Práctica 5

Asincrónismo

## Temas:

* Asincronismo: Callback, promesas
* Node: I/O archivo y red.

## Referencias:

* [Modulo fs](https://nodejs.org/dist/latest-v8.x/docs/api/fs.html" \l "fs_file_system)
* [fs.writeFile](https://nodejs.org/dist/latest-v6.x/docs/api/fs.html" \l "fs_fs_writefile_file_data_options_callback)
* [fs.writeFileSync](https://nodejs.org/dist/latest-v6.x/docs/api/fs.html" \l "fs_fs_writefilesync_file_data_options)
* [fs.readFile](https://nodejs.org/dist/latest-v6.x/docs/api/fs.html" \l "fs_fs_readfile_file_options_callback)

## Manejo de Archivos en node

A continuación se muestran dos maneras de leer un archivo, una asincrónica y otra sincrónica. Ejecute ambas en su ambiente Node. Analice la ejecución (puede debuggear y/o agregar más líneas de logging). Pruebe usando un archivo chico y uno grande.

Opción 1

|  |
| --- |
| const fs = require('fs'); const filename = 'unArchivo.txt';  function processFile(readContent) {  console.log('Processing my file');  console.log('file size: ', readContent.length); }  fs.readFile(filename, (err, data) => {  if (err) {  console.log('Ouch! Error!');  throw err;  }  console.log('The file was correctly read.');  processFile(data); });  console.log('Running the last line of main file') |

Opción 2

|  |
| --- |
| const fs = require('fs'); const filename = 'unArchivo.txt';  function processFile(readContent) {  console.log('Processing my file');  console.log('file size: ', readContent.length); } try {  let content = fs.readFileSync(filename);  console.log('The file was correctly read.');  processFile(content); } catch (exception) {  console.log('Ouch! Error!!');  } console.log('Running the last line of main file') |

Compare ambas soluciones y responda brevemente:

1- **Qué ocurre en flujo de control para cada caso? Explique cómo ocurre la ejecución para ambas versiones.**

En el caso de la segunda es un read sincrónico, por lo que hasta no terminar la lectura no avanza en la ejecución. En el caso del primero es asincrónico, por lo que el read queda como una tarea en paralelo que se termina de ejecutar para cuando se vuelva el control a las tareas pendients.

2- **Suponiendo que el archivo a leer es muy grande y va a demorar, cual de las dos opciones prefiere? Cual le permite “seguir” procesando?**En este casp como en la ejecución no se necesita más adelante de la info del archivo a procesar, se podría elegir la opción asincronica

**3- ¿Cuál de las dos variantes sigue la filosofía de Node?**

La asinc

**4- ¿Qué efectos negativos puedo tener en un sistema implementado en node donde se hace de manera sincrónica E/S?**

Que se demore mucho en la ejecución de manera innecesaria

5- Pruebe usando un filename válido y uno inválido. Compare cómo se realiza el manejo de los errores para cada caso.

En el caso de la manera asincronica, el error viene por parámetro en el callback, y hay que manejar ese caso. En el casio de las sincronicas , hay que atraparlo con un try.. catch...

## Callbacks y manejo de archivos

Copie [este](https://drive.google.com/file/d/0B0pTTrBCIS2sVGd5R3NRQWNPcWc/view?usp=sharing) json a un archivo en su máquina y haga una función que:

1. Lea el archivo y convierta de JSON a javascript,
2. Saque el promedio de los alumnos. Generando un nuevo objeto de la forma  
   {  
    <nombreAlumno1>: <promedioAlumno1>,  
    <nombreAlumno2>: <promedioAlumno2>  
   }
3. Escriba los promedios en un nuevo archivo promedios.json
4. Una vez escrito el archivo promedios.json deberá imprimir en consola el mensaje: “Todos los promedios calculados exitosamente”
5. Ante cualquier error deberá imprimir un mensaje de error en la consola.

## Callbacks: Comida rápida (Visado opcional)

Se desea modelar un restaurante de comida rápida.

En él los clientes llegan y son atendidos por un cajero quien les cobra y les da un número.

Los clientes esperan a que sus pedidos estén listos, mientras eso ocurre, como son muy ansiosos imprimen la pantalla un mensaje diciendo cuantos segundos esperaron.

Asuma que son N (con N<9) clientes, los cuales tienen asignados los números del 1 al N.

Para simular “la cocina” usaremos la consola, es decir ud. es el cocinero y va a ingresar el número del pedido que está listo.

Cuando un cliente obtiene su pedido imprime un cartel notificandolo, y deja de mostrar el mensaje que indicaba que estaba en espera.

Cuando se han satisfecho TODOS los pedidos el local debe cerrar, entonces imprimirá en pantalla un cartel que diga: “------------------ CERRADO --------------------”

Para implementar este ejercicio deberá usar readline, settimeout y bind. A continuación se muestran ejemplos de uso. Recomendamos leer la documentación.

Readline

|  |
| --- |
| const readline = require('readline'); let rl = readline.createInterface(process.stdin, process.stdout);  function processInput(input) {  console.log('Input: ', input); }  rl.on('line', processInput); |

Settimeout

|  |
| --- |
| class Person {  constructor(\_name) {  this.name = \_name;  }  endWaiting() {  console.log('Termine de esperar');  } wait(seconds) {  setTimeout(this.endWaiting.bind(this), seconds \* 1000);  } } pepe = new Person('Pepe'); pepe.wait(5); |

Modele el objeto Cliente. El restaurante puede ser un objeto o simplemente le main donde se manejan a los clientes.

**Nota**: una vez que ya no haya más clientes puede utilizar el método close() del objeto readline.

---------------------

## Promesas

### Ejemplo (ver archivo adjunto prepararHambuguesa.js)

Se desea modelar el proceso de preparación de una hamburguesa en una casa de comidas rápidas.

1. Vea el archivo prepararHambuguesa.js el cual ya tiene la solución, ejecutelo, entienda el código y consulte sus dudas. **Nota:** Ejecutarlo varias veces y vea cómo las acciones que ocurren en paralelo no se resuelven siempre en el mismo orden, pero las dependencias entre tareas son respetadas.
2. Modifique el còdigo para imprimir en consola: 'La hamburguesa no pudo ser hecha' en caso de que haya ocurrido un error en cualquier parte de la cadena de producción de la hamburguesa.   
   **Nota:** tendrà que forzar un error en alguna parte de la cadena, por ejemplo agregando la sentencia: throw new Error('Algo malo paso') o, retornando una Promesa rechazada: Promise.reject(new Error('Algo malo paso'))
3. Modifique el código para agregar Mayonesa a la hamburguesa. Esto se debe hacer despuès de haber Obtenido los tres ingredientes, para eso, la “Carne cocida” deberá transformada en “Carne cocida con Mayonesa”

### Preparando un pedido

Continuando con el ejemplo anterior, ahora se desea modelar el armado de un pedido completo, el cual está compuesto por: bebida, papas fritas y la hamburguesa.

1. Vea el archivo prepararPedido.js, complete con las partes de còdigo faltante necesarias para resolver el problema. Deberá chequear que el pedido impreso en consola:
   1. Se imprime una vez finalizados los pasos previos (y no antes)
   2. El objeto pedido que imprime es el esperado según el enunciado del problema.
2. Modifique su código para que soporte preparar papas fritas sin sal. Deberá agregarle un parámetro booleano al método prepararPedido que indique si las papas se quieren con sal o no. Si no se quieren con sal, se deberá evitar el paso de salarPapas()
3. Modifique su código para que retorne una “Botella de agua” si no se pudo preparar la bebida. **Nota:** Para probar su código deberá forzar una falla en su implementación de prepararBebida()

## Bajando contenido de Internet

Analice el ejemplo provisto a continuación, el cual descarga un recurso de internet (puede cambiar la URL si lo desea). Implemente una versión equivalente utilizando promesas.

Lea la documentación de <https://www.npmjs.com/package/request-promise> para resolver este ejercicio

|  |
| --- |
| const request = require('request'); const options = {   url: 'https://www.reddit.com/r/funny.json',  method: 'GET',  headers: {  'Accept': 'application/json',  'Accept-Charset': 'utf-8',  'User-Agent': 'my-reddit-client'  } };  request(options, function(err, res, body) {   let json = JSON.parse(body);  console.log(json); }); |

### Promesas y manejo de archivos

Implemente el ejercicio Callbacks y manejo de archivos utilizando promesas, para eso deberá “promisificar” las funciones de manejo de archivos. Para promisificar fs.readfile:

|  |
| --- |
| const util = require('util');  const fs = require('fs');  const readFile = util.promisify(fs.readFile);  readFile('archivo.txt').then((data) => {  // Hacer algo con data  }).catch((error) => {  // Manejar el error  }); |