NODE.JS

Node.js es un entorno de tiempo de ejecución de JavaScript (de ahí su terminación en .js haciendo alusión al lenguaje JavaScript). Este **entorno de tiempo** de ejecución en tiempo real incluye todo lo que se necesita para ejecutar un programa escrito en JavaScript.

Node.js fue creado por los **desarrolladores originales de JavaScript.** Lo transformaron de algo que solo podía ejecutarse en el navegador en algo que se podría ejecutar en los ordenadores como si de aplicaciones independientes se tratara.

Tanto JavaScript como **Node.js** se ejecutan en el motor de tiempo de ejecución JavaScript V8 (V8 es el nombre del motor de JavaScript que alimenta Google Chrome. Es lo que toma nuestro JavaScript y lo ejecuta mientras navega con Chrome). Este motor coge el código JavaScript y lo convierte en un **código de máquina** más rápido. El código de máquina es un código de nivel más bajo que la computadora puede ejecutar sin necesidad de interpretarlo primero, ignorando la compilación y por lo tanto aumentando su velocidad.

JavaScript se utilizó principalmente para la creación de scripts del lado del cliente. Dado que JavaScript sólo podía utilizarse dentro de la etiqueta <script>, los desarrolladores tenían que trabajar en múltiples lenguajes y marcos entre los componentes del front-end y del back-end. Más tarde llegó Node.js, que es un entorno de ejecución que incluye todo lo necesario para ejecutar un programa escrito en JavaScript.

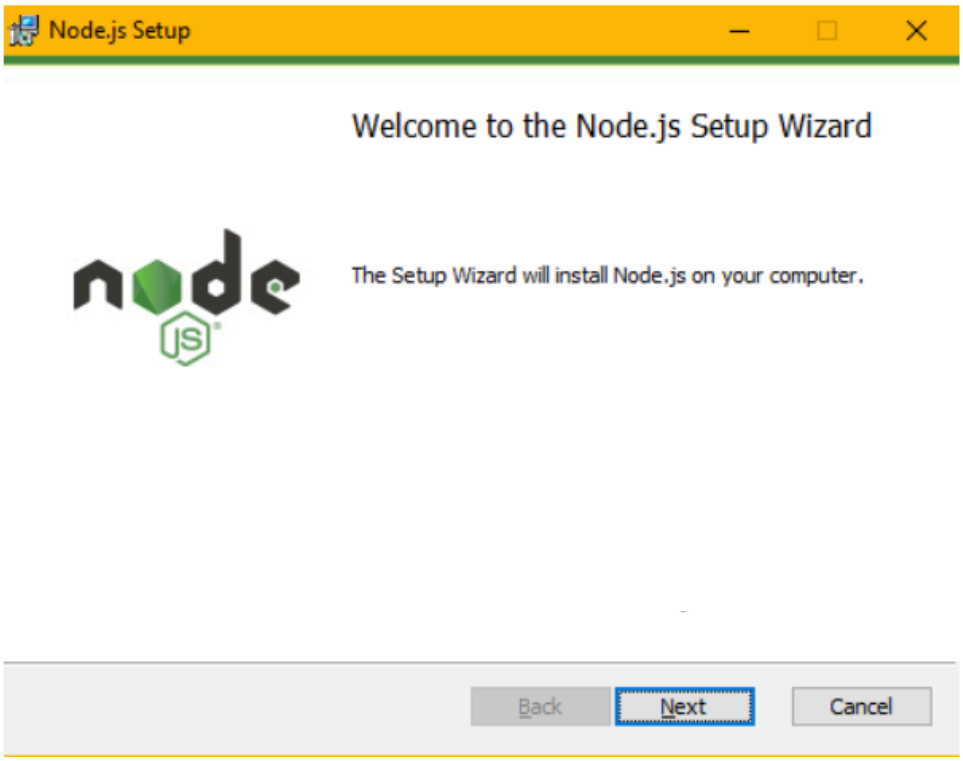
Node.js no es un lenguaje de programación. Más bien, es un entorno de ejecución que se utiliza para ejecutar JavaScript fuera del navegador. Node.js tampoco es un framework (una plataforma para desarrollar aplicaciones de software). El tiempo de ejecución de Node.js se construye sobre un lenguaje de programación -en este caso, JavaScript- y ayuda a la ejecución de los propios frameworks.

NPM es el ecosistema de paquetes de Node.js. Es el mayor ecosistema de todas las bibliotecas de código abierto del mundo, con más de un millón de paquetes y creciendo. El uso de NPM es gratuito y miles de desarrolladores de código abierto contribuyen a él diariamente.

NPM viene con una utilidad de línea de comandos fuera de la caja. Sólo tienes que dirigirte al sitio web de NPM para buscar el paquete que necesitas e instalarlo con un solo comando. También puedes gestionar las versiones de tus paquetes, revisar las dependencias e incluso configurar scripts personalizados en tus proyectos a través de esta utilidad de línea de comandos. Sin duda, NPM es la posesión más querida por la comunidad de Node.js; Node.js atrae a un gran número de desarrolladores debido en gran parte a su excelente soporte de paquetes.

1. DESCARGA E INSTALACIÓN

La descarga desde paquete se realiza de forma inmediata desde el sitio web.



Tras la descarga se ejecuta el instalador descargado y comienza el proceso a través de una asistente de instalación. Basta con pulsar Next hasta completar la instalación, no es necesaria ninguna configuración específica.

Para comprobar si la instalación se ha realizado de forma correcta se utilizan los comandos:

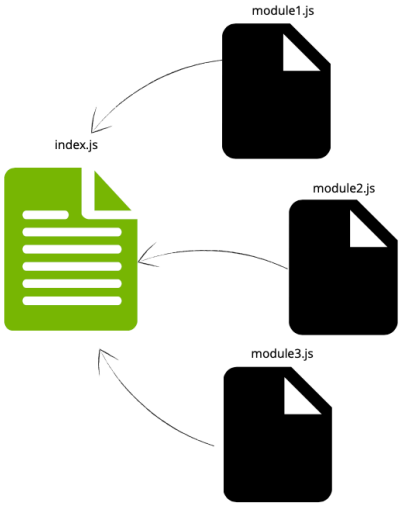
- **node --version.** Tras realizar el proceso de instalación anterior para confirmar el resultado satisfactorio se ejecuta esta línea por comandos y el resultado mostrará la versión de Node.js instalada.

- **npm --version.** NPM es un gestor de paquetes de JavaScript, si al ejecutarlo se muestra una versión, está instalado. Con la instalación de Node.js se instala también este gestor. Si se desea actualizar al versión basta con ejecutar por línea de comandos **npm install -g npm**

2. FUNCIONAMIENTO GENERAL

El desarrollo con Node.js se puede realizar desde un entorno de desarrollo más "complejo" como Visual Studio Code, hasta otros muchos más sencillos de usar, es decir, en un editor de texto como Notepad++.

El funcionamiento de Node.js se basa en la construcción de diferentes módulos que implementan la funcionalidad completa de cualquier aplicación que se va a desarrollar.



En primer lugar, se crean las funciones y demás código de aplicación en la distribución de módulos diseñada. Podemos diferenciar entre dos tipos de ficheros, el principal (habitualmente llamado index.js) y los ficheros que implementan cada uno de los módulos.

Como se puede observar en la imagen anterior, habitualmente se creará un archivo principal para el proyecto (**index.js**) ,desde el que serán llamados el resto de módulos de la aplicación.

Todos los ficheros que se van a desarrollar serán de tipo JavaScript, por lo tanto, **todos se almacenan con la extensión .js.**

A continuación, se crean tantos archivos correspondientes a los módulos y se implementa en ellos el código necesario para el funcionamiento de la aplicación.

Importante, para ejecutar un fichero creado utilizando Node.js se usa la siguiente instrucción desde el terminal o cmd. Es necesario que nos situemos en la ruta en la que está almacenado el fichero, por ejemplo, si hemos creado el archivo ficheroNode.js en la carpeta "proyectosNode" en primer lugar debemos acceder a esta ruta o ejecutar la instrucción añadiéndola.

```
>> cd proyectosNode
>> node ficheroNode.js
```

3. CONEXIÓN ENTRE MÓDULOS

Para realizar la conexión entre los módulos y con el fichero principal, el llamamiento entre archivos js no se realiza de la forma habitual, es decir, como cuando se desarrolla para el navegador, sino que ahora requiere de una implementación diferente, estamos trabajando desde el servidor.

```
<script src="fichero.js"></script>
```

Ahora bien, Node.js se utiliza la instrucción **require**, seguida del nombre completo del fichero .js del que se van a tomar las funciones implementadas.

Es importante indicar dónde está el fichero, es decir, la ruta exacta del mismo. Por ejemplo, si queremos conectar con un fichero que se encuentra en la misma ruta que el fichero desde el que es llamado se utiliza **"./"** delante del nombre del fichero.

```
require('./fichero.js');
```

Finalmente, el resultado de la llamada a la función **require** se almacena en una constante, puesto que para poder acceder a las funciones contenidas en el fichero .js serán necesario utilizar esta constante de acceso.

```
const c = require('./fichero.js');
```

Por tanto, la llamada a las funciones contenidas en los ficheros de módulo quedarían:

```
c.nombreFunción(parámetros);
```

HELLO WORLD CON NODE.

Vamos a empezar con el programa básico «Hello World», donde crearemos un servidor en Node.js que devolverá una salida «Hello World» en una petición al servidor. Antes de que te sumerjas, asegúrate de tener un buen editor de texto.

Node.js viene con un módulo incorporado llamado «HTTP» que permite a Node.js transferir datos a través del Protocolo de Transferencia de Hipertexto (HTTP).

Los pasaos para la creación de este primer programa se exponen a continuación:

1. Cargamos el módulo **http** en nuestro programa.

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

2. Indicamos hostname y port. Se definen las variables `hostname` y `port` que representan la dirección IP del host (en este caso, '127.0.0.1', que se refiere a la dirección local) y el número de puerto (en este caso, 3000) en el que el servidor escuchará las solicitudes.

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

3. Luego usamos el método **createServer** para aceptar una petición y devolver una respuesta con un código de estado. Dentro de la función de devolución de llamada, se configura la respuesta HTTP estableciendo el código de estado en 200 (OK), el tipo de contenido en 'text/plain', y el cuerpo de la respuesta en 'Hola Mundo!'.

```
const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World! Welcome to Node.js');
});
```

4. Finalmente, escuchamos en un puerto definido con **listen**. La función `listen` se utiliza para hacer que el servidor escuche en el puerto y la dirección IP especificados. Cuando el servidor comienza a escuchar, se ejecuta la función de devolución de llamada que imprime un mensaje en la consola indicando que el servidor está en ejecución.

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

5. Guarda este archivo como `server.js`. Desde el terminal inicia el servidor usando **node server js**

6. El servidor debería empezar a funcionar. Para verificar la salida, abra *http://localhost:3000* en su navegador.