Práctica 5

Asincrónismo

Temas:

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

Referencias:

- Modulo fs
- fs.writeFile
- fs.writeFileSync
- fs.readFile

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.

```
const fs = require('fs');
const filename = '/Users/arturo/English/IELTS Advantage - Reading + Writing/IELTS Advantage.
Writing Skills.pdf';

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')
```

```
const fs = require('fs');
const filename = '/Users/arturo/English/IELTS Advantage - Reading + Writing/IELTS Advantage.
Writing Skills.pdf';

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 en cómo ocurre la ejecución para ambas versiones.
2- Suponiendo que el archivo a leer es muy grande y va demorar, cual de las dos opciones prefiere? Cual le permite “seguir” procesando?
3- ¿Cuál de las dos variantes sigue la filosofía de Node?
4- ¿Qué efectos negativos puedo tener en un sistema implementado en node donde se hace de manera sincrónica E/S?
5- Pruebe usando un filename válido y uno inválido. Compare cómo se realiza el manejo de los errores para cada caso.

Callbacks: Comida rápida

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 esten 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 numeros 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.

Callbacks y manejo de archivos

Copie este 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: “To
5. dos los promedios calculados exitosamente”
6. Si no puede escribir el archivo `promedios.json` deberá imprimir un mensaje de error en la consola.

Promesas

Ejemplo (ver archivo adjunto preparandoHambuguesas.js)

Se desea modelar el proceso de preparación de una hamburguesa en una casa de comidas rápidas.
Los pasos para preparar la hamburguesa son:

a. Tostar el pan de arriba.
b. Tostar el pan de Abajo.
c. Cocinar la carne de un lado.
d. Cocinar la carne del otro lado. (Depende de c)
e. Armar la hamburguesa. (Depende de a, b y d)

Depende significa, en este contexto, que una tarea sólo puede ser realizada cuando su predecesor ha finalizado exitosamente.

Además, se disponen de 4 máquinas que trabajan de manera independiente. Cada máquina ejecuta una de las acciones mencionadas anteriormente (a. a la d.)

Estas máquinas se pueden accionar utilizando las funciones:

```
cocinarCarneLadoA,
cocinarCarneLadoB,
tostarPanA,
tostarPanB
```

Cada una de ellas retornará una promesa, que se resolverá con el recurso mandado a hacer.

Cada recurso es un string:

`cocinarCarneLadoA` resolverá al string `"carne semicocida"`
`cocinarCarneLadoB` resolverá al string `"carne cocida"`
`tostarPanA` resolverá al string `"Pan A"`
`tostarPanB` resolverá al string `"Pan B"`

La hamburguesa final armada deberá representarse como un array de la forma:

```
["pan A", "carne cocida", "pan B"]
```

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 y la hamburguesa.

Los pasos para armar el pedido son:

1. Preparar las papas fritas
2. Preparar la bebida.
3. Preparar la hamburguesa (ya lo tiene resuelto por el ejemplo provisto).
4. Poner todo en la bandeja. (1, 2 y 3 deben estar terminados)

Preparar las papas implica

a. Freir las papas
b. Empaquetarlas (depende de a)
c. salarlas (depende de b)

Preparar la bebida implica un solo paso, servir la bebida en el vaso.

Además, se disponen de 3 empleados: uno que prepara las papas, otro que prepara la hamburguesa y otro que prepara la bebida. Cada empleado trabaja de manera independiente.

El pedido deberá ser un objeto javascript de la forma:

```
{
  papas: 'Papas fritas empaquetadas con sal',
  hamburguesa: ["pan A", "Carne cocida", "pan B"],
  bebida: 'Bebida',
}
```

Usted dispone de las siguientes funciones:

`prepararBebida` retorna una promesa que se resolverá con el string `"Bebida"`
`freirPapas`: retorna una promesa que se resolverá con el string `"Papas fritas"`
`empaquetarPapas`: retorna una promesa que se resolverá con el string `"empaquetadas"`
`salarPapas`: retorna una promesa que se resolverá con el string `"con sal”`

Para este ejercicio use como base el archivo preparandoPedido.js

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
});
```