¡Les damos la bienvenida!

¿Comenzamos?

Esta clase va a ser

- grabada

Clase 05. DESARROLLO AVANZADO DE BACKEND

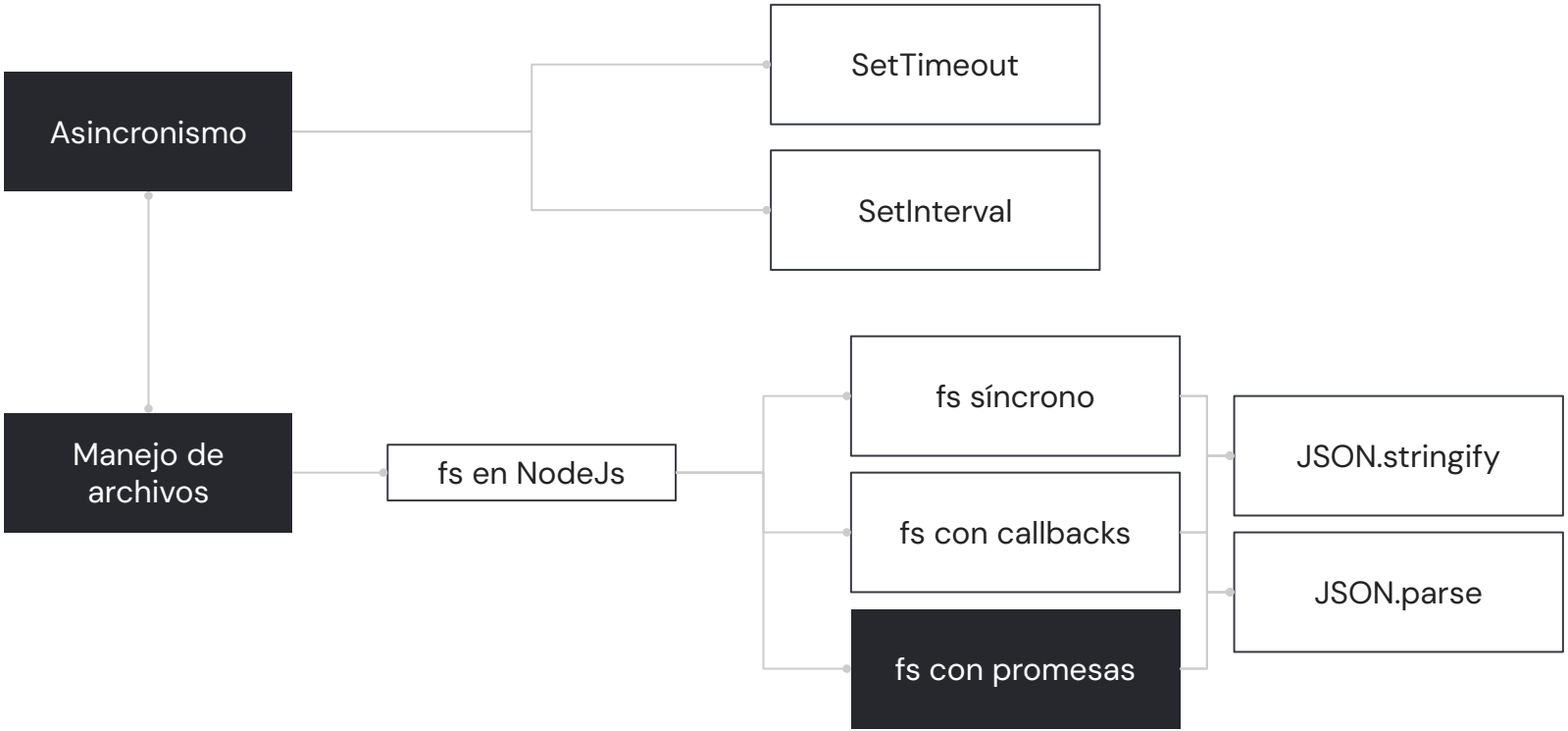
Manejo de archivos en Javascript

CODERHOUSE

Objetivos de la clase

- Utilizar la programación sincrónica y asincrónica y aplicarla en el uso de archivos
- Conocer el módulo nativo de Nodejs para interactuar con los archivos.
- Conocer la utilización de archivos con callbacks y promesas
- Conocer las ventajas y desventajas del FileSystem, así también como ejemplos prácticos.

MAPA DE CONCEPTOS



PARA RECORDAR

Sincronismo

Las operaciones síncronas o bloqueantes, nos sirven cuando necesitamos que las operaciones se ejecuten **una detrás de otra**, es decir, se utiliza cuando **deseamos que las tareas sean secuenciales, independientemente del tiempo que demore cada operación.**

PARA RECORDAR

Asincronismo

Las operaciones asíncronas o **no bloqueantes**, nos sirven cuando necesitamos que haya múltiples tareas ejecutándose, sin tener que esperar a las tareas que ya se están ejecutando. **Úsalas cuando necesites hacer alguna operación, sin afectar al flujo principal.**

Ejemplos de asincronismo: SetTimeout y SetInterval

Más allá de la memoria...
manejo de archivos

El problema: persistencia en memoria

Cuando comenzamos a manejar más información, nos encontraremos con una de las grandes molestias del programador: tener que comenzar desde 0 cada vez que el programa termina su ejecución.

Todas las cosas que creamos, movimos o trabajamos desaparecen, ya que **sólo persiste en memoria**, y esta se borra automáticamente al finalizar el programa.

fs en Nodejs

fs es la abreviación utilizada para FileSystem, el cual, como indica el nombre, es un sistema de manejador de archivos que nos proporcionará node para poder crear, leer, actualizar o eliminar un archivo, sin tener que hacerlo nosotros desde cero.

Así, crear un archivo con contenido será tan fácil como escribir un par de líneas de código, en lugar de tener que lidiar con los datos binarios y transformaciones complejas y de un nivel más bajo en la computadora.

Usando fs de manera sincrónica

fs síncrono

¡El uso de fs de manera síncrona es bastante sencillo! para ello, sólo utilizaremos la palabra Sync después de cada operación que queramos realizar. Hay muchas operaciones para trabajar con archivos, pero sólo abarcaremos las principales.
Las principales operaciones que podemos hacer con fs síncrono son:

- ✓ **writeFileSync** = Para escribir contenido en un archivo. Si el archivo no existe, lo crea. Si existe, lo sobreescribe.
- ✓ **readFileSync** = Para obtener el contenido de un archivo.
- ✓ **appendFileSync** = Para añadir contenido a un archivo. ¡No se sobreescribe!
- ✓ **unlinkSync** = Es el “delete” de los archivos. eliminará todo el archivo, no sólo el contenido.
- ✓ **existsSync** = Corrobora que un archivo exista!

fs con Callbacks

fs con callbacks

Las principales operaciones que podemos hacer con fs con callbacks son:

- ✓ **writeFile** = Para escribir contenido en un archivo. Si el archivo no existe, lo crea. Si existe, lo sobreescribe. Al sólo escribir, su callback sólo maneja: **(error)=>**
- ✓ **readFile** = Para obtener el contenido de un archivo. Como pide información, su callback es de la forma: **(error, resultado)=>**
- ✓ **appendFile** = Para añadir contenido a un archivo. ¡No se sobreescribe!, al sólo ser escritura, su callback sólo maneja: **(error)=>**
- ✓ **unlink** = Es el “delete” de los archivos. eliminará todo el archivo, no sólo el contenido.
  Al no retornar contenido, su callback sólo es
  (error)=>

¡Importante!

Recuerda que los callbacks son peligrosos si necesitas encadenar múltiples tareas. Si necesitas hacer operaciones muy complejas con archivos, no uses callbacks o terminarás en un ***CALLBACK HELL***

ACTIVIDAD EN CLASE

Almacenar fecha y hora

- ✓ Realizar un programa que cree un archivo en el cual escriba la fecha y la hora actual. Posteriormente leer el archivo y mostrar el contenido por consola.
- ✓ Utilizar el módulo fs y sus operaciones de tipo callback.

Break

¡10 minutos y volvemos!

fs con promesas

Ya sabemos trabajar con archivos, ya vimos cómo trabajarlos de manera asíncrona, ahora viene el punto más valioso: **trabajar con archivos de manera asíncrona, con promesas**. esto lo haremos con su propiedad **fs.promises**

fs.promises

Al colocar a nuestro módulo fs el **.promises** estamos indicando que, la operación que se hará debe ser ejecutada de manera **asíncrona**, pero en lugar de manipularla con un callback, lo podemos hacer con .then +.catch, o bien con async/await.
Los argumentos y estructura es casi idéntico al síncrono, por lo tanto sus operaciones principales serán:

- ✓ **fs.promises.writeFile** = Para escribir contenido en un archivo. Si el archivo no existe, lo crea. Si existe, lo sobreescribe.
- ✓ **fs.promises.readFile** = Para obtener el contenido de un archivo.
- ✓ **fs.promises.appendFile** = Para añadir contenido a un archivo. ¡No se sobreescribe!
- ✓ **fs.promises.unlink**= Es el “delete” de los archivos. eliminará todo el archivo, no sólo el contenido.

Manejo de datos complejos

Como ya podrás imaginar, no todo son archivos .txt, y por supuesto que no todo es una cadena de texto simple. ¿Qué va a pasar cuando queramos guardar el contenido de una variable, aun si esta es un objeto? ¿Y si es un arreglo? Normalmente los archivos que solemos trabajar para almacenamiento, son los archivos de tipo **json**.

Para poder almacenar elementos más complejos, nos apoyaremos del elemento JSON.stringify() y JSON. parse()

ACTIVIDAD EN CLASE

Lectura y escritura de archivos

Escribir un programa ejecutable bajo node.js que realice las siguientes acciones:

- ✓ Abra una terminal en el directorio del archivo y ejecute la instrucción: *npm init -y*.
  - Esto creará un archivo especial (lo veremos más adelante) de nombre *package.json*
- ✓ Lea el archivo *package.json* y declare un objeto con el siguiente formato y datos:

```
const info = {
    contenidoStr: (contenido del archivo leído en formato string),
    contenidoObj: (contenido del archivo leído en formato objeto),
    size: (tamaño en bytes del archivo)
}
```

ACTIVIDAD EN CLASE

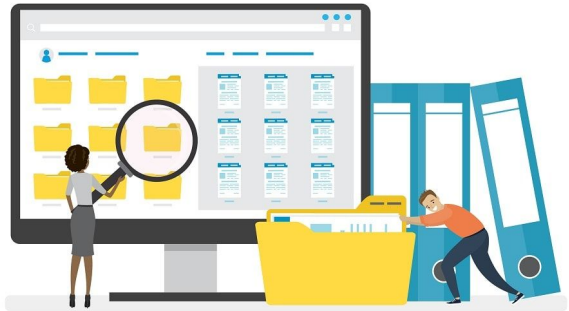
Lectura y escritura de archivos

- ✓ Muestre por consola el objeto info luego de leer el archivo
- ✓ Guardar el objeto info en un archivo llamado info.json dentro de la misma carpeta de package.json
- ✓ Incluir el manejo de errores (con throw new Error)
- ✓ Utilizar el módulo promises de fs dentro de una función async/await y utilizar JSON.stringify + JSON.parse para poder hacer las transformaciones json->objeto y viceversa

Ventajas y desventajas de utilizar archivos

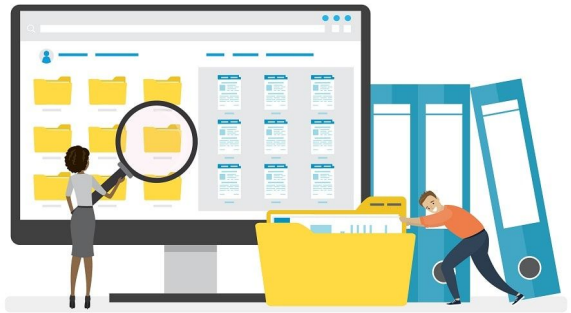
¿Por qué usarlos?

- ✓ Son excelentes para empezar a aprender persistencia, ya que son muy fáciles de usar
- ✓ Al ser nativo de node js, no tenemos que hacer instalaciones externas.
- ✓ Es muy fácil de manipular dentro o fuera de nuestro programa, además de ser transferible.



Desventajas

- ✓ Conforme la información crece, nos daremos cuenta que, para querer modificar una sola cosa, necesitamos leer todo el archivo, lo cual consume recursos importantes.
- ✓ Similar al punto anterior, una vez modificado un dato puntual del archivo, tengo que **reescribir el archivo** completamente, lo cual es un proceso innecesario y pesado cuando la información es grande.
- ✓ Al final, puede ser peligroso tener toda la información en un archivo fácilmente extraíble con un drag&drop a otra carpeta.



Para pensar

Una vez vistas las ventajas y desventajas de su uso, toca preguntarnos **¿en el desarrollo web es utilizado, o es algo más propio sólo de Sistemas Operativos?**

HANDS ON LAB

Manager de usuarios

¿Cómo lo hacemos? **Se creará una clase que permita gestionar usuarios usando fs.promises, éste deberá contar sólo con dos métodos: Crear un usuario y consultar los usuarios guardados.**

- ✓ El Manager debe vivir en una clase en un archivo externo llamado UsersManager.js (Como el realizado en la clase pasada).
- ✓ El método “Crear usuario” debe recibir un objeto con los campos:
  - Nombre
  - Apellido
  - Edad
  - Curso

  El método debe guardar un usuario en un archivo “Usuarios.json”, **deben guardarlos dentro de un arreglo, ya que se trabajarán con múltiples usuarios**

- ✓ El método “ConsultarUsuarios” debe poder leer un archivo Usuarios.json y devolver el arreglo correspondiente a esos usuarios

¿Preguntas?

Muchas gracias.