Contenidos a Trabajar

1. Páginas Estáticas vs Páginas Dinámicas
2. ¿Por qué Node?
    a. NPM: Node Package Manager
    b. ¿Dónde debe utilizarse Node JS?
    c. ¿Dónde debe utilizarse Node JS?
3. ¿Cómo empiezo con Node.js?
4. Comentarios en Node.
5. Módulos

Páginas web estáticas vs páginas web dinámicas

Cuando se trata de desarrollar un sitio web lo primero que tenes que considerar es cómo lo querés construir, como un sitio web estático o como un sitio web dinámico. Pero, ¿qué hace que un sitio web sea categorizado “estático” o “dinámico”?

Veamos algunas diferencias para que puedas decidir cuál es el que te conviene según tu necesidad. Pero antes de entrar en detalles de cada uno de estos tipos de sitios web, primero debes entender cómo funciona la comunicación en Internet cuando queremos ver una página web.

Comunicación entre servidores y navegadores web

Para entender la comunicación más básica que ocurre en Internet, debemos saber que en ella están involucrados un servidor web, como IIS, Apache o NGINX, que contiene los archivos HTML, CSS y JavaScript de las páginas web, y un cliente, el navegador (Chrome, Firefox, Edge).

El servidor web y el cliente se comunican a través de los protocolos HTTP (Hypertext Transfer Protocol), el protocolo de Transferencia de Hipertexto y la versión segura HTTPS (HyperText Transfer Protocol Secure), una serie de reglas que permiten la transferencia de información a través de archivos en la Internet.

La diferencia principal entre HTTP y HTTPS, es que, con este último, la comunicación entre el servidor y el cliente es cifrada permitiendo que la transmisión de los datos sea segura, brindando mayor integridad y confidencialidad a los mismos.

La comunicación entre ellos inicia cuando el usuario, a través del navegador, ingresa la dirección de un sitio web (conocida como la **URL** del inglés Uniform Resource Locator ó **LRU** que se refiere al Localizador de Recursos Uniforme en español), generando una petición al servidor web para encontrar los archivos de dicha página en dónde están alojados (HTTP request). Luego, el servidor web responde a la petición y devuelve los archivos del sitio web (HTTP response). La respuesta llega al navegador como una copia en formato HTML de la página web, y es en este momento cuando el usuario puede verla en su pantalla.

¿Qué es una página web estática?

Lo primero que debemos entender es ¿a qué nos referimos con la palabra estática en el contexto de una página web?, y no es más que aquello que en el ámbito del código fuente del sitio web se encuentra fijo, no se mueve ni cambia de ninguna manera.

Cuando hablamos de "estático" también podemos referirnos a que la página web tiene un número fijo de páginas, es decir, que tal como fue diseñada y almacenada en el servidor web, así mismo la recibe el navegador y la ve el usuario, como un número fijo de páginas HTML.

Una página web estática está compuesta por archivos HTML individuales por cada página que son pre-generados y presentados al usuario a través del navegador de la misma forma.

Como una página web estática básica está compuesta por elementos como títulos, cuadros de textos, etiquetas, imágenes y otros elementos multimedia, un usuario solo puede interactuar con una página web estática a través de lo que permiten los elementos HTML, por ejemplo haciendo clic en enlaces, botones o rellenando formularios como el clásico formulario de subscripción.

No son tan complejos técnicamente como un sitio web dinámico, pero tampoco son tan versátiles y efectivos cuando se trata de entregar funcionalidad. En

pocas palabras, en una página web estática, verás la misma información, diseño y contenido cada vez que la visites, a menos que alguien aplique cambios al código fuente de forma manual.

Si quisieras crear una página web estática solo necesitas un editor de texto como el Bloc de notas y saber de HTML y CSS, no es necesario utilizar entornos de desarrollo complejos.

Ventajas de una página web estática

Entre las ventajas de una página web estática podemos mencionar:

- El costo inicial de una página web estática puede ser mucho menor que al de una dinámica.

  Por su naturaleza estática, la complejidad y tiempo de desarrollo es menor porque no requiere del uso de lenguajes de programación o bases de datos, y por ende su costo monetario es más bajo.

- Son muy flexibles cuando se trata del diseño.

  Dado a su naturaleza independiente, cada página puede tener un diseño diferente. No es necesario un solo diseño para múltiples tipos de

contenido, lo que en los sitios web dinámicos se le conoce como plantillas (templates).

- Los tiempos de carga son muy rápidos.

  Ya que los sitios web estáticos son construidos previamente. No implica ejecución de scripts o secuencias de comandos complejas, bases de datos ni análisis de contenido a través de lenguajes de plantillas, etc.

Sin embargo, con la revolución del Jamstack, los generadores de sitios web estáticos como Jekyll, GatsbyJS o Eleventy, y los Headless CMS como Netlify CMS, Siteleaf o Forestry, y además la incorporación de CDN (Content Delivery Network en inglés) para gestionar los recursos multimedia, se puede generar un aumento en el tiempo de carga de una página web estática dependiendo de sus características.

Desventajas de una página web estática

Algunas desventajas de elegir una página web estática son:

- Una página web estática puede ser más difícil de actualizar.

  Para usuarios no técnicos, una vez que la página es creada, hacer pequeños ajustes en el contenido puede representar un desafío a

menos que estén familiarizados con HTML, CSS y el código del sitio web en general. Si no es así, es posible que deban pedirle al desarrollador que la creó originalmente, que realice los cambios que necesitan.

- Agregar contenido a la página web o realizar actualizaciones puede incurrir en costos adicionales.

  Esto puede verse como una consecuencia de la desventaja anterior. Es decir que, con el tiempo, el mantenimiento de un sitio estático puede generar costos de mantenimiento continuo que podrían evitarse si tuvieras una página web dinámica.

- Agregar nuevas páginas o funcionalidades a una web estática puede ser más difícil que hacerlo para una web dinámica.

  Por ejemplo, si creas una página web para promocionar productos de tecnología, cada vez que querés agregar un producto, como un nuevo televisor o un nuevo celular, tendrías que crear una nueva página específicamente para ese producto, lo que puede llevar mucho tiempo además del costo que puede llevar este proceso.

Ejemplos de páginas web estáticas

Un ejemplo sencillo de cómo es una página web estática, es el siguiente:

```
<!DOCTYPE html>
<html>
<head>
  <title>Ejemplo página web estática</title>
</head>
<body>
  <p>La fecha de hoy es Enero 1, 2022</p>
</body>
</html>

```

Aquí, la fecha está escrita directamente en el código de la página (estática) y cada vez que se recargue la página, dirá lo mismo, Enero 1, 2022... la única forma de que cambie es si alguien actualiza el código y escribe otra fecha o aplica alguna instrucción que la haga dinámica para que la fecha sea diferente cada vez que carga.

Qué es una página web dinámica

La palabra dinámica se refiere a elementos que cambian continuamente, son interactivos y funcionales, en lugar de ser simplemente informativos. Por supuesto, eso requiere utilizar más que solo código HTML y CSS.

En comparación con las páginas web estáticas, que son mayoritariamente informativas, una página web dinámica incluye aspectos que se caracterizan por la interactividad y la funcionalidad, por ejemplo, los usuarios pueden interactuar con la información que se presenta en la página gracias a las instrucciones creadas a través de los lenguajes de programación y la base de datos sobre la que está construida.

Los sitios web dinámicos basan su comportamiento y funcionalidad en dos tipos de programación, front-end (del lado del cliente) y back-end (del lado del servidor). Las instrucciones del lado del cliente es código JavaScript que se ejecuta en el navegador. Mientras que las instrucciones que se ejecutan del lado del servidor son instrucciones escritas en lenguajes de scripting o programación, como ASP.Net, PHP, Python, etc. y que son ejecutadas para crear lo que el usuario ha solicitado en su interacción con la página. En nuestro caso utilizaremos Node.Js como lenguaje de back-end.

Una vez ejecutadas las instrucciones en el servidor, un nuevo HTTP response se envía al navegador del usuario para mostrarle lo que ha solicitado. El

resultado final es el mismo que en un sitio web estático: una página HTML que el usuario ve desde el navegador.

Por resumir, una página web dinámica puede ser más compleja cuando hablamos de su diseño y desarrollo, pero también es más versátil cuando se trata de la funcionalidad que ofrece.

Ventajas de una página web dinámica

Entre las ventajas de una página web dinámica están:

- Puede gestionar información a través de bases de datos.

  Esto permite que el usuario pueda solicitar información fácilmente de una manera organizada y estructurada dentro de un catálogo, además de crear y mostrar contenido según el tipo de usuario que acceda a la página.

- El contenido se puede gestionar a través de un CMS.

  El contenido almacenado en el CMS puede incluir una variedad de archivos, desde el texto hasta las imágenes que se muestran, diseños de página, configuraciones del sitio y más. Esto permite una flexibilidad

extrema a la hora de crear el sitio y también permite que varios usuarios puedan manipular el contenido según sea necesario.

- El coste de mantenimiento es menor.

  Si la página no necesita cambios en el diseño básico o en la funcionalidad definida al inicio de su desarrollo. Ya que se puede gestionar la información a través de un CMS (por ej: WordPress) , existe poco o nada de costes cuando se trata de su mantenimiento.

Desventajas de una página web dinámica

Algunas desventajas de una página web dinámica son:

- Pueden existir limitaciones en el diseño.

  Ya que el contenido está principalmente basado en la información contenida en la base de datos y la presentación al usuario se basa en la estructura de la misma. Esto puede hacer que el diseño sea complicado, ya que lo más sencillo es optar por un enfoque único para todas las páginas. Dependiendo del CMS, puede resultar difícil crear varios diseños o plantillas que permitan mostrar diferentes tipos de contenido de diferentes formas.

- Puede involucrar altos costos de construcción iniciales.

  Al coste del desarrollo de la página web se le suma el coste del desarrollo de las bases de datos donde se guardará el contenido a mostrar, etc. El desarrollo también puede costar más a medida que se agregan nuevas funcionalidades. Si bien los costos de mantenimiento pueden ser más bajos como fue mencionado en las ventajas, también puede involucrar costos de desarrollo iniciales mucho más altos que al desarrollar una página web estática.

Ejemplos de páginas web dinámicas

Como ya hemos visto, es muy sencillo determinar si una página web es dinámica: por ejemplo, cuando puedes interactuar con ella, o si cada vez que la recargas, puedes ver contenido distinto. Por lo tanto, la mayoría de las páginas que regularmente visitas es probable que sean dinámicas porque son interactivas.

Una página web dinámica te permite crear un perfil de usuario Facebook.com, comentar una publicación LinkedIn.com, o hacer una reserva . Siguiendo el ejemplo de la página que muestra una fecha, si queremos convertirla en una página web dinámica, podemos cambiar la fecha escrita textualmente por una función que retorne la fecha actual, de esta forma:

```
<head>
<title>Página web dinámica</title>
</head>
<body>
La fecha de hoy es <%=Datetime.Now()%>
</body>
</html>
```

Aquí, cada vez que se recarga la página, se mostrará la fecha y hora actual, es decir será diferente en cada recarga de la página, ya que la instrucción **<%=Datetime.Now()%>** le indica al servidor que retorne la fecha del momento en que recibe la petición.

Página web dinámica vs estática: Conclusión

En conclusión, si tenés que crear una página web y vas a tomar la decisión entre crear una página web estática o una dinámica, tu decisión debe ser principalmente en los objetivos que querés cumplir con tu página web y los recursos de tiempo y conocimientos que tengas disponibles. La mayoría de las personas que no poseen conocimientos técnicos de diseño y desarrollo de páginas web, prefieren los sitios web dinámicos porque a través de

plataformas CMS como WordPress, Joomla, Drupal o Ghost pueden crear sitios web dinámicos de una forma muy fácil y rápida, a la vez que son más fáciles de mantener a largo plazo.

Si bien es cierto que las páginas web dinámicas ofrecen más posibilidades, pueden ser mucho más complejas de construir y mantener para los usuarios que no tengan conocimientos técnicos y deseen incorporar integraciones que no ofrezcan los CMS; mientras que las páginas web estáticas son algo más limitadas, pero en principio son mucho más simples de crear y mantener si tenés conocimientos en HTML y CSS.

¿Por qué Node?

La creciente popularidad de JavaScript ha traído consigo varios cambios, incluyendo sobre la superficie del desarrollo web, ya que hoy en día es radicalmente diferente. Las cosas que podemos hacer en la web hoy, con JavaScript ejecutando en el servidor, como también en el navegador, eran difíciles de imaginar hace varios años, cuando algunas funcionalidades se encapsulaban dentro de entornos “sandbox” como Flash y Java.

Según el sitio oficial, “Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google.”

Más allá de eso, vale la pena señalar que el creador de Node.js, Ryan Dahl fue encargado de crear sitios web en tiempo real con función de inserción, “inspirado por aplicaciones como Gmail”. En Node.js, dió a los desarrolladores una herramienta para trabajar en el paradigma no-bloqueante, event-driven I/O. Por ende, Node.js se destaca en aplicaciones web de tiempo real empleando la tecnología push a través de Websockets.

¿Qué es tan revolucionario acerca de eso?

Bueno, después de más de 20 años de webs basadas en el paradigma de petición respuesta, finalmente tenemos aplicaciones web en tiempo real, las conexiones bidireccionales, donde tanto el cliente como el servidor pueden iniciar la comunicación, lo que les permite intercambiar datos libremente. Esto está en contraste con el paradigma de respuesta web típica, donde el cliente siempre inicia la comunicación. Además, todo se basa en el Open Web Stack (HTML, CSS y JS) que se ejecuta en el puerto estándar 80.

Podríamos argumentar que hemos tenido este formato durante años en forma de Flash y Applets de Java, pero en realidad, eran simplemente un entorno de Sandbox usando la web como un protocolo de transporte para ser entregado al cliente. Además, se ejecutan en aislamiento y a menudo operan a través de un puerto no estándar, el cual podía tener requisitos adicionales para su uso.

Con todas sus ventajas, Node.js ahora juega un papel crítico en el stack de tecnología de muchas empresas de alto perfil que dependen de sus exclusivas ventajas.

¿Cómo funciona?

La idea principal de Node.js: uso no-bloqueante, event-driven I/O, permanecer ligero y eficiente en la superficie del uso intensivo de datos en tiempo real de las aplicaciones que se ejecutan en dispositivos distribuidos.

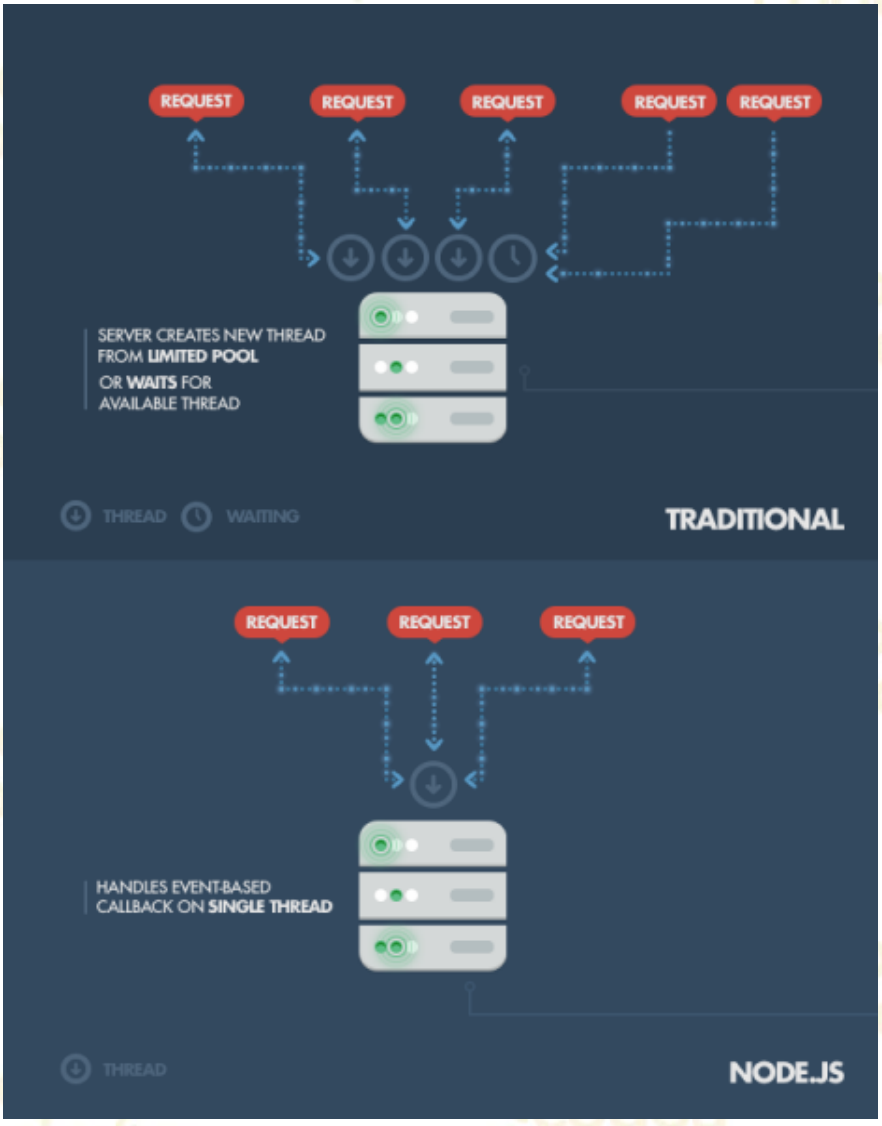
Y este entendimiento es absolutamente esencial. Definitivamente no tenés que usar Node.js para operaciones intensivas de CPU; de hecho, utilizándolo para el cálculo pesado anulará casi todas sus ventajas. Donde Node REALMENTE destaca es en la construcción rápida y escalable de aplicaciones de red, debido a que es capaz de manejar un gran número de conexiones simultáneas con alto rendimiento, lo que equivale a una alta escalabilidad.

Cómo funciona internamente es bastante interesante. Frente a las tradicionales técnicas de servicio web donde cada conexión (solicitud) genera un nuevo subproceso, retomando la RAM del sistema y finalmente a tope a la cantidad de RAM disponible, Node.js opera en un solo subproceso, no utiliza el bloqueo de llamadas de E/S, lo que le permite admitir decenas de miles de conexiones simultáneas (celebrada en el caso de loop).

Un cálculo rápido: suponiendo que cada subproceso tiene un potencial acompañado de 2 MB de memoria, el cual se ejecutará dentro de un sistema con 8 GB de RAM nos pone a un máximo teórico de 4.000 conexiones

simultáneas, además del costo de cambio de contexto entre subprocesos. Ese es el escenario que se suelen tratar con técnicas de servicio web tradicional.

Evitando todo eso, Node.js alcanza niveles de escalabilidad de más de 1M de conexiones simultáneas.



Existe por supuesto, la posibilidad de compartir un único subproceso entre todas las solicitudes de clientes, convirtiéndola en una falla potencial de escribir aplicaciones Node.js.

En primer lugar, el cómputo pesado podría estancarse y provocar problemas para todos los clientes, como las peticiones entrantes, las cuales serían bloqueadas hasta que dicho cálculo se haya completado.

En segundo lugar, los desarrolladores necesitan ser muy cuidadosos en no permitir que un error no manejado se propague hasta el núcleo de la aplicación, lo que provocaría que la instancia de Node.js se terminase.

NPM: El Node Package Manager

Cuando hablamos de Node.js, una cosa que definitivamente no debe omitirse es integrarlo en el apoyo de la gestión de paquetes utilizando la herramienta NPM que viene por defecto con cada instalación de Node.js.

La idea de los módulos NPM es muy similar a la de Ruby Gems, Pip en Python: un conjunto de componentes reutilizables disponibles públicamente a través de una fácil instalación a través de un repositorio en línea, con la versión y la dependencia de gestión.

Una lista completa de los paquetes de módulos puede encontrarse en el sitio web de NPM **https://npmjs.org/** o acceder utilizando la herramienta de la CLI de NPM que automáticamente se instala con Node.js.

NPM es un ecosistema abierto a todos y cualquiera puede publicar su propio módulo que será incluido en el repositorio de NPM.

Algunos de los más populares hoy en día son módulos de NPM:

- **express** - Express.js, inspirado en el framework de desarrollo web para Node.js, y el estándar de facto para la mayoría de aplicaciones Node.js de hoy en día.
- **connect** - Connect es un servidor HTTP extensible framework para Node.js, que proporciona una colección de alto rendimiento de plugins conocidos como middleware; sirve como fundamento para expresar.
- **socket.io** y **sockjs** - Componente del servidor de los dos componentes de websockets más comunes en la actualidad.
- **Jade** - Uno de los más populares motores de plantillas, inspirados por HAML, un defecto en Express.js.
- **mongo y mongojs** - mongoDB wrappers para proporcionar la API para bases de datos de objetos MongoDB en Node.js.
- **redis** - Redis biblioteca cliente.

La lista es interminable. Hay toneladas de paquetes realmente útiles y disponibles para todos.

Ejemplos en donde Node.js debe utilizarse:

Chat

Es la forma más típica en tiempo real y una multi-aplicación de usuario. Desde IRC , a través de muchos propietarios y protocolos abiertos girando en puertos no estándar, con la capacidad de instrumentar todo en Node.js con websockets corriendo sobre el puerto estándar 80.

La aplicación de chat es realmente perfecta para Node.js: es ligera, tiene un alto tráfico de datos intensivos (pero bajo procesamiento de cómputo) y es una aplicación que funciona en dispositivos distribuidos. También es un gran caso de uso para el aprendizaje, ya que es demasiado simple, pero al mismo tiempo que cubre la mayoría de herramientas que podés utilizar en una típica aplicación Node.js.

PROXY

Node PROXY.js es empleado como un servidor proxy el cual puede manejar una gran cantidad de conexiones simultáneas en un modo de no-bloqueo. Es

especialmente útil para proxy de diferentes servicios con distintos tiempos de respuesta, o para la recopilación de datos desde varios puntos de origen.}

Un ejemplo: considere una aplicación de servidor que se comunica con recursos de terceros, extrayendo datos de diferentes fuentes, o almacenando los activos como imágenes y vídeos a servicios terceros de Cloud.

Node.js con Express.js también pueden ser utilizados para crear aplicaciones web clásicas en el servidor. Sin embargo, este paradigma no es el más típico de los casos de uso. Hay argumentos para estar a favor y en contra de este enfoque.

Aquí están algunos hechos a considerar:

PROS:

Si tu aplicación no tiene ningún cálculo intensivo del CPU, podés construir en Javascript de arriba a abajo, inclusive a nivel de base de datos si utilizas el objeto de almacenamiento JSON como MongoDB DB. Esto facilita el desarrollo (incluyendo la contratación) significativamente.

Los Crawlers reciben una respuesta totalmente HTML, que es mucho más SEO-friendly, digamos, una sola página o en una aplicación de Websockets app se ejecuta sobre Node.js.

CONS:

Un CPU de cálculo intensivo bloqueará la receptividad del Node.js, por lo que una plataforma de roscado es un mejor enfoque. Alternativamente, podrías intentar escalar el cómputo.

Donde Node.js no debe usarse

Cuando se trata de cómputo pesado, Node.js no es la mejor plataforma. Definitivamente no querés construir un servidor de cálculo Fibonacci en Node.js. En general, cualquier operación de uso intensivo de CPU anula todas las ventajas de rendimiento y bloquearía cualquier petición entrante de un subproceso.

Como se dijo anteriormente, Node.js es Single-threaded y utiliza un único núcleo del CPU. Cuando se trata de la adición de la concurrencia en un servidor multi-core, hay algunos trabajos realizados por el Node básico en la forma de un módulo cluster [ref: http://nodejs.org/api/cluster.html]. También podés ejecutar varias instancias del servidor Node.js bastante fácil detrás de un proxy inverso a través de nginx.

Conclusión

Node.js nunca fue creado para resolver el problema de escalado de computación. Fue creado para resolver el problema de escalado de E/S (entrada/salida), lo que lo hace muy bien.

¿Por qué usar Node.js?

Si el caso de uso no contiene operaciones intensivas del CPU ni el acceso a los recursos que pudieran generar bloqueos, podés aprovechar los beneficios de Node.js y disfrutar de aplicaciones de red rápidas y escalables.

Inicialmente uno de los puntos que se deben tener en claro es que Node.JS por definición es un entorno de ejecución para JavaScript. Así mismo, sus características son aquellas que hacen que sea tan interesante a la hora de utilizarlo ya bien sea para un servicio web, una API Rest o cualquier herramienta a nivel de batch.

Anteriormente, los desarrolladores de JavaScript sólo podían utilizar este lenguaje con la obligación de utilizar un navegador web ya sea Firefox, Chrome, entre otros. Lo que ocasionaba que se tuviera una limitación a la hora de realizar cierto tipo de aplicaciones, ya que no se podían generar o programar aplicaciones que se renderizaran en el servidor.

Con la llegada de Node.JS, se abrió un nuevo mundo y empezaron a surgir los servidores web hechos con Express o con otras librerías basadas en Node, las de API Rest, incluso se abrió un nuevo mundo a la hora de desarrollar para IOT (internet of things). Por ejemplo: las placas arduino, ya que éstas se pueden desarrollar con Node en una aplicación y utilizarlas en ese tipo de placas.

Con lo cual, se puede decir que Node tiene una cantidad considerable de características entre las cuales destacan las siguientes:

- Desarrollo en JavaScript: Para desarrollar en Node se realiza a través del lenguaje de programación JavaScript, que actualmente está teniendo popularidad y mejoras permitiendo el desarrollo tanto para frontend como para backend, abriendo el camino a los profesionales fullstack.

- Basado en el motor V8 de Chrome: Es uno de los motores más avanzados a nivel de JavaScript ya que se mantiene actualizado con las nuevas funcionalidades del estándar ECMAScript.

- Operaciones de E/S sin bloqueos: Node está pensado para que las operaciones de entrada y salida sean sin bloqueos, por ejemplo: un servidor web realiza una petición única y espera una respuesta.

- Orientado a eventos (POE): Para comprender esta característica, pensemos en un bus de datos cuando una porción de código realiza una

operación, esta pública un evento, ese evento en otra instancia (en otro momento del tiempo) lo recibe otro trozo de código y hace otra acción con él; en este punto de hecho, se habla mucho del término asincronía del tema de ajax. Por ejemplo, a la hora de hacer peticiones a un servicio web externo una API Rest la sincronía es una consecuencia de la orientación a eventos, Node.JS funciona perfectamente con temas de asincronía y es una muy buena opción si queremos hablar de códigos asíncronos que queramos hacer para nuestra aplicación.

- Liviano y Eficiente: En resumen por todo lo anteriormente mencionado (entradas y salidas sin bloqueo, desarrollado bajo JavaScript, basado en V8, orientado a eventos y el tema de la asincronía) hace que Node.JS sea liviano (pesa muy poco) y a su vez sea muy eficiente en los casos de gestión de eventos, orientación a eventos y los casos de entrada y salida.

Node.JS es una buena opción para desarrollar cierto tipo de cosas como lo son:

- Servidores Web: Con el uso de librerías que se encuentran en los paquetes propios de Node.JS o de terceros como Express, Koa, Hapi y Nest.

- Sockets: Son eventos que para realizar chats y aplicaciones en tiempo real es una excelente opción, sobretodo gracias a su gran velocidad.
- IOT: Programar placas pequeñas con poco hardware como un Arduino, permite desarrollar una aplicación y desplegarla.

¿Cómo empiezo con Node.js?

Una vez que hayamos instalado Node.js, construyamos nuestro primer servidor web. Creá un archivo llamado app.js que contenga el siguiente contenido:

```
const http = require('http');
const hostname = '127.0.0.1';
const port = 3000;

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

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

Ahora, ejecutalo en tu servidor web usando

node app.js

Visitá http://localhost:3000 en tu navegador web y verás un mensaje que dice

"Hola mundo".

Comentarios en Node.Js

No tenemos que olvidarnos que Node es JavaScript, por ende, permite insertar comentarios en el código, al igual que la mayoría de los lenguajes de programación y puntualmente que JS. En concreto hay dos tipos de comentarios permitidos, los comentarios en línea que comienzan con una doble barra: //, y los comentarios multilínea, que comienzan con /* y terminan con */.

Módulos

Importando y creando módulos

Los módulos son bloques de código reutilizables y organizados que encapsulan la funcionalidad de una aplicación. Permiten dividir el código en partes más pequeñas y manejables, lo que facilita el mantenimiento, la reutilización y la colaboración en proyectos.

Node.js utiliza el sistema de módulos CommonJS, que proporciona una forma estándar de definir, importar y exportar módulos. Cada archivo de JavaScript en Node.js se considera un módulo por defecto, lo que significa que el código dentro de ese archivo está contenido en ese ámbito y no afectará a otros módulos a menos que se exporten específicamente ciertos elementos.

Para exportar elementos desde un módulo en Node.js, se utiliza el objeto **module.exports** o la sintaxis de **exports**. Por ejemplo:

```
// módulo.js
const mensaje = "¡Hola desde el módulo!";

function saludar() {
  console.log(mensaje);
}

module.exports = {
  saludar: saludar
};
```

Para importar y utilizar un módulo en otro archivo, se utiliza la función **require()**. Por ejemplo:

```
// app.js
const modulo = require('./modulo');

modulo.saludar(); // Imprime "¡Hola desde el módulo!"
```

Los módulos en Node.js también pueden tener dependencias de otros módulos. Cuando se importa un módulo, Node.js busca automáticamente ese módulo en el sistema de archivos y lo carga. Esto permite estructurar una aplicación Node.js en una jerarquía de módulos interconectados.

Módulos Internos

Node.js proporciona una amplia gama de módulos internos (core modules) que ofrecen funcionalidades listas para usar, como **“http”** para la creación de servidores web, **“fs”** para la manipulación de archivos, **“path”** para la manipulación de rutas de archivo, entre otros. Estos módulos internos se pueden utilizar directamente sin necesidad de instalar paquetes adicionales.

Módulos Externos

Además de los módulos internos, Node.js cuenta con un vasto ecosistema de módulos externos disponibles en el repositorio npm (Node Package Manager).

El repositorio **npm** contiene miles de módulos de código abierto que abarcan desde utilidades generales hasta frameworks y librerías especializadas. Estos módulos externos se pueden instalar y utilizar en proyectos de Node.js a través del comando npm install o yarn add, y se pueden importar y utilizar de manera similar a los módulos internos.

ESModules

ESModules (también conocidos como ECMAScript modules o ESM) son un sistema de módulos nativo de JavaScript que se introdujo en la especificación de ECMAScript 6 (ES6). Proporcionan una forma estándar de definir, importar y exportar módulos en el lenguaje JavaScript.

A diferencia de CommonJS, que es utilizado en Node.js, los ES modules son compatibles directamente en los navegadores web modernos y en entornos que admiten ES6. Esto significa que se pueden utilizar tanto en el lado del cliente (front-end) como en el lado del servidor (back-end) sin necesidad de transpilación o herramientas adicionales.

Las principales diferencias entre ES modules y CommonJS son:

1. **Sintaxis**: ES modules utilizan una sintaxis basada en palabras clave como import y export para importar y exportar elementos. Por otro lado, CommonJS utiliza la asignación de objetos (module.exports y require) para exportar e importar elementos.

2. **Comportamiento asincrónico**: los ES modules se cargan de forma asincrónica, lo que significa que las importaciones se resuelven dinámicamente en tiempo de ejecución. En cambio, CommonJS se carga

de forma sincrónica, lo que significa que las importaciones se resuelven estáticamente en tiempo de compilación.

3. **Ámbito**: Los ES modules tienen un ámbito propio por archivo, lo que significa que las variables y funciones declaradas dentro de un módulo no se filtran al ámbito global. En CommonJS, las variables y funciones declaradas en un módulo están disponibles en el ámbito global.

4. **Exportación e importación estática**: ES modules solo permiten exportaciones e importaciones estáticas en la parte superior del archivo. No se pueden realizar exportaciones o importaciones condicionales o dentro de bloques de código. CommonJS permite exportaciones e importaciones dinámicas en cualquier parte del archivo.

Veamos algunos ejemplos de ESModules:

- Exportar e importar funciones:

```
// modulo.js
export function saludar() {
  console.log('¡Hola desde el módulo!');
}

// app.js
import { saludar } from './modulo.js';

saludar(); // Imprime "¡Hola desde el módulo!"
```

- Exportar e importar variables:

```
// modulo.js
export const mensaje = '¡Hola desde el módulo!';

// app.js
import { mensaje } from './modulo.js';

console.log(mensaje); // Imprime "¡Hola desde el módulo!"
```

- Exportar e importar elementos por defecto:

```
// modulo.js
export default function saludar() {
  console.log('¡Hola desde el módulo!');
}

// app.js
import saludar from './modulo.js';

saludar(); // Imprime "¡Hola desde el módulo!"
```

En conclusión, ES modules y CommonJS son sistemas de módulos utilizados en JavaScript con diferencias en la sintaxis, comportamiento y compatibilidad.

La elección entre uno u otro dependerá del entorno de desarrollo, el ecosistema de paquetes y las características específicas requeridas en el proyecto.