



Introducción a Node Js



**Activen las cámaras los que puedan y
pasemos asistencia**





Inicio

{desafío}
latam_



Objetivo general:

Crear una aplicación en Node que registre y lea información en formato JSON

Objetivos

/ Ejecutar scripts con Node Js desde la terminal */*

/ Crear archivos con el módulo File System */*

/ Leer archivos con el módulo File System */*

/ Importar y exportar módulos en Node Js */*

Objetivos



Desarrollo

{desafío}
latam_

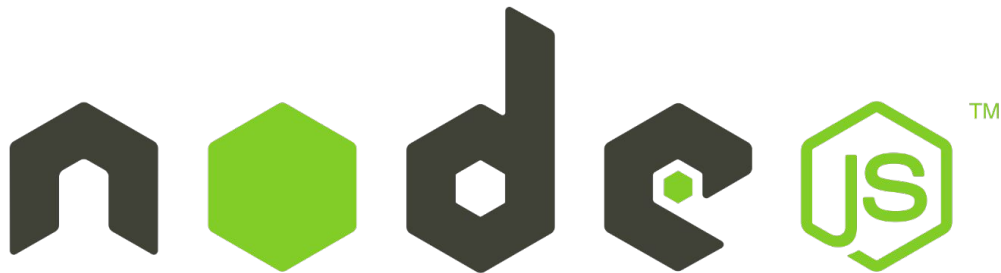


`/* Hola mundo con Node Js */`

Introducción a Node

¿Por qué Node Js?

Con Node podemos ejecutar JavaScript en el lado del servidor, esto quiere decir que nuestros scripts podrán interactuar con el sistema operativo en el que se ejecuten y realizar modificaciones a archivos, acceder a las variables de entorno e incluso ejecutar líneas de comando.



Introducción a Node

Mi primer script

Para ejecutar nuestro primer script con Node Js, solo necesitamos asegurarnos de tener Node Js instalado, para esto abre la terminal que prefieras y ejecuta la siguiente línea de comando:

```
node --version
```

Si al ejecutar esta línea se muestra alguna versión de Node, quiere decir que estás listo para ejecutar tu primer script.

```
$ node --version  
v16.14.2
```

Introducción a Node

Mi primer script

Para ejecutar JavaScript desde la terminal podemos simplemente ejecutar la palabra **node**.

```
C:\Users\Brian>node  
Welcome to Node.js v16.14.2.  
Type ".help" for more information.  
> new Date()  
2022-04-14T13:47:13.057Z
```

Inicia el entorno de Node js

Aquí ya podemos escribir JavaScript

Respuesta

Si quieres ejecutar un archivo JavaScript desde la terminal, deberás ubicarte en la carpeta donde esté este archivo y escribir:

```
node <nombre del archivo.js>
```

Introducción a Node

Hola mundo con Node Js

Hagamos nuestro *¡Hola Mundo!*, para esto:

- Crea una nueva carpeta en el escritorio
- Abre la carpeta con VSC
- Abre la terminal de VSC y agrega un archivo llamado *index.js* y agrega lo siguiente:

```
console.log('Hola Node Js 🙌')
```

- Posteriormente, ejecuta este archivo por la terminal y verás lo siguiente:

```
Brian@LAPTOP-7I8PV10A MINGW64 ~/Desktop/Node  
$ node index.js  
Hola Node Js 🙌
```

/ Ejecutar scripts con Node Js desde la terminal */* ✓

/ Crear archivos con el módulo File System */*

/ Leer archivos con el módulo File System */*

/ Importar y exportar módulos en Node Js */*

Objetivos

Introducción a Node

El módulo File System

Node js en su instalación viene acompañado de una importante y variada galería de módulos, entre ellos uno de sus más populares es el módulo ***File System***.

Conocido también como solo *fs*, este módulo nos permite **leer, crear, eliminar** y varias otras operaciones relacionadas con el sistema de archivos de nuestro computador.

Veamos un ejemplo, creando un archivo de texto que almacene tareas usando el método ***writeFileSync***:

```
const fs = require('fs')
const tareas = `
  1. Ir al gimnasio
  2. Buscar al niño al colegio
  3. Pagar los gastos comunes
`
fs.writeFileSync('tareas.txt', tareas)
```

Introducción a Node

El módulo File System

Si ejecutas el código anterior, verás que ha aparecido un nuevo archivo con nombre *tareas.txt* y al abrirlo encontrarás las tareas declaradas en el script ejecutado.



Introducción a Node

El método *writeFileSync*

Para entender mejor cómo sucedió lo anterior, veamos lo siguiente:

```
const fs = require('fs')
```

Se importa el módulo fs(File System)

```
const tareas = `
```

1. Ir al gimnasio
2. Buscar al niño al colegio
3. Pagar los gastos comunes

Se prepara el contenido del archivo a crear

```
fs.writeFileSync('tareas.txt', tareas)
```

Se ejecuta el método *writeFileSync* de File System

Introducción a Node

El método *writeFileSync*

El método *writeFileSync* tiene la siguiente anatomía:

```
fs.writeFileSync( <nombre del archivo>, <contenido del archivo> )
```

Ambos argumentos deben ser estrictamente en formato **String**.

Si quisiéramos almacenar un objeto o un arreglo, debemos convertirlo primero en String para poder usarlos como contenido. Para esto podemos ocupar el método **JSON.stringify()**

Introducción a Node

El método writeFileSync


Ahora procedamos a crear un nuevo archivo de tareas pero ahora en formato JSON.

Para esto utiliza este nuevo script

```
const fs = require('fs')

const tareas = [
  { text: 'Ir al gimnasio' },
  { text: 'Buscar al niño al colegio' },
  { text: 'Pagar los gastos comunes' },
]

fs.writeFileSync('tareas.json', JSON.stringify(tareas) )
```



El método **JSON.stringify()** convertirá a formato JSON cualquier arreglo u objeto que se le asigne como argumento

Introducción a Node

El método writeFileSync

Al ejecutar el script anterior, verás cómo se crea un archivo *tareas.json* que contiene el arreglo de objetos de las tareas

```
{ } tareas.json × JS index.js
{ } tareas.json > ...
1  [
2    { "text": "Ir al gimnasio" },
3    { "text": "Buscar al niño al colegio" },
4    { "text": "Pagar los gastos comunes" }
5  ]
6
```

Introducción a Node

Sobrescribir un archivo

Cuando queramos actualizar el contenido de un archivo, utilizaremos nuevamente el `writeFileSync` para sobrescribirlo con los nuevos datos.

Probemos sobrescribiendo el archivo `tareas.json` agregando una nueva tarea.

```
const fs = require('fs')

const tareas = [
  { text: 'Ir al gimnasio' },
  { text: 'Buscar al niño al colegio' },
  { text: 'Pagar los gastos comunes' },
  { text: 'Cargar bencina' }
]

fs.writeFileSync('tareas.json', JSON.stringify(tareas) )
```

Nueva tarea

```
{ } tareas.json X
Unidad 1 > clase > { } tareas.json > ...
1  [
2    { "text": "Ir al gimnasio" },
3    { "text": "Buscar al niño al colegio" },
4    { "text": "Pagar los gastos comunes" },
5    { "text": "Cargar bencina" }
6  ]
```

Ejercicio

Ejecuta un script que cree un archivo JSON con el método `writeFileSync` y el siguiente arreglo de objetos:

```
const autos = [  
  {  
    marca: 'Susuki',  
    modelo: 'Kicks',  
  },  
  {  
    marca: 'Toyota',  
    modelo: 'Corola',  
  },  
  {  
    marca: 'GAC',  
    modelo: 'GS4',  
  }  
]
```

Ejercicio

¡Manos al teclado!



/ Ejecutar scripts con Node Js desde la terminal */* ✓

/ Crear archivos con el módulo File System */* ✓

/ Leer archivos con el módulo File System */*

/ Importar y exportar módulos en Node Js */*

Objetivos

Introducción a Node

El método `readFileSync`

Así como podemos crear archivos con el método `writeFileSync`, también podemos leerlos con el método **`readFileSync`**.

Este método tiene la siguiente anatomía:

```
fs.readFileSync( <nombre del archivo>, <codificación del contenido> )
```

Por defecto, siempre especificaremos como codificación 'UTF8', puesto que esta es la codificación de caracteres para nuestro idioma.

Al ejecutar este método, recibiremos como retorno el contenido del archivo.

Introducción a Node

El método readFileSync

Veamos un ejemplo leyendo el mismo archivo que creamos previamente.

Crea un nuevo script que contenga el siguiente código:

```
const fs = require('fs')

const tareas = fs.readFileSync('tareas.json', 'utf8')

console.log(tareas)
```

El objetivo de este código será mostrar por consola el contenido del archivo *tareas.json*

Introducción a Node

El método readFileSync

Ejecutando el script anterior podremos ver por consola lo siguiente:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Brian@LAPTOP-7I8PV10A MINGW64 ~/Desktop/Node
$ node index.js
[
  { "text": "Ir al gimnasio" },
  { "text": "Buscar al niño al colegio" },
  { "text": "Pagar los gastos comunes" },
  { "text": "Cargar bencina" }
]
```

Introducción a Node

El método readFileSync

Al leer un archivo JSON obtendremos un string cuyo contenido podemos transformar o (**Parsear**) fácilmente a un objeto javascript, pero tenemos que recordar transformarlo primero de lo contrario al intentar hacer algo como lo siguiente obtendremos un error.

```
tareas.forEach( (tarea) => {  
  console.log(tarea)  
})
```

```
tareas.forEach((tarea) => {  
  ^
```

```
TypeError: tareas.forEach is not a function
```

Lo cual sucede porque tareas es un **String** y no cuenta con el método *forEach* como los arreglos

Introducción a Node

El método `readFileSync`

Para poder manipularlo como un arreglo (o un objeto si fuera el caso), es necesario parsear el JSON, y esto se hace a través del método `JSON.parse()`, el cual sirve como la inversa del método `stringify`.

```
JSON.parse(tareas).forEach((tarea) => {  
  console.log(tarea)  
})
```

Ejecutando lo anterior podremos ver que la consola nos mostrará correctamente el `console.log()` de cada tarea:

```
$ node index.js  
{ text: 'Ir al gimnasio' }  
{ text: 'Buscar al niño al colegio' }  
{ text: 'Pagar los gastos comunes' }  
{ text: 'Cargar bencina' }
```

Ejercicio

Ejecuta un script que lea el archivo JSON creado en el ejercicio anterior con el método *readFileSync*.

Además de parsear su contenido e imprimir cada uno de los autos por consola:

```
$ node autos.js
[
  { marca: 'Susuki', modelo: 'Kicks' },
  { marca: 'Toyota', modelo: 'Corola' },
  { marca: 'GAC', modelo: 'GS4' }
]
```

Ejercicio

¡Manos al teclado!



/ Ejecutar scripts con Node Js desde la terminal */* ✓

/ Crear archivos con el módulo File System */* ✓

/ Leer archivos con el módulo File System */* ✓

/ Importar y exportar módulos en Node Js */*

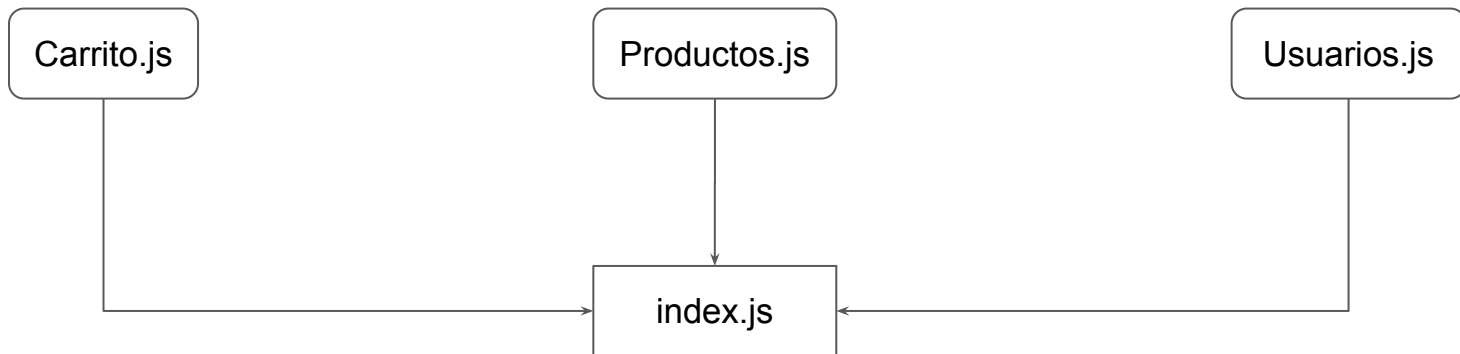
Objetivos

Introducción a Node

Importación y exportación de módulos

A partir de ahora nuestro ***index.js*** va a crecer en líneas de código y empezará a ser cada vez más incómodo leerlo y ubicar la lógica que vamos agregando.

Para que esto no sea un problema existe una buena e importante práctica que consiste en dividir la lógica de la aplicación en diferentes archivos, en donde cada uno podrá exportar e importar la lógica del otro.



Introducción a Node

Exportación de módulos

Nos referiremos como **módulos** a los archivos que estén exportando variables o funciones. Ocupando esta práctica gozaremos de un mayor orden y manejo de todo el código que escribimos.

Exportar e **importar** módulos es una manera de compartir y distribuir la lógica de nuestra aplicación en diferentes archivos interconectados.

Las instrucciones o herramientas que nos ayudarán a aplicar la programación modular serán:

- **require()**
- **module.exports**

Introducción a Node

Exportación de módulos

Veamos un ejemplo creando un archivo llamado *modales.js* que contenga 2 funciones que muestren por consola un saludo cordial y un gracias respectivamente.

```
const saludar = (nombre) => {  
  console.log(`Hola ${nombre}, ¿cómo estás? 😊`)  
}  
  
const darLasGracias = (nombre) => {  
  console.log(`Muchas gracias, ${nombre}`)  
}  
  
module.exports = { saludar, darLasGracias }
```

Con ***module.exports*** podemos declarar qué variables o funciones queremos exportar de este archivo.

Es recomendable contener todas las exportaciones en un objeto para facilitar la importación de los mismos.

Introducción a Node

Importación de módulos

Ahora en el *index.js*, así como importamos el módulo *fs*, importemos las funciones del archivo *modales.js*. Para esto solo necesitamos requerir el archivo y opcionalmente desestructurar su contenido.

El código quedaría de la siguiente manera:

```
const { saludar, darLasGracias } = require('./modales.js')

saludar('Gonzalo')

darLasGracias('Javiera')
```

Introducción a Node

Importación de módulos

Ahora ejecutemos el *index.js* y deberíamos ver lo siguiente por la terminal:

```
Unidad 1 > clase > JS index.js > ...
1  const { saludar, darLasGracias } = require("./modales.js")
2
3  saludar("Gonzalo")
4
5  darLasGracias("Javiera")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Brian@LAPTOP-7I8PV10A MINGW64 ~/Desktop/Node y Express/Unidad 1/clase
$ node index.js
Hola Gonzalo, ¿cómo estás? 😊
Muchas gracias, Javiera
```

Ejercicio

Crea un archivo **Operaciones.js** que cree las siguientes funciones:

- **createFile**: Recibe 2 argumentos (nombre del archivo, contenido del archivo), crea el archivo y muestra por consola un mensaje indicando que el archivo fue creado
- **readFile**: Recibe 1 argumento (nombre del archivo), lee el archivo con *readFileSync* y muestra su contenido por consola.

Exporta ambas funciones

Crea un archivo **index.js** que:

- Importe las funciones de *operaciones.js*
- Pruebe ambas funciones importadas

Ejercicio ¡Manos al teclado!



/ Ejecutar scripts con Node Js desde la terminal */* ✓

/ Crear archivos con el módulo File System */* ✓

/ Leer archivos con el módulo File System */* ✓

/ Importar y exportar módulos en Node Js */* ✓

Objetivos

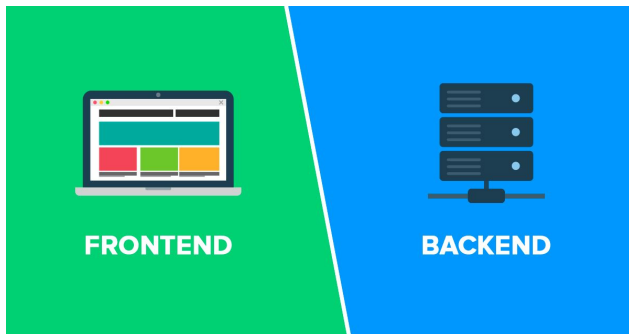
Introducción a Node

JavaScript en el navegador VS JavaScript en el servidor

Existen diferencias importantes entre lo que podemos hacer al escribir código JavaScript en el navegador versus código escrito en el servidor con Node Js.

De las principales diferencias está la inexistencia del objeto global ***window*** en el lado del servidor, debido a que no existe un árbol de nodos ni elementos HTML que manipular, así como tampoco el uso de CSS.

Los métodos para interactuar con los usuarios en los navegadores como *alert()* o *prompt()* son exclusivos del desarrollo frontend. Mientras que en Node js contamos con módulos como **fs** (File System) o la programación modular por defecto.



Introducción a Node

¿Qué podemos hacer con Node Js?

Entre los diferentes usos de Node Js, tenemos los siguientes:

Creación de servidores

Crear API REST

Conexiones con Bases de
Datos

Envío de correos
electrónicos

Gestión de archivos

Conexión con
componentes electrónicos



Todas estas funcionalidades son exclusivas para el desarrollo backend de aplicaciones.



Cierre

{desafío}
latam_



¿Existe algún concepto que no
hayas comprendido?

Reflexionemos

- Revisar la guía que trabajarán de forma autónoma.
- Revisar en conjunto el desafío.

¿Qué sigue?



*Academia de
talentos digitales*

www.desafiolatam.com



/DesafioLatam



/DesafioLatam



/DesafioLatam



/DesafioLatam