
SENPAI
ACADEMY

Node.js

SENPAI
ACADEMY

Node.js - Introducción

Node.js es un entorno de ejecución de JavaScript, que nos permite correr JavaScript fuera del navegador.

Es un entorno de ejecución multiplataforma, de código abierto, y lo vamos a utilizar para la capa del servidor de nuestras aplicaciones Web.

Al contrario que la mayoría del código JavaScript, no se ejecuta en un navegador, sino en el servidor.

Está basado en el motor de JavaScript V8 de Chrome (Google).

Dato: V8 es el entorno de ejecución para JavaScript creado para Google Chrome. Es software libre desde 2008, está escrito en C++ y compila el código fuente JavaScript en código de máquina en lugar de interpretarlo en tiempo real.

SENPAI
ACADEMY

Node.js - Instalación

Necesitamos instalarlo localmente en nuestras computadoras para poder ejecutar y probar las aplicaciones que crearemos.

Cuando se publica una aplicación en producción, Node.js estará instalado en el servidor donde dejamos nuestra aplicación final.

Se pueden descargar la última versión (+14) desde la página oficial:
https://nodejs.org/es/

Una vez instalado, pueden abrir una terminal (consola de comandos) y validar que haya quedado instalado correctamente ejecutando el comando:

```
node -v
```

También pueden consultar información sobre los distintos comandos de Node con:

```
node --help
```

SENPAI ACADEMY

Node.js - Primeros pasos

```
// Archivo app.js

const carrito = ["pan", "manzana", "naranja", "tomate"];

carrito.forEach((item) => {
  console.log(item);
});
```

Si creamos un archivo de ejemplo, llamado *app.js*, y le agregamos este código JavaScript, podemos desde una terminal (parados en la ubicación del archivo) ejecutar el archivo con Node.js, con el comando:

```
node app.js
```

SENPAI ACADEMY

Node.js vs Navegador

Node.js vs Navegador

Tanto Node.js como el Navegador utilizan JavaScript como lenguaje de programación, sin embargo, escribir aplicaciones que corran en el Navegador es completamente diferente a escribir aplicaciones que corran en Node.js

Lo principal que cambia es el entorno de ejecución, el ecosistema.
En el Navegador normalmente vamos a estar interactuando con el **DOM**, u otras APIs Web como cookies.
Esto no existe en Node.js, no tenemos los objetos globales ***document*** o ***window*** que si tenemos en el Navegador.

Osea que desde Node.js no podemos acceder al **DOM**, pero si podemos realizar otras cosas como acceder al FileSystem para leer y escribir archivos.

SENPAI
ACADEMY

Node.js vs Navegador

Otra diferencia es que en Node.js nosotros controlamos el entorno, en que versión de Node.js corre nuestra aplicación, mientras que en el Navegador no sabemos qué versión del Navegador los usuarios van a tener.

Esto nos permite escribir “código moderno” ES6-7-8-9 que nuestra versión de Node.js soporte. Los navegadores suelen ser más lentos para actualizar las características del lenguaje que soportan.

Otra diferencia es que Node.js utiliza el sistema de módulos llamado CommonJS. Esto en la práctica esto significa que podemos utilizar `require()` en vez de `import` como en el Navegador.

SENPAI ACADEMY

Exportar - Importar en Node.js

SENPAI
ACADEMY

Node.js - Sistema de Módulos

Node.js tiene un sistema de módulos incorporado. Un archivo Node.js puede importar la funcionalidad expuesta por otros archivos Node.js (y es algo que vamos a realizar muy frecuentemente).

Para importar una función en *app.js* desde otro archivo (por ejemplo: *carrito.js*) utilizamos **`require`** y la ruta del archivo:

```
1  // Archivo app.js
2
3  const carrito = require("./carrito");
4
5  carrito.forEach((item) => {
6    console.log(item);
7  });
```

Y desde el archivo *carrito.js* exportamos ese array:

```
1  // Archivo carrito.js
2
3  const carrito = ["pan", "manzana", "naranja", "tomate", "queso"];
4
5  module.exports = carrito;
```

SENPAI ACADEMY

Node.js - Sistema de Módulos

También podemos exportar múltiples variables:

```
// Archivo carrito.js

const elementos = ["pan", "manzana", "naranja", "tomate"];
const total = 1000;

module.exports = {
  elementos: elementos,
  total: total
};

```

SENPAI ACADEMY

Node.js - Sistema de Módulos

Y ahora al importarlo, tenemos un objeto con esas dos propiedades:

```
// Archivo app.js

const carrito = require("./carrito");

carrito.elementos.forEach((item) => {
  console.log(item);
});

console.log("Precio Total: ", carrito.total);
```

Y si utilizamos destructuring:

```
// Archivo app.js

const { elementos, total } = require("./carrito");

elementos.forEach((item) => {
  console.log(item);
});

console.log("Precio Total: ", total);
```

SENPAI ACADEMY

package.json

Node.js - package.json

El *package.json* es el archivo que contiene la información de nuestro proyecto, estará en la carpeta raíz del mismo.

Por ahora, lo que más nos importa es que dentro tendrá una lista de dependencias y scripts, entre otros datos como el nombre, autor, licencia, etc.
Documentación

Podemos crearlo con el comando:

```
npm init -y
```

```json
{
  "name": "playground",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

npm es instalado junto con Node.js, en breve vamos a ver de qué se trata

SENPAI ACADEMY

NPM

Node Package Manager

SENPAI
ACADEMY

Node.js - NPM

NPM es el administrador de paquetes estándar para Node.js. Existen otros, como yarn, que funcionan de forma muy similar pero con otros comandos.

Lo vamos a utilizar desde la línea de comandos para instalar las dependencias que tendrá nuestro proyecto, es decir, librerías externas que utilicemos, y también para correr algunos scripts que podemos definir en el *package.json*

Instalar un paquete:

```
npm install <package-name>
```

Por ejemplo, instalemos cowsay

```
npm install cowsay
```

Esto actualiza nuestro *package.json* y agrega la sección de dependencias que mencionamos antes:

```json
{
  "name": "playground",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cowsay": "^1.4.0"
  }
}
```

Node.js - NPM

Las dependencias se instalan en una carpeta llamada *node_modules*. Esta carpeta no la vamos a subir a nuestro repositorio, ya que normalmente es muy pesada, tiene miles de archivos, y no es necesario porque se puede volver a generar mientras tengamos las dependencias listadas en el *package.json*, y ejecutemos el comando:

```
npm install
```

Normalmente vamos a instalar las dependencias de forma local al proyecto (como ya vimos), pero a veces queremos instalar algo de forma global, que queda en nuestra computadora para cualquier proyecto, para lo que agregamos la bandera *-g*

```
npm install -g <package-name>
```

SENPAI ACADEMY

Node.js - NPM

Para utilizar una dependencia en nuestro proyecto (librería), la debemos que importar de la misma forma que cuando importamos una función o variable desde otro archivo, pero sin necesidad de darle la ruta a la carpeta, simplemente con el nombre:

```
1  // Archivo app.js
2
3  const cowsay = require("cowsay");
4
5  console.log(
6    cowsay.say({
7      text: "Soy un moooodulo",
8      e: "Oo",
9      T: "U",
10   })
11 );
```

```
1   ___________________
2  < Soy un moooodulo >
3   -------------------
4          \   ^__^
5           \  (Oo)\_______
6              (__)\       )\/\
7               U ||----w |
8                 ||     ||
```

SENPAI ACADEMY

Node.js - NPM

También vamos a utilizar NPM para ejecutar los scripts que definamos en nuestro *package.json*, con:

```
npm run <script-name>
```

Es muy común tener un script para ejecutar nuestra aplicación, por lo que ahora podemos correr:

```
npm run dev
```

```
{
  "name": "playground",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "dev": "node app.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cowsay": "^1.4.0"
  }
}
```

SENPAI ACADEMY

Servidor HTTP

SENPAI
ACADEMY

Node.js - Servidor HTTP

Hypertext Transfer Protocol
El Protocolo de transferencia de hipertexto es el protocolo de comunicación que permite las transferencias de información entre aplicaciones Web.

- Se utiliza para intercambiar información entre el cliente y servidor
- El Servidor queda a la espera de alguna solicitud HTTP (Request) ejecutada por el cliente, y proporciona una respuesta (Response)

HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado.

Los que normalmente utilizamos son: **GET, POST, PUT, DELETE**

SENPAI
ACADEMY

Node.js - Servidor HTTP

```
1   const http = require("http");
2
3   const port = process.env.PORT || 4000;
4
5   const server = http.createServer((req, res) => {
6     res.statusCode = 200;
7     res.setHeader("Content-Type", "text/html");
8     res.end("<h1>Hello, World!</h1>");
9   });
10
11  server.listen(port, () => {
12    console.log(`Server running at port ${port}`);
13  });
```

Un pequeño ejemplo rápido de un servidor

SENPAI ACADEMY

SENPAI
ACADEMY

gustavguez

gustavguez

GUSTAVO RODRIGUEZ

FULL STACK DEVELOPER
SOLCRE