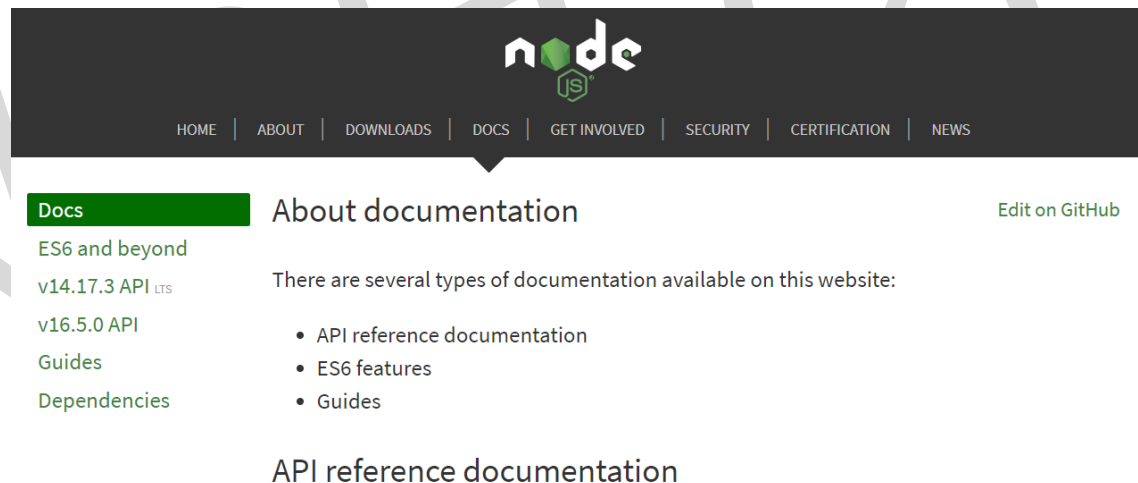
+ body-parser@1.19.0
updated 1 package and audited 51 packages in 1.321s
found 0 vulnerabilities
```


Aceptar peticiones get y post

```
router.get('/', function(req, res) {  
  res.send ('Hola desde get');  
});  
router.post('/', function(req, res) {  
  res.send ('Hola desde post');  
});
```

Para saber mas...

<https://nodejs.org/en/docs/>



git commit -m "Git course ends"

