Introducción a Node.js para el desarrollo en cliente

1. ¿Qué es Node.js?

Node.js es un entorno de ejecución en el lado del servidor construido utilizando el motor Javascript de Google Chrome, llamado V8. En esta sesión veremos cómo instalarlo y empezar a trabajar con él, pero antes conviene ser conscientes de lo que supone este paso en la historia del desarrollo web.

1.1. Evolución de Javascript

Como hemos comentado, Node.js es un entorno que emplea el lenguaje de programación Javascript y que se ejecuta en el lado del servidor. Esta afirmación puede resultar mundana, pero en realidad es algo sorprendente. Si echamos la vista atrás, el lenguaje Javascript ha pasado por varias etapas o fases de expansión sucesivas:

1. La primera tuvo lugar con el primer apogeo de la web, allá por los años 90. Se comenzaban a desarrollar webs con HTML, y el Javascript que se empleaba entonces permitía añadir dinamismo a esas páginas, bien validando formularios, abriendo ventanas, o explorando el DOM (estructura de elementos de la página), añadiendo o quitando contenidos del mismo. Era lo que se conocía como *HTML dinámico* o DHTML.

2. La segunda etapa llegó con la incorporación de las comunicaciones asíncronas, es decir, con AJAX, más o menos a principios del siglo XXI. Se desarrollaron librerías como *Prototype* (primero) o *jQuery* (después) que abrieron todo un mundo nuevo de posibilidades con el lenguaje. Con ellas se podían actualizar fragmentos de la página, llamando desde Javascript a documentos del servidor, recogiendo la respuesta y pegándola en una zona concreta de la página, sin necesidad de recargarla por completo.

3. Una siguiente etapa, vinculada a la anterior, tuvo lugar con la aparición de distintos frameworks Javascript para desarrollo de aplicaciones web en el lado del cliente, o *frontends*. Mediante una serie de funcionalidades incorporadas, y de librerías externas, permiten dotar a la aplicación cliente de una estructura muy determinada con unos añadidos que facilitan, entre otras cosas, la compartición de variables entre vistas o páginas, o la generación dinámica de contenido HTML en la propia vista. Hablamos, fundamentalmente, de frameworks como Angular o React.

4. La última etapa ha llegado con la expansión de Javascript al lado del servidor. Hasta este momento sólo se utilizaba en la parte cliente, es decir, fundamentalmente en los navegadores, por lo que sólo estábamos utilizando y viendo una versión reducida o restringida del lenguaje. Creíamos que Javascript sólo servía para validaciones, exploración del contenido HTML de una página o carga de contenidos en zonas concretas. Pero la realidad es que Javascript puede ser un lenguaje completo, y eso significa que podemos hacer con él cualquier cosa que se puede hacer con otros lenguajes completos, como Java o C#: acceder al sistema de ficheros, conectar con una base de datos, etc.

1.2. Javascript en el servidor. El motor V8

Bueno, vayamos asimilando esta nueva situación. Sí, Javascript ya no es sólo un lenguaje de desarrollo en el cliente, sino que se puede emplear también en el servidor. Pero... ¿cómo? En el caso de Node.js, como hemos comentado, lo que se hace es utilizar de manera externa el mismo motor de ejecución que emplea Google Chrome para compilar y ejecutar Javascript en el navegador: el motor V8. Dicho motor se encarga de compilar y ejecutar el código Javascript, transformándolo en código más rápido (código máquina).

También se encarga de colocar los elementos necesarios en memoria, eliminar de ella los elementos no utilizados (*garbage collection*), etc.

V8 está escrito en C++, es open-source y de alto rendimiento. Se emplea en el navegador Google Chrome y "variantes" como Chromium (adaptación de Chrome a sistemas Linux), además de en Node.js y otras aplicaciones. Podemos ejecutarlo en sistemas Windows (XP o posteriores), Mac OS X (10.5 o posteriores) y Linux (con procesadores IA-32, x64, ARM o MIPS).

Además, al estar escrito en C++ y ser de código abierto, podemos extender las opciones del propio Javascript. Como hemos comentado, inicialmente Javascript era un lenguaje concebido para su ejecución en un navegador. No podíamos leer un fichero de texto local, por ejemplo. Sin embargo, con Node.js se ha añadido una capa de funcionalidad extra a la base proporcionada por V8, de modo que ya es posible realizar estas tareas, gracias a que con C++ sí podemos acceder a los ficheros locales, o conectar a una base de datos.

1.2.1. Requisitos para que Javascript pueda correr en el servidor

¿Qué características tienen lenguajes como PHP, ASP.NET o JSP que no tenía Javascript hasta la aparición de Node.js, y que les permitían ser lenguajes en entorno servidor? Quizá las principales sean:

- Disponen de mecanismos para acceder al sistema de ficheros, lo que es particularmente útil para leer ficheros de texto, o subir imágenes al servidor, por poner dos ejemplos.

- Disponen de mecanismos para conectar con bases de datos.

- Permiten comunicarnos a través de Internet (el estándar ECMAScript no dispone de estos elementos para Javascript).

- Permiten aceptar peticiones de clientes y enviar respuestas a dichas peticiones.

Nada de esto era posible en Javascript hasta la aparición de Node.js. Este nuevo paso en el lenguaje le ha permitido, por tanto, conquistar también el otro lado de la comunicación cliente-servidor para las aplicaciones web.

1.2.2. Consecuencia: un único lenguaje de desarrollo web

La consecuencia lógica de utilizar Node.js en el desarrollo de servidor es que, teniendo en cuenta que Javascript es también un lenguaje de desarrollo en el cliente, nos hará falta conocer un único lenguaje para el desarrollo completo de una aplicación web. Antes de que esto fuera posible, era indispensable conocer, al menos, dos lenguajes: Javascript para la parte de cliente y PHP, JSP, ASP.NET u otro lenguaje para la parte del servidor.

1.3. Características principales de Node.js

Entre las principales características que ofrece Node.js, podemos destacar las siguientes:

- Node.js ofrece una API **asíncrona,** es decir, que no bloquea el programa principal cuando llamamos a sus métodos esperando una respuesta, y **dirigida por eventos**, lo que permite recoger una respuesta cuando ésta se produce, sin dejar al programa esperando por ella. Comprenderemos mejor estos conceptos más adelante, cuando los pongamos en práctica.

- La ejecución de código es **muy rápida** (recordemos que se apoya en el motor V8 de Google Chrome).

- Modelo **monohilo** pero muy **escalable**. Se tiene un único hilo atendiendo peticiones de clientes, a diferencia de otros servidores que permiten lanzar hasta N hilos en paralelo. Sin embargo, la API asíncrona y dirigida por eventos permite atender múltiples peticiones por ese único hilo, consumiendo muchos menos recursos que los sistemas multihilo.

- Se elimina la necesidad de ***cross-browser***, es decir, de desarrollar código Javascript que sea compatible con todos los navegadores, que es a lo que el desarrollo en el cliente nos tiene acostumbrados. En este caso, sólo debemos preocuparnos de que nuestro código Javascript sea correcto para ejecutarse en el servidor.

1.4. ¿Quién utiliza Node.js?

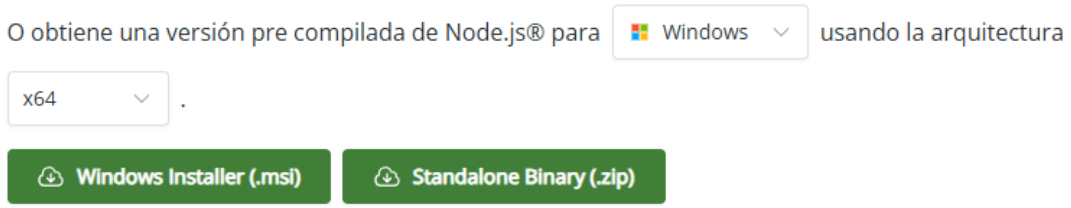
Es cierto que la mayoría de tecnologías emergentes suelen tardar un tiempo hasta tener buena acogida en nuestro país, salvo algunas pocas empresas pioneras. Pero sí hay varias empresas extranjeras, algunas de ellas de un peso relevante a nivel internacional, que utilizan *Node.js* en su desarrollo. Por poner algunos ejemplos representativos, podemos citar a Netflix, PayPal, Microsoft o Uber, entre otras.

2. Descarga e instalación

Antes de comenzar a dar nuestros primeros pasos con Node.js, vamos a instalarlo dependiendo de nuestro sistema operativo, comprobar que la instalación y versión son correctas, y editar un primer programa básico para probar con nuestro IDE.

2.1. Descargar e instalar Node.js

Para descargar Node.js, en general debemos acudir a su **web oficial (nodejs.org)**, y hacer clic sobre el enlace de descarga que aparecerá, que ya está preparado para el sistema operativo que estamos utilizando. Actualmente, en la página principal se nos ofrecen dos versiones alternativas:



La versión LTS es la recomendada para la mayoría de usuarios. Es una versión algo obsoleta (ya que no es la última), pero ofrece soporte a largo plazo (LTS, *Long Term Support*) y casi todas las funcionalidades añadidas a Node.js, salvo las que aparecen en la última versión disponible, que es la otra opción que se nos ofrece para los que quieran probar las últimas novedades.

2.1.1. Instalación en Windows, Mac y Linux

En el caso de querer instalar Node sobre un sistema Windows o Mac, hacemos clic en el enlace a la última versión (enlace derecho) y descargamos el paquete:

- Para sistemas **Windows** el paquete es un instalador (archivo *.msi*) que podemos directamente ejecutar para que se instale. Aceptamos el acuerdo de licencia, y confirmamos cada paso del asistente con los valores por defecto que aparezcan

Para sistemas **Mac OS X**, el paquete es un archivo *.pkg* que podemos ejecutar haciendo doble click en él, y seguir los pasos del asistente como en Windows.

2.1.2. Instalación en Linux o en la máquina virtual

Si estamos utilizando un sistema Linux, o la máquina virtual proporcionada para este curso, se recomienda instalar Node.js desde un repositorio. Vamos a suponer que utilizamos una distribución Debian (o Ubuntu, o similares), como la de la máquina virtual que tenéis disponible. En ese caso, podemos escribir estos comandos en un terminal en modo *root,* es decir, anteponiendo el comando su, con la correspondiente contraseña de *root*:

```
apt-get update
apt-get upgrade
apt-get install curl
curl -sL https://deb.nodesource.com/setup_9.x | bash -
apt-get install -y nodejs
```

NOTA: en el caso de tener el comando curl ya previamente instalado, sólo será necesario ejecutar las dos últimas instrucciones.

3. Módulos de terceros. El gestor "npm"

3.1. El gestor de paquetes "npm"

npm (*Node Package Manager*) es un gestor de paquetes para Javascript, y se instala automáticamente al instalar Node.js. Podemos comprobar que lo tenemos instalado, y qué versión concreta tenemos, mediante el comando:

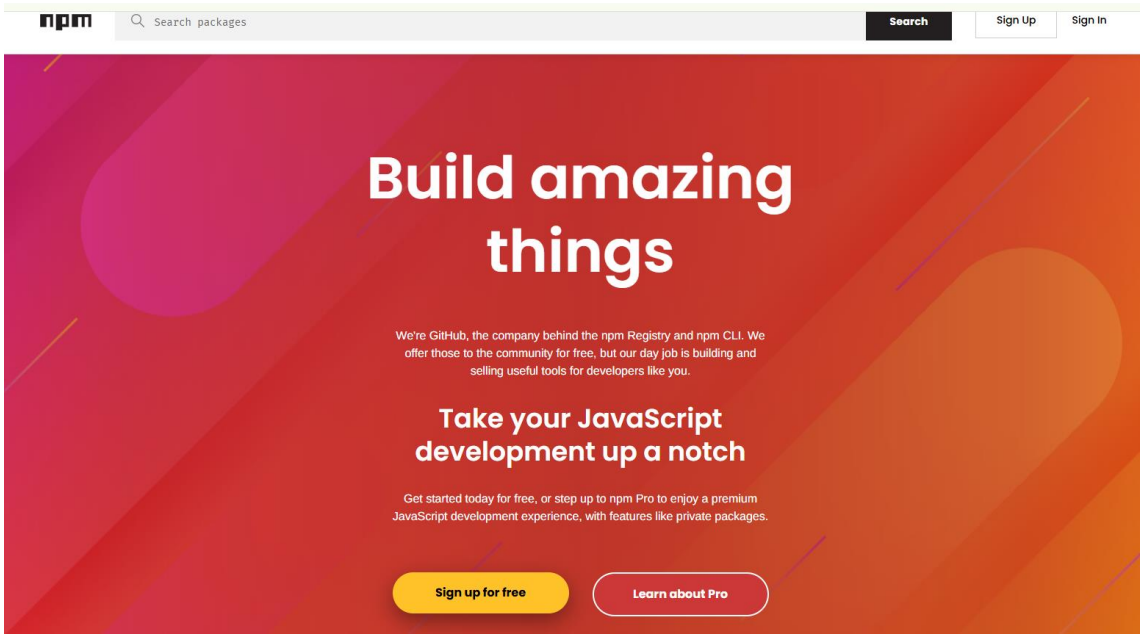
```
npm -v
```

aunque también nos servirá el comando **npm --version**.

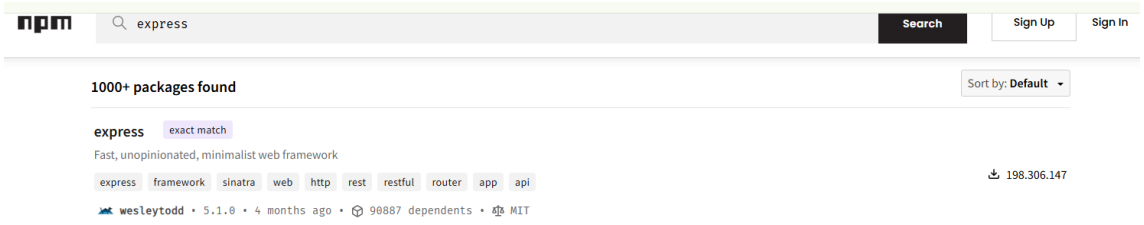
Inicialmente, npm se pensó como un gestor para poder instalar módulos en las aplicaciones Node, pero se ha convertido en mucho más que eso, y a través de él podemos también descargar e instalar en nuestras aplicaciones otros módulos o librerías que no tienen que ver con Node, como por ejemplo jQuery.

Si echamos un vistazo a la página principal de ***nodejs.org***, hay una frase que dice que npm es el mayor ecosistema de librerías open-source del mundo, gracias a la comunidad de desarrolladores que hay detrás. Esto nos permite centrarnos en las necesidades específicas de nuestra aplicación, sin tener que "reinventar la rueda" cada vez que necesitemos una funcionalidad que ya han hecho otros antes.

El registro de librerías o módulos gestionado por NPM está en la **web npmjs.com**.



Podemos consultar información sobre alguna librería en particular, consultar estadísticas de cuánta gente se la descarga, e incluso proporcionar nosotros nuestras propias librerías si queremos. Por ejemplo, esta es la ficha de la librería *express*:



La opción más habitual de uso de npm es instalar módulos o paquetes en un proyecto concreto, de forma que cada proyecto tenga sus propios módulos. Sin embargo, en algunas ocasiones también nos puede interesar (y es posible) instalar algún módulo de forma global al sistema. Veremos cómo hacer estas dos operaciones.

3.2. Instalar módulos locales a un proyecto

En este apartado veremos cómo instalar módulos de terceros de forma local a un proyecto concreto. Haremos pruebas dentro de un proyecto llamado "PruebaNPM" en nuestra carpeta de "ProyectosNode/Pruebas", cuya carpeta podemos crear ya.

3.2.1. El archivo "package.json"

La configuración básica de los proyectos Node se almacena en un archivo JSON llamado "package.json". Este archivo se puede crear directamente desde línea de comandos, utilizando una de estas dos opciones (debemos ejecutarla en la carpeta de nuestro proyecto Node):

- npm init --yes, que creará un archivo con unos valores por defecto, como éste que se muestra a continuación:

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

- npm init, que iniciará un asistente en el terminal para que demos valor a cada atributo de la configuración. Lo más típico es rellenar el nombre del proyecto, la versión, el autor y poco más. Muchas opciones tienen valores por defecto puestos entre paréntesis, por lo que si pulsamos Intro se asignará dicho valor sin más.

```
Press ^C at any time to quit.
package name: (pruebanpm)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
```

  Observad que el nombre del proyecto lo toma automáticamente de la carpeta en que se encuentra, y el archivo principal de la aplicación suele llamarse "index.js" (o "app.js" en algunos ejemplos que podéis encontrar por Internet).

Al final de todo el proceso, tendremos el archivo en la carpeta de nuestro proyecto. En él añadiremos después (de forma manual o automática) los módulos que necesitemos, y las versiones de los mismos, como explicaremos a continuación.

3.3. Instalar módulos globales al sistema

Para cierto tipo de módulos, en especial aquellos que se ejecutan desde terminal como Grunt (un gestor y automatizador de tareas Javascript) o JSHint (un comprobador de sintaxis Javascript), puede ser interesante instalarlos de forma global, para poderlos usar dentro de cualquier proyecto.

La forma de hacer esto es similar a la instalación de un módulo en un proyecto concreto, añadiendo algún parámetro adicional, y con la diferencia de que, en este caso, no es necesario un archivo "package.json" para gestionar los módulos y dependencias, ya que no son módulos de un proyecto, sino del sistema. La sintaxis general del comando es:

```
npm install -g nombre_modulo
```

donde el flag -g hace referencia a que se quiere hacer una instalación *global*.

Es importante, además, tener presente que cualquier módulo instalado de forma global en el sistema no podrá importarse c o n require en una aplicación concreta (para hacerlo tendríamos que instalarlo también de forma local a dicha aplicación).