Exportaciones de módulos

A medida que nuestro proyecto crece podemos separar nuestros componentes en múltiples archivos conocidos como módulos. Un módulo puede contener una clase o una biblioteca de funciones para un propósito en específico. Dentro del sistema de módulos de Node se puede distinguir 3 tipos de módulos: Los que generamos en nuestro proyecto, los que descargamos desde otros desarrolladores y los que vienen incluido con Node.

Para generar nuestros propios módulos debemos usar la sentencia `module.exports`:

operaciones.js

```
sumar = function(a,b){
 return a+b;
}
function restar(a,b){
  return a-b;
}
module.exports = { sumar,restar};
```

Para utilizar el módulo que acabamos de exportar, debemos importar el módulo con la sentencia `require` y ejecutarlo de la siguiente forma:

app.js

```
const op= require('./operaciones');
console.log(op.sumar(3,4)); //7
```

El anterior ejemplo lo podemos desestructurar en objetos separados:

```
const {sumar,restar} = require('./modulos/operaciones');
console.log(sumar(3,5)); ///retorna 8
console.log(restar(5,3)); //retorna 2
```

Para importar librerias de Node o módulos de otros desarrolladores usamos la sentencia `require` seguido del nombre del módulo cómo parámetro. Ejemplo para importar la libreria FileSystem `fs`:

```
const fs = require('fs');
fs.writeFile('./archivos/archivo1.txt','Hola',(err)=>{
    if(err) throw err;
    console.log("El archivo a sido creado");
});
```

async y await

las funciones `async` y `await` están disponibles desde ecmaScript 7, permiten escribir funciones asíncronas de una

forma más fácil de leer y escribir. Otra característica especial de las funciones `async` es que siempre retornan promesas. Ejemplo:

```
let getNombre = async ()=>{
  return 'Marco Etcheverry';
}
```

El código anterior es equivalente a:

```
let getNombre = ()=>{
    return new Promise((resolve,reject)=>{
        resolve('Marco Etcheverry');
    });
}
```

Por tanto `getNombre` se lo ejecuta de la siguiente forma:

```
getNombre().then(res=>{
    console.log(res);// retorna Marco Etcheverry
});
```

Para disparar un error usamos la sentencia `throw`. Ejemplo:

```
let getNombre = async ()=>{
    throw new Error("no existe el nombre");
    return 'Marco Etcheverry';
}


getNombre().then(res=>{
    console.log(res);
}).catch((e)=>{
    console.log(e);//despliega el error en consola
})
```

await es otra palabra reservada que está disponible desde ecmaScript 7 que trabaja dentro de las funciones `async`. La sentencia `await` se usa para indicar que se espere la ejecución hasta que la promesa retorne un resultado. Ejemplo:

```
let getNombre = () => {

    return new Promise((resolve, reject)=>{
        setTimeout(()=>resolve('Marco Etcheverry'),3000);
    });
};


let saludar = async ()=>{
    let nombre = await  getNombre();
    return `Hola ${nombre}`;
```

```
}

saludar().then((msj)=>console.log(msj));//retorna 'Hola Marco Etcheverry' despues de 3 seg.
```

```
function resolveAfter2Seconds() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve('resolved');
    }, 2000);
  });
}

async function asyncCall() {
  console.log('calling');
  const result = await resolveAfter2Seconds();
  console.log(result);
  // Expected output: "resolved"
}

asyncCall();
```

El Objeto Process

La variable global `process` es un objeto que nos ofrece diversos métodos, eventos y propiedades acerca del proceso que esta ejecutando un script Node. En la documentación oficial podemos ver toda la información completa respecto a este objeto: https://nodejs.org/docs/latest/api/process.html Entre las informaciones respecto al proceso que se puede obtener, podemos destacar:

```
console.log('id del proceso: ', process.pid);
console.log('título del proceso: ', process.title);
console.log('versión de node: ', process.version);
console.log('sistema operativo: ', process.platform);
```

Salir de la ejecución de un programa Esto lo podemos conseguir invocando al método `exit()`:

```
process.exit();
```

Escuchar evento Podemos agregar eventos de escucha al objeto process usando el método `on()`. Por ejemplo si queremos escuchar el evento exit:

```
 process.on('exit',function(codigo){
   console.log("Saliendo del proces con codigo de salida",codigo);
})
```

Leer argumentos

Los argumentos pueden ser pasados al momento de ejecutar un proceso node. Ejemplo:

```
$ node app.js --arg1='Hola' --arg2=5
```

En NodeJs los argumentos del CLI los podemos obtener desde el objeto `process.argv`. el objeto `argv` es un array que contiene 2 o mas elementos:

- El primer argumento es la ruta completa del comando node.
- El segundo elemento es la ruta completa del archivo que se está ejecutando.
- Todos los argumentos adicionales están presentes desde la tercera posición en adelante.

Esto lo podemos verificar:

```
console.log(process.argv);
```

Para poder trabajar de forma optima con los argumentos podemos usar la libreria `minimist`

```
const argv = require('minimist')(process.argv.slice(2))
console.log(argv.arg1) //Hola
console.log(argv.arg2) //5
```

Trabajar con Archivos

Para poder leer y escribir archivos necesitamos hacer uso del módulo Filesystem , este módulo se encuentra dentro del paquete core de Node y nos permite realizar las tareas básicas con archivos como:

- Leer archivos
- Crear archivos
- Actualizar archivos
- Eliminar archivos
- Cambiar el nombre de los archivos

Este módulo tiene los métodos necesarios para trabajar tanto de forma síncrona y asíncrona. Ejemplo para leer el directorio actual usando `fs`:

```
var fs = require('fs');
var files = fs.readdirSync('./');
console.log(files);// devuelve un array con los archivos[ 'app.js', 'personas.json' ]
```

Para leer el contenido de un archivo se ejecuta el método `readFile`

Leer archivo

```
fs.readFile("./lineas.txt",function(err, data){
    if(err) throw err;
    console.log(data.toString());
});
```

Escribir líneas

Para añadir nuevas líneas se usa el método appendFile

```
fs.appendFile("./lineas.txt","\n Este texto se añadira al final",function(err){
    if(err)throw err;
    console.log("El archivo a sido creado y actualizado");
})
```

Práctica Nro. 20

Desarrollar una aplicación NodeJs para gestionar tareas en un archivo .txt, de tal forma que las nuevas tareas se lo agregan como argumento desde el CLI. Ejemplo

$ node app.js –tarea “Limpiar habitación”

ListaTareas.txt Tarea 1 Tarea 2 Tarea 3 Tarea 4 Limpiar habitacion