# Fundamentos de Node.js

## Prework

### Conocer qué es Node.js

#### Conocer qué es Node.js.

##### Node.js es un entorno de ejecución de JavaScript creado en el año 2010, dando la posibilidad de ejecutar código fuera del navegador. Su uso principal es para la creación de servidores (lo que conocemos hoy en día como Back-end) pero también puede utilizarse para Aplicaciones Móviles, Aplicaciones de Escritorio, Internet de las Cosas, etc.

##### Node.js es una tecnología extremadamente popular hoy en día, teniendo un aumento en el número de vacantes disponibles en el ámbito laboral. Es ideal para crear funciones a tiempo real como podrían ser un chat o un sistema de notificaciones en vivo.

#### ¿Qué es node.js?

##### Como ya lo habíamos mencionado anteriormente Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Usa un modelo de concurrencia para manejar operaciones de Entrada y Salida (E/S) sin bloqueo y orientado a eventos, que lo hace liviano y eficiente. El ecosistema de paquetes de Node.js, NPM, es el ecosistema más grande de librerías de código abierto en el mundo.

#### Recuerda

##### En Windows los comandos tienen diversas variaciones. Si tienes problemas basta con googlear para buscar su equivalente.

### ¿Para qué sirve node.js?

#### CHATS

##### Debido a la naturaleza asíncrona y orientada a eventos de node.js, este es ideal para desarrollar chats. Si lo piensas estas son las características que necesita un chat para ser funcional.

##### Tanto el servidor como el cliente estarán siempre comunicados y en cuanto uno de los usuarios escriba un mensaje, Node.js reaccionaría al evento y de manera rápida comunicaría el mensaje al otro usuario.

#### REDES SOCIALES

##### Vivimos en un mundo dominado por las Redes Sociales, cada día hay más y más y son millones las personas las que las utilizan a diario. Nunca sabes si puedes crear la próxima Red Social exitosa, ya sea mundialmente o en su nicho y por supuesto es una muy buena idea crearla con Node.js.

##### Una red social debe funcionar a tiempo real y en ese aspecto Node.js es el rey.

#### CUALQUIER COSA QUE TE PUEDAS IMAGINAR

##### Si bien hemos mencionado la importancia de Node.js en implementaciones que conllevan tiempo real, eso no limita a que puedas implementar desde una página web hasta una tienda en línea.

### ¿Qué es un entorno de ejecución?

#### Un entorno de ejecución (run time environment en inglés) es un estado de máquina virtual que suministra servicios para los procesos de un programa de computadora que se está ejecutando. Puede pertenecer al mismo sistema operativo, o ser creado por el software del programa en ejecución.

#### Es decir, que node.js lo que hace es proporcionar servicios extra a JavaScript mientras este se está ejecutando.

### Características de node.js

#### ¿Qué es V8?

##### V8 es un motor open-source escrito en C++ para compilar JavaScript y WebAssembly en código máquina. Esto quiere decir que traduce JavaScript a un código puramente digital capaz de ser interpretado por la CPU donde se ejecuta.

##### Este motor fue desarrollado por Google para Google Chrome y su primera versión vio la luz en 2008 junto con la primera versión del navegador.

##### Además de Chrome, el resto de navegadores basados en Chromium también usan este motor como, por ejemplo, en el nuevo Microsoft Edge. El motivo por el que este motor es tan eficiente y rápido es precisamente porque compila el código JavaScript en lugar de interpretarlo, lo cual baja drásticamente su tiempo de ejecución.

##### Además de esto, V8 se encarga de gestionar el call stack, que es la pila de ejecución; el memory heap, que es la zona de almacenamiento dinámica donde se almacenan las variables declaradas en los bloques y el garbage colector para limpiar y liberar espacio. Por último, proporciona todos los tipos de datos, operadores, objetos y funciones.

#### V8 y node.js

##### Una característica del motor V8 es que es independiente del navegador en el que está alojado. Esta característica fue clave en el rápido auge de Node.js.

##### V8 fue elegido para ser el motor que impulsó Node.js en 2009. A medida que la popularidad de Node.js creció, V8 se convirtió en un gran motor para ejecutar JavaScript del lado del servidor.

#### Tamaño y velocidad

##### Gracias a la compresión de punteros es capaz de reducir en un 40% el uso de memoria. En la práctica, esto se traduce en una carga más rápida de las webs. Estos son los números de las pruebas que ha hecho el equipo de V8 sobre webs reales: 📊

###### prueba

#### JavaScript en servidor/backend

##### JavaScript siempre ha sido conocido por ser el lenguaje de programación web que actúa en el lado del cliente (Frontend), nunca ha podido interactuar por el lado del servidor (Backend). Para esta labor siempre se han encargado otros lenguajes como por ejemplo PHP, Python o Ruby.

##### Pero todo esto ha cambiado con Node.js. Ahora JavaScript puede actuar por los dos lados, tanto para el lado del cliente como para el del servidor, pudiendo así encargarse de todo por sí mismo

#### Modelo asíncrono orientado a eventos

##### Estamos muy acostumbrados a que cuando se realiza algún cambio en la página para visualizarlo tenemos que volver a recargar dicha página.

##### Por ejemplo; si se cambia el texto de este post mientras tú estás leyendo, hasta que tú no recargues la página no serías capaz de ver los cambios, y por supuesto, nadie te avisará de ello.

##### Node.js funciona en tiempo real y por ello, cada vez que ocurra algún evento este será modificado inmediatamente de forma que el cliente aun estando dentro de la página vea el cambio.

##### Esta es de hecho una de las características principales y uno de los motivos por los cuales Node.js se utiliza tanto hoy en día. Se podría considerar como la programación del futuro, una programación más eficiente, sólida e interactiva.

#### E/S sin bloqueos

##### JavaScript siempre ha tenido el problema de ser secuencial, pero las cosas han cambiado con Node.js.

##### Pero… ¿Qué significa esto de Entrada y Salida sin Bloqueos? Básicamente que las operaciones que deba realizar el entorno se pueden hacer en modo multitareas. Es decir, que, si hay que realizar 3 procesos, por ejemplo, estos tres se realizaran a la vez en lugar de ir de uno en uno.

##### ¿Y esto sirve para algo? ¡Pues claro! Mejorará muchísimo la velocidad de procesado, imagínate que tienes que realizar en una función 5 procesos y que cada uno tarda 1 segundo en procesarse. Si los procesos se realizarán de uno en uno, nos llevaría un total de 5 segundos en realizarlos todos, pero en su lugar tardará solo 1 segundo, ya que los cinco se procesarán simultáneamente.

##### Esto obviamente tiene sus límites, pero sin ninguna duda convierte a Node.js en un entorno muy eficiente.

#### Incluye NPM

##### Si eres desarrollador web, sobre todo en la parte del Frontend, seguramente ya conozcas NPM. NPM es un gestor de paquetes para JavaScript. Es una especie de Maven (si usas Java te sonará) para paquetes JavaScript, es decir, sirve para instalar y gestionar versiones de paquetes y librerías JS.

### ¿Qué es NPM?

#### ¿Qué es NPM?

##### npm es el gestor de paquetes que viene incluido en la instalación de Node.js, así que para instalarlo simplemente tendrás que instalar NodeJS.

##### Pero… ¿Qué es esto de un gestor de paquetes? Para los que hayan trabajado con el sistema de LINUX ya sabrán de sobras que es todo esto, pero para los que no… Un gestor de paquetes es un conjunto de utilidades o herramientas diseñadas para mejorar algo, en el caso de npm son mejoras para el entorno de Node.js.

#### ¿Para qué sirve npm?

##### Como hemos visto NPM es un gestor de paquetes, eso quiere decir que puede:

###### Descargar librerías js.

###### Actualizar en caso de nueva versión las librerías instaladas.

###### Descargar una versión en específico de la librería.

###### Gestionar las dependencias entre paquetes.

##### Una de las ventajas de npm es que todos los paquetes la descarga de un repositorio de paquetes llamado npmjs.

##### NPM usa un fichero especial llamado package. json en el que se declaran las librerías y sus versiones. Esto es muy útil ya que puedes tener este archivo con todas las librerías que necesites para que con un simple comando se descarguen todas y no tengas que estar buscándolas en sus respectivos repositorios.

##### Lo bueno es que al repositorio de librerías puedes subir las tuyas propias de tal forma que si actualizas la librería y la subes al repositorio, con un comando dentro del proyecto en el que se usa se actualiza a la nueva versión.

##### .

## Work

### Bienvenidos al mundo del Back-end.

#### Node.js es un entorno de ejecución que nos permite correr JavaScript del lado del servidor, en este programa podrás entender cómo funciona, sus bondades y lo poderoso que puede llegar a ser.

#### Objetivos

##### Darás el primer paso en el ecosistema NodeJS. Esta sesión comprende la instalación y uso básico de NodeJS, así como algunos datos sobre la estructura de un nuevo proyecto.

#### Tabla de Contenidos

##### Ejemplo 1

###### Descargar e instalar la versión recomendada de Nodejs para tu sistema operativo.

##### Ejemplo 2

###### Conocer la línea de comandos de Node.js para ejecutar archivos con extensión .js.

###### Creamos un archivo llamado index.js en nuestra carpeta de trabajo (puedes utilizar la terminal si lo deseas). Después vamos a copiar el siguiente código que es la implementación del problema FizzBuzz:

function fizzBuzz() {

for (let i = 1; i <= 100; i++) {

if (i % 3 === 0 && i % 5 === 0) {

console.log('Fizzbuzz');

} else if (i % 3 === 0) {

console.log('Fizz');

} else if (i % 5 === 0) {

console.log('Buzz');

} else {

console.log(i);

}

}

}

fizzBuzz();

###### Ahora guarda el archivo, abre tu terminal y ejecuta el comando node index.js. Verás que en tu terminal aparecerá el resultado de ejecutar el programa.

##### Ejemplo 03

###### Conocer y trabajar con módulos del core y módulos de terceros.

###### Módulos del Core

Node.js tiene módulos que ofrece de manera nativa, es decir, no requiere instalación de nada más para ser utilizados (los módulos compilados se encuentran en la instalación de Nodejs en la carpeta lib/), a estos se les conoce como módulos del Core.

Node ofrece una gran cantidad de módulos, podemos encontrarlos en su documentación:

###### ENTENDAMOS LOS MODULOS

Son herramientas o grupo de funciones prefabricadas que nos facilitan la construcción de nuestro código.

###### ¿COMO SE USAN?

Imaginemos que tenemos que ir a la escuela, y nuestra mochila es nuestro documento de JavaScript "mochila.js", ahora imaginemos que cada materia es un módulo, y ese día tendremos que ver 3 materias, matemáticas, física y química. Entonces... yo necesito meter cada libro en la mochila para poder llevar esas materias.

Ahora pensemos que la manera de realizar la acción de meter los libros a la mochila requiere un comando, y ese comando es el siguiente :

var MATEMATICAS = requiere('LIBRO\_DE\_MATEMATICAS');

var FISICA = require('LIBRO\_DE\_FISICA');

var QUIMICA = require('LIBRO\_DE\_QUIMICA');

Entonces en mi archivo mochila.js ya tengo 3 variables que cada una está cargando una materia.

Ahora si una materia es igual a un módulo, esto quiere decir que tengo 3 módulos cargados en mi archivo, si lo pasamos al mundo de node, eso quiere decir que puedo cargar los módulos mandándolos llamar con esta sintaxis:

var VARIABLE\_DONDE\_SE\_GUARDARA = require('NOMBRE\_DEL\_MODULO');

Básicamente la función "require()" es la que se encarga de cargar los módulos en nuestro archivo, a este proceso se le conoce como : INYECCION DE DEPENDENCIAS.

Existen 3 tipos de módulos que podemos utilizar en nuestros proyectos:

MODULOS INTEGRADOS DE NODE.JS

MODULOS DE TERCEROS NPM

MODULOS CREADOS

###### MODULOS INTEGRADOS DE NODE.JS

Estos módulos ya vienen integrados en node.js no requieren ningún tipo de instalación, solo los mandamos llamar directo y podremos hacer uso de ellos.

Si revisamos la lista de módulos, encontraremos el módulo OS, este módulo nos permite revisar los datos del hardware de nuestra computadora, como uso de CPU, memoria, cuantos Cores tiene de procesador etc.

Si yo quiero saber cuántos CPU tiene mi computadora, con el módulo de OS, el primer paso sería cargar el módulo en mi archivo

var os = require('os');

El siguiente paso sería revisar la documentación para determinar cuál función del módulo OS es la que me permite ver los CPUS (Procesadores)

podemos observar que para saber cuántos CPU tengo, se usa la función cpus()

lo siguiente entonces es llamar la función cpus() y almacenaremos en una variable que llevara el nombre de "misCpu" también imprimiré la variable en consola para ver el resultado final:

--- EJEMPLO COMPLETO ---

var os = require('os');

var misCpu = os.cpus();

console.log(misCpu);

###### USANDO MODULOS DE TERCEROS CON NPM

Node.js tiene un número limitado de módulos que nos ofrece de manera nativa como lo vemos en la documentación, por ello npm viene a solucionarnos la vida, ya que, en sus arcas, podemos encontrar millones de módulos para hacer cualquier cosa que queramos, pero a diferencia de los módulos de node.js, los de npm requieren una serie de procesos para incorporarlos a nuestros proyectos.

## Postwork

### Desarrollo

#### Objetivo

##### Entender los usos de npm y porque publicar módulos

#### Publicar un modelo en Npm

##### Necesitas una cuenta y también es importante mantener un módulo actualizado para mantenerlo estable y resolver los issues reportados por los usuarios. Este articulo mencionan como subir versiones nuevas

#### Automatizar subida da de versiones

##### La liga no funciona

# Módulos Esenciales

## Prework

### Objetivo

#### Manipular Archivos con Node.js

### Desarrollo

#### Módulos Esenciales de node.js

##### modelos Esenciales

###### Como se ha mencionado anteriormente con la instalación de node se incluyen módulos que llamamos a Módulos esenciales y pueden ser utilizados en cualquier parte del código

##### Manipulación de archivos

###### Con node podemos manipular archivos de manera sencilla ya que sea subir editar eliminar y crear etc. Un archivo con el servidor.

###### Por otro lado, también puedes manipular las rutas de archivos obtener respuesta cuando realices una determinada acción con un archivo en tu servidor

##### Importación del módulo FS

###### Igual que con otros módulos de Node hay que procesar el correspondiente requiere para tener disponibles las funciones de acceso al sistema de ficheros

Sistema de ficheros son sectores y clursters dentro de nuestro sistema en otras palabras es una estructura de datos que se encarga de organizar los bloques y sectores en una partición de memoria de almacenamiento masivo para guardar archivos

###### Se encuentran en el module llamado FS y lo vamos a importar con el siguiente código

Cost fs = requiere ('fs')

##### Recuerda que el nombre del objetivo para operar con el sistema de archivo lo hemos guardado en una variable llamada Fs podrías dar el nombre variable que tu desees observa demás que en vez de var estamos usando const

#### Crear archivo

##### Crear archivo

###### El método fs.ApendFile() se usa para agregar asíncronamente los datos dados a un archivo se crea un nuevo archivo si no existe. El parámetro de opciones se puede usar para modificar el comportamiento de la operación

###### Sintaxis

fs.appendFile(path,data,[options],callback)

###### Parámetros

Este método acepta 4 parámetros

PATH

Es una cadena, buffer, url o un número del archivo fuente o el sector del archivo al que se agregara.

DATA

Es un string o buffer que denota los datos que se deber agregarse.

OPTIONS

Es una cadena o un objeto que se puede usar para especificar parámetros opciones que afectaran la salida.

Tiene 3 parámetros opcionales

Encoding – es una cadena que especifica la codificación del archivo el valor predeterminado es UTF 8

Mode – es un numero entero que especifica el modo del archivo. El valor default es 0o666

Flag es una cadena que especifica la manera utilizada mientras se agra al archivo. El default es A.

CALLBACK

Es una función que se llamaría cuando se ejecuta el método

###### Ejemplo

fs.appendFile('primer\_Archivo.html', 'primerArchivo', (err) => {

if (err) throw err;

console.log('Archivo creado satisfactoriamente');

});

##### Recuerda

###### Si el archivo existe será remplazado por en luego archivo creado.

#### Leer archivo

##### El método Fs readFile() es un método incorporado que se utiliza para leer el archivo. Este método lee todo el archivo en el buffer

##### Sintaxis

###### Fs.readFile(filename,endcoding,callback\_function)

##### **Parametros –** el metodo accepta 3 parametros

###### ilename: Contiene el nombre del archivo a leer o la ruta completa si se almacena en otra ubicación.

###### codificación: Contiene la codificación del archivo. Su valor predeterminado es utf8

###### callback\_function: Es una función de devolución de llamada que se llama después de leer el archivo. Se necesitan dos parámetros:

err: Si se produjo algún error.

data: Contenido del archivo.

###### Return Value: Devuelve los contenidos / datos almacenados en el archivo o si hubo algún error.

##### Ejemplo

###### fs.readFile('documentos/primer\_Archivo.html', (err, data) => {

###### if (err) throw err;

###### console.log(data);

###### });.

#### Actualizar Archivo

##### Podemos actualizar el contenido de un archivo con la función fs.writeFile() , también puedes usar fs.appendFile() según lo que creas te sea conveniente, primero le indicamos que archivo es el que debe actualizar y luego le pasamos el contenido que va a actualizar en el archivo.

##### Ejemplo

###### fs.writeFile('segundo\_Archivo.html', 'segundoArchivo', (err) => {

###### if (err) throw err;

###### console.log('Archivo actualizado Satisfactoriamente');

###### });.

#### Renombrar Archivo

##### El método fs.rename() se utiliza para cambiar el nombre asincrónico de un archivo en la ruta anterior dada a una ruta nueva dada. Sobreescribirá el archivo de destino si ya existe.

##### Sintaxis

###### fs.rename (oldPath, newPath, callback)

##### Parameters: Este método acepta tres parámetros como se mencionó anteriormente y se describe a continuación:

###### oldPath: Contiene la ruta del archivo que debe renombrarse. Puede ser una cadena, búfer o URL.

###### newPath: Contiene la nueva ruta de acceso al nombre del archivo. Puede ser una cadena, búfer o URL.

###### callback: Es la función que se llamaría cuando se ejecuta el método. Tiene un argumento opcional para mostrar cualquier error que ocurra durante el proceso.

##### Ejemplo

###### fs.rename('/documento/segundo\_Archivo.html', '/documento/tercer\_Archivo.html', (err) => {

###### if (err) throw err;

###### console.log('Archivo renombrado');

###### });

#### Eliminar Archivo

##### El método fs.unlink () se usa para eliminar un archivo o enlace simbólico del sistema de archivos. No funciona en directorios, por lo tanto, se recomienda usar fs.rmdir () para eliminar un directorio.

##### Sintaxis

###### fs.unlink( path, callback )

##### Parámetros: este método acepta dos parámetros como se mencionó anteriormente y se describe a continuación:

###### path: Es una cadena, búfer o una URL que representa el archivo o enlace simbólico que debe eliminarse.

###### callback: Es una función que se llamaría cuando se ejecuta el método.

err: Es un error que se lanzaría si el método falla.

##### Ejemplo

###### fs.unlink('/documentos/segundo\_Archivo.html', (err) => {

###### if (err) throw err;

###### console.log('Archivo eliminado satisfactoriamente');

###### });.

#### Como crear un servidor Web en node js

##### Que es un Servidor

###### Un servidor es un ordenador que esta al servicio de otros ordenadores dispositivos móviles, y personas a que le da información

##### Tipos de servidores

###### Servidor de correo

Tambien es conocido como mail server, es un ordenador que envia recibe y guarda los mensajes

###### Servidor proxy

Este tipo de servidor actúa como intermedario en tre es servidor y el cliente. De este modo el servidor no conoce la id del cliente. Se usa para mejorar la privdacidades del usario

###### Servidor FTP

Se utiliza para enviar arvhivosd de una maquina a otra o para descárgalos desde el servidor

###### Servidor bases de datos

Ofrese cervios de almancenmiento y gestión de base de datos de sus clientes. Perminte guardar grandes cantidades de información

###### Servidor web

Guarda todos los archivos propios de una pagina web como imagen video texto, y los muestra los clientes atraves de los navegadores.

Utiliza protocoles de HTTP hyper text transfer protocol

##### Cuando visualizamos una página web en un navegador, está realizando una solicitud a otro equipo en Internet, que a continuación proporciona la página web como respuesta. Esa computadora con la que está hablando a través de Internet es un servidor web. Un servidor web recibe solicitudes HTTP de un cliente, como su navegador, y proporciona una respuesta HTTP, como una página HTML o JSON desde una API.

##### Para que un servidor devuelva una página web, se emplea una gran cantidad de software. Este software generalmente se divide en dos categorías: frontend y backend. El código front-end se refiere a cómo se presenta el contenido, como el color de la barra de navegación y el estilo de texto. El código back-end se encarga de la forma en la que los datos se intercambian, procesan y almacenan. El código que administra las solicitudes de red desde su navegador o se comunica con la base de datos lo gestiona principalmente el código back-end.

##### Node.js permite a los desarrolladores usar JavaScript para escribir código back-end, aunque tradicionalmente se usaba en el navegador para escribir código front-end. Al tener el frontend y el backend juntos de esta forma, se reduce el esfuerzo necesario para crear un servidor web, que es el motivo principal por el que Node.js es una opción popular para escribir código back-end.

#### Crear un servidor http básico

##### Si nos vamos a la definición de HTTP podríamos afirmar que es un protocolo de red. Ahora bien, ¿Qué es un protocolo de red?

##### Un protocolo de red se define como un conjunto de reglas y permisos para establecer la comunicación entre dispositivos. Además se encarga de la configuración de la conexión, es decir cómo y de qué forma se va a actuar.

##### HTTP (Protocolo de Transferencia de Hipertexto) es el protocolo por excelencia de la web. Se utiliza para todo tipo de transacción, como por ejemplo que el navegador pida una página o recurso a un servidor y que éste responda.

##### Por lo tanto, como se muestra en el gráfico, el cliente, que puede ser nuestro navegador web, envía una petición por la URL, el servidor procesa ese pedido y devuelve la respuesta correspondiente.

##### En el caso de Node.Js, debemos configurar manualmente su comportamiento, es decir, no es como Php+Apache que ya trae un conjunto de configuraciones y comportamientos previos listos para usar.

##### Recuerda

###### En Node.Js tenemos la tarea de configurarlo por nuestra cuenta .

##### Lo primero que vamos a hacer es crear un archivo con el nombre de app.js y, una vez dentro, vamos a requerir un módulo del core de Node llamado http.

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

##### http es un módulo que ya viene con node, por lo tanto no debemos instalarlo con npm o otro gestor.

##### Luego crearemos dos variables, una para el host y otra para el puerto, ya que el servidor necesitará de estos datos.

###### const host = '127.0.0.1';

###### const port = 3000;

##### Vamos a correr nuestro servidor en nuestra máquina local, por lo tanto con la variable host nos vamos a referir a localhost usando su dirección IP que es 127.0.0.1, y por convención, vamos a correr nuestro servidor en el puerto 3000.

##### Lo próximo que vamos a hacer es llamar al método createServer que devuelve una nueva instancia de http.Server.

###### const server = http.createServer((req, res) => {

###### res.statusCode = 200;

###### res.setHeader('Content-Type', 'text/plain');

###### res.end('Primer servidor con Node.Js');

###### });

##### Al método createServer le pasamos un callback con dos parámetros req y res .

##### req es la petición que le hacemos al servidor, en sí, es un objeto que tiene toda la información de la petición, y res de la respuesta. Por lo tanto establecemos el código 200 (HTTP 200 = exitoso), los headers (cabeceras de contenido) y para finalizar cerramos la conexión con un mensaje.

##### Luego vamos a llamar al método listen qué hará la magia de concretar la conexión.

###### server.listen(port, host, () => {

###### console.log(`Servidor corriendo en http://${host}:${port}`);

###### });

##### El método listen recibe el puerto, el host y un callback en donde le pasamos el mensaje una vez que la conexión se abra.

##### Por último, una vez dentro de nuestro proyecto/carpeta, mediante la consola, llamamos al archivo.

###### node app.js

###### Servidor corriendo en <http://127.0.0.1:3000>

## Work

### Sesión 02: Módulos esenciales

#### Objetivos

##### Conocer los módulos nativos de NodeJS, para que sirven los principales módulos y sus posibilidades.

#### Ejempolo 1

##### Desarrollo

###### A pesar de que node.js y el navegador ejecutan un mismo estándar de Javascript, Node.js tiene ciertas diferencias marcadas en los objetos globales.

##### Objetos globales

###### Estos objetos o palabras clave, están disponibles en cualquier módulo de node.js

Buffer

\_\_dirname

\_\_filename

clearImmediate(immediateObject)

clearInterval(intervalObject)

clearTimeout(timeoutObject)

console

exports

global

module

process

queueMicrotask(callback)

require()

setImmediate(callback[, ...args])

setInterval(callback, delay[, ...args])

setTimeout(callback, delay[, ...args])

TextDecoder

TextEncoder

URL

URLSearchParams

WebAssembly

###### Los siguientes solo existen dentro del sistema de módulos:

require()

module

exports

\_\_dirname

\_\_filename

###### Hasta ahora has hecho uso de los primeros dos. En el siguiente ejemplo trabajaremos con archivos y utilizaremos las palabras clave \_\_dirname y \_\_filename

##### Core Modules

###### Los Core Modules (módulos principales) vienen con Node.js y no necesitan ser instalados. Estos proporcionan funcionalidad de bajo nivel y métodos auxiliares. Permiten a Node.js trabajar con el sistema de archivos, redes, datos binarios, flujos de datos (streams), generar procesos externos, rutas de archivos y URLs, y realizar otras tareas útiles, como crear clientes y servidores HTTP(S).

###### Aquí está el listado de los principales:

fs - módulo para trabajar con el sistema de archivos.

path - módulo para parsear rutas de archivos a través de distintas plataformas.

net - redes y otros protocolos.

stream - para trabajar con flujos de datos(streams).

events - modulo para implementar event emitters.

child\_process - módulo para generar procesos externos

os - sirve para acceder a la información del sistema operatico, incluyendo la plataforma, número de procesadores, memoria, etc.

url - módulo para procesar URLs.

http - módulo para realizar (cliente) y recibir peticiones (servidor) web.

https - sirve para hacer lo mismo que el anterior, pero para HTTPS

util - utilidades varias

assert - módulo para realizar pruebas basadas en aserciones.

crypto - módulos para cifrar información y crear hashes.

###### Estos módulos no necesitan descargarse o ser instalados.

###### Ejemplo

const http = require('http') // reemplaza `http` con el módulo principal que quieras utilizar

##### path.join()

##### . Como Node.js está diseñado para ser multiplataforma y existen diferencias de sintaxis para leer archivos en cada plataforma, path nos permite crear rutas de archivos de manera independiente.

##### Ejemplo.

###### Vamos a componer una ruta para el archivo app/server.js

const path = require('path')

const ruta

const server = require(path.join('app', 'server.js'))

###### Ejecuta esto y mira lo que imprime

Este código nos servirá para obtener la misma ruta en windows y en sistemas operativos basados en UNIX.

###### Utilizando \_\_dirname obtendremos la ruta absoluta, en lugar de una relativa:

const path = require('path')

const server = require(path.join(\_\_dirname, 'app', 'server.js'))

###### Ejecuta esto y mira lo que imprime

##### .

## Postwork

# Modelo Asíncrono y No Bloqueante

## Prework

### ¿Qué es lo que Node.js tiene de diferente a otros lenguajes de programación?

#### Cuando resolvemos un problema de software, una de las razones por las que nos inclinamos hacia la elección de un lenguaje determinado es por su modelo de concurrencia. El modelo de concurrencia de un lenguaje de programación define el comportamiento y las capacidades que tiene cuando se trata de manejar a múltiples usuarios al mismo tiempo.

#### Existen diferentes modelos de concurrencia, cada uno con sus ventajas y desventajas, teniendo como el más común el modelo de multihilos (multi threads).

#### Un hilo de ejecución (thread) es la capacidad que tiene el procesador para ejecutar un bloque de instrucciones de un programa. Nuestras computadoras ejecutan múltiples programas al mismo tiempo, es decir, tiene múltiples hilos de ejecución disponibles para ejecutar bloques de código de diferentes programas. Traducido a lo que conocemos como Back-end, esto brinda la posibilidad de atender peticiones de múltiples usuarios al mismo tiempo.

#### Lenguajes de programación como Java o Python tienen este modelo de concurrencia, pero JavaScript no. A diferencia de otros lenguajes, JavaScript trabaja con un solo hijo de ejecución (single thread), es decir, solo puede realizar un proceso a la vez.

#### Imagina que tienes una tienda en línea, si tu modelo de concurrencia fuera single threaded entonces una persona podría comprar al mismo tiempo, no habría posibilidad de que dos usuarios estuvieran interactuando con la tienda, agregando productos al carrito o incluso pagando al mismo tiempo. Con un modelo multi threaded, cada usuario sería “atendido” por un hilo de ejecución, permitiendo que más de un usuario pueda interactuar con la tienda.

#### Entonces… ¿cómo es posible que JavaScript sea considerado como uno de los lenguajes más rápidos y que soporta una gran cantidad de usuarios si solamente cuenta con un solo hilo de ejecución? La respuesta es que su modelo de concurrencia se basa en el loop de eventos (Event Loop).

#### A continuación se explicarán los conceptos que hay detrás de Node.js.

### Asincronía

#### La asincronía es uno de los pilares fundamentales de Javascript, ya que es un lenguaje de programación de un solo subproceso o hilo (single thread), lo que significa que sólo puede ejecutar una cosa a la vez.

#### JavaScript fue diseñado para ser ejecutado en navegadores, trabajar con peticiones sobre la red y procesar las interacciones de usuario, al tiempo que mantiene una interfaz fluida.

#### Javascript usa un modelo asíncrono y no bloqueante, con un loop de eventos implementado en un solo hilo de ejecución, para operaciones de entrada y salida (input/output).

##### prueba

#### Antes de explicar cómo funciona el modelo de JavaScript es importante entender algunos conceptos:

##### Operaciones de CPU y Operaciones de I/O (Entrada y Salida).

##### Operaciones Concurrentes y Paralelas.

##### Operaciones Bloqueantes y No Bloqueantes.

##### Operaciones Síncronas y Asíncronas.

#### .

### Operaciones de CPU y de Entrada y Salida

#### Operaciones CPU: Aquellas que pasan el mayor tiempo consumiendo Procesos del CPU, por ejemplo, la escritura de ficheros.

#### Operaciones de Entrada y Salida: Aquellas que pasan la mayor parte del tiempo esperando la respuesta de una petición o recurso, como la solicitud a una API o BD.

#### Concurrencia y Paralelismo

##### Concurrencia: La concurrencia es la capacidad del procesador para simular la ejecución de múltiples programas al mismo tiempo. El CPU ejecuta bloques pequeños de instrucciones de un programa de manera tan rápida que ante nuestros ojos pareciera que todos los programas de nuestra computadora se ejecutan al mismo tiempo.

##### Paralelismo: El paralelismo es la capacidad de ejecutar programas de manera simultánea. A diferencia de la concurrencia, el paralelismo es realmente ejecutar múltiples cosas al mismo tiempo. Hoy en día esto es posible gracias a que nuestras computadoras y dispositivos tienen cores.

##### .

### Bloqueante y No Bloqueante

#### Se refiere a como la fase de espera de las operaciones afectan a nuestra aplicación:

##### Bloqueante: Son operaciones que no devuelven el control a nuestra aplicación hasta que se ha completado. Por tanto el thread queda bloqueado en estado de espera.

##### No Bloqueante: Son operaciones que devuelven inmediatamente el control a nuestra aplicación, independientemente del resultado de esta. En caso de que se haya completado, devolverá los datos solicitados. En caso contrario (si la operación no ha podido ser satisfecha) podría devolver un código de error.

#### Síncrono y Asíncrono

##### Se refiere a ¿cuándo tendrá lugar la respuesta?:

###### Síncrono: La respuesta sucede en el presente, una operación síncrona esperará el resultado.

###### Asíncrono: La respuesta sucede a futuro, una operación asíncrona no esperará el resultado.

##### Con lo anterior en JavaScript podemos tener:

###### Código síncrono y bloqueante o

###### Código asíncrono y no bloqueante

##### JavaScript Síncrono

###### Cada operación se hace de una vez, bloqueando el flujo de ejecución, el hilo es bloqueado mientras espera la respuesta, cuando esta se procesa pasa a la siguiente operación y así sucesivamente al terminar todas las operaciones.

##### JavaScript Asíncrono

###### Cada operación se ejecuta y devuelve inmediatamente el control al hilo, evitando el bloqueo, cuando cada operación termine se enviará una notificación de que ha terminado, es entonces cuando la respuesta se encolará para ser procesada.

##### Mecanismos asíncronos en JavaScript

###### Para controlar la asincronía, JavaScript cuenta con algunos mecanismos:

Callbacks.

Promises.

Async / Await.

###### Callbacks

Un Callback (llamada de vuelta) es una función que se ejecutará después de que otra lo haga.

Es un mecanismo para asegurar que cierto código no se ejecute hasta que otro haya terminado.

Al ser JavaScript un lenguaje orientado a eventos, las callbacks son una buena técnica para controlar la asincronía, sin embargo… ( ver el enlace)

En este enlace encontrarás uno o varios documentos de consulta 📖 para saber más sobre el tema. Estos no son obligatorios, ¡pero te recomendamos revisarlos y así también fomentamos la curiosidad! 😉

##### El temido Callback Hell o Pyramid of Doom

###### Por suerte esto se ha podido resolver utilizando librerías como async, o empleando promesas con librerías como Q.

###### En estos enlaces encontrarás uno o varios documentos de consulta 📖 para saber más sobre el tema. Estos no son obligatorios, ¡pero te recomendamos revisarlos y así también fomentamos la curiosidad! 😉

##### Promises

###### Una promesa es un objeto que representa el resultado de una operación asíncrona y tiene 3 estados posibles:

Pendiente.

Resuelta.

Rechazada.

###### Tienen la particularidad de que se pueden encadenar (then), siendo el resultado de una promesa, los datos de entrada de otra posible función.

###### Las promesas mantienen un código más legible y mantenible que las callbacks, además tienen un mecanismo para la detección de errores (catch) que es posible usar en cualquier parte del flujo de datos.

##### Async / Await

###### Las promesas fueron una gran mejora respecto a las callbacks para controlar la asincronía en JavaScript, sin embargo pueden llegar a ser muy verbosas a medida que se requieran más y más .then().

###### Las funciones asíncronas (async / await) surgen para simplificar el manejo de las promesas.

###### La palabra async declara una función como asíncrona e indica que una promesa será automáticamente devuelta.

###### Podemos declarar como async funciones con nombre, anónimas o funciones flecha.

###### La palabra await debe ser usada siempre dentro de una función declarada como async y esperará de forma asíncrona y no bloqueante a que una promesa se resuelva o rechace.

### Event Loop

#### Event loop

##### El event loop es el que se encarga de implementar las operaciones asíncronas o el non-blocking. El event loop corre en el único hilo que existe en Node y como mencionamos anteriormente, al bloquear el único hilo de node, estamos bloqueando el event loop.

##### Recuerda

###### Libuv (una librería escrita en C), es el que permite que el event loop funcione y todo el comportamiento asíncrono en Node. Puedes conocer más sobre esta librería ingresando a su sitio oficial.

#### CALL STACK

##### Cada vez que una función va a ser ejecutada pasa por el call stack. Como ya sabemos, al trabajar con operaciones asíncronas, estas poseen callbacks, que se ejecutarán una vez el proceso de la operación haya terminado y que se irán añadiendo al callback queue.

#### CALLBACK QUEUE

##### Aquí se agregan los callback o funciones que se ejecutan una vez las operaciones asíncronas hayan terminado. Se utiliza el método FIFO (first input, first output), traducido, primero en entrar, primero en salir.

##### El event loop es el que se encarga de revisar que el call stack este vacío para añadir lo que está dentro del callback queue y ejecutarlo. Mostraré lo anteriormente mencionado con un ejemplo de código para lograr entender de una mejor manera el trabajo del event loop. 😉

##### Ejemplo

###### 1 const fs = requiere ('fs')

###### 2

###### 3 fs.readFile(`${\_\_dirname}/user.json`, function callback(err, file){

###### 4 setTimeout(function timeout2(){

###### 5 console.log(JSON.parse(file));

###### 6 }, 4000);

###### 7 })

###### 8

###### 9 console.log('Hello Event Loop');

###### 10

###### 11 setTimeout(function timeout(){

###### 12 console.log('Hi, I'm executed')

###### 14 }, 5000)

##### Brevemente te explicaremos qué hace el snippet de código mostrado.

###### Línea 1: Obtenemos la librería File System de Nodejs para poder leer un archivo del disco duro.

###### Línea 3: Implementamos la operación asíncrona readFile, el cual recibirá como parámetros la ruta del archivo y el callback.

###### Línea 4: Se encuentra un timer, en este caso, setTimeout la cual es una operación asíncrona y posee una función llamada timeout2 que se ejecutará después de 4 segundos.

###### Línea 9: Operación síncrona que imprimirá en consola el mensaje correspondiente.

###### Línea 11: Un timer que posee una función llamada timeout que se ejecutará después de 5 segundos. Ahora si puedes, piensa por un momento, cómo crees que será el orden en que se va a ejecutar este código y cuál será el orden en el que se imprimirá en consola.

##### Bueno, pasemos a explicar todo el trabajo que Nodejs hace para correr el programa que escribimos.

##### Primeramente, se agrega la operación readFile al call stack, debido a que es la primera del programa. Al ser una operación asíncrona, esta se mueve a la sección de APIs donde ahí esperaremos a que las herramientas encargadas hagan lo necesario para completar la tarea.

##### El programa estará esperando a que el archivo termine de leerse, pero él continuará con la ejecución del programa. En este caso, se encuentra una operación síncrona (se ejecuta de manera muy rápida) que imprime en consola el mensaje.

##### Ahora, vemos que la operación de leer el archivo terminó. Por lo tanto, pasa al callback queue. También, vemos que en el call stack se encuentra un timer. Esto porque es lo que esta después del console.log anterior.

##### El timer posee una función llamada timeout que se va a ejecutar después de 5 segundos e imprimirá en consola un mensaje. Por ello, pasa a la sección de APIs y espera los 5 segundos.

##### Ahora, el event loop revisa si esta vacío el call stack para ejecutar el callback de la operación readFile.

##### Al ejecutar el callback de la operación readFile nos encontramos con otra operación asíncrona. Un timer, con una función llamada timeout2, la cual se ejecutará dentro de 4 segundos. Seguimos esperando a que pasen los 5 segundos del primer timer.

##### Se agrega la función timeout2 a la sección de APIs a esperar que los segundos pasen.

##### Podemos observar que la función timeout2 ya terminó su tiempo. Por lo tanto, pasa al callback queue y como el call stack no contiene ninguna operación en ejecución, lo ejecuta.

##### Se ejecuta el callback de la funcion timeout2, el cual imprime en consola el archivo en formato JSON. También, vemos que la operación timeout culminó y se agrega al callback queue.

##### Finalmente, el event loop vuelve a revisar el call stack y ejecuta el callback de la función que estaba en el callback queue. Al no encontrar nada más el event loop dentro del call stack, el programa se acaba. El resultado final del programa sería el siguiente:

## Work

### .