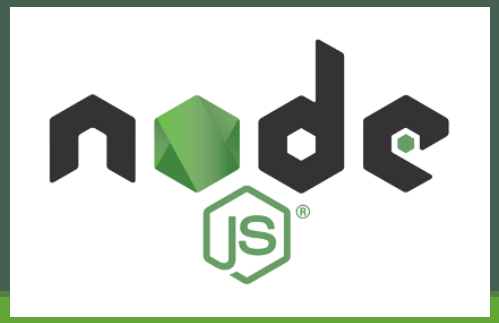

Streams

Contenido

- Buffers
- Streams
- FileSystem

Buffers en NodeJS

- Conjuntos de datos en crudo, datos binarios, que podemos tratar en NodeJS para realizar diversos tipos de acciones
- Son instancias para almacenar datos sin procesar similares a una matriz de números enteros
- Los implementa Node a través de una clase específica llamada **Buffer**
- Se utilizan en la comunicación bidireccional que tenemos cuando manejamos sockets, pero también al manipular imágenes o streams de datos.

Crear un Buffer

- La clase Buffer es global en NodeJS, por lo que no se necesita hacer require de ningún módulo para poder usarla.
- El tamaño de un buffer se establece en el momento de su creación y luego ya no es posible cambiarlo.
- Para crear un buffer:

```
var b1 = Buffer.alloc(20);
```

  - Esto crea un buffer con tamaño de 20 bytes sin contenido. Cada uno de sus bytes estará inicializado a cero.
- Para crear un buffer con contenido:

```
var b2 = Buffer.from('Ejemplo de buffer');
```

- En este caso la codificación por defecto es 'utf8'

Buffer: Escribir en una cadena

```
buffer.write(string[, offset][, length][, encoding])
```

- Descripción de los parametros utilizados:
  - String: Esta es la cadena de datos que se escribirá en el buffer.
  - Offset: Este es el índice del buffer para comenzar a escribir en un punto establecido, el valor predeterminado es 0.
  - Length: Este es el número de bytes a escribir.
  - Encoding: Codificación a utilizar. 'Utf8' es la codificación predeterminada.

- Valor del retorno: Este método devuelve el número de octetos escritos. Si no hay suficiente espacio en el buffer para ajustar la cadena entera, escribirá solo una parte de la cadena (hasta donde alcance el espacio).

Buffer: Lectura

```
buf.toString([encoding][, start][, end])
```

- Descripción de los parametros utilizados:
  - Encoding: Codificación a utilizar. 'Utf8' es la codificación predeterminada.
  - Start: Índice inicial para empezar a leer, el valor predeterminado es 0.
  - End: El índice final de la lectura, por defecto es el buffer completo.

- Valor del retorno: Este método decodifica y devuelve una cadena de datos lo que esta escrito en el buffer.

Buffer: Concatenar buffers

```
Buffer.concat(list[, totalLength])
```

- Descripción de los parametros utilizados:
  - List: Lista array de objetos de los buffers a concatenar.
  - TotalLength: Ésta es la longitud total de los buffers cuando están concatenados.

Streams

- Objetos que permiten leer datos de una Fuente o escribir datos en un destino en modo continuo:
- Cuatro tipos de streams:
  - **Readable** – Streams de lectura
  - **Writable** – Streams de escritura
  - **Duplex** – Stream de lectura/escritura
  - **Transform** – A type of duplex stream where the output is computed based on input.
- Cada tipo de Stream es una instancia de EventEmitter y lanza distintos **events** en diferentes instantes de tiempo. Los eventos más comunes son:
  - **data** – Evento de datos disponibles para lectura
  - **end** – Evento de no hay más datos de lectura
  - **error** – Error recibiendo o escribiendo datos
  - **finish** – Finalización de la tarea

Streams: Ejemplo de Lectura

- Crear un stream a partir de la lectura de un fichero utilizando el método createReadStream() del objeto fs

```
var fs = require('fs');
var streamLectura = fs.createReadStream('./archivo-texto.txt');
```

- Asociar un evento "data" a un listener que se ejecutará cuando el dato se haya leído y se encuentre disponible, recibiendo como parámetro un buffer de datos

```
streamLectura.on('data', (chunk) => {
//chunk es un buffer de datos
console.log(chunk instanceof Buffer);
//escribe "true" por pantalla
});
```

- Leemos su contenido:

```
streamLectura.on('data', (chunk) => {
console.log('He recibido ' + chunk.length + ' bytes de datos.');
console.log(chunk.toString());
});
```

Ejercicio 8

- Leer el fichero movieDetails.json y mostrar el contenido por consola. Asociar listeners a cada uno de los eventos: data, end, error

Streams: Ejemplo de Escritura

- Utilizar la propiedad **stdout** del objeto global process. Es un stream de escritura con el que se puede generar salida del programa. En este caso la consola:
- Mediante el método **write()** se escribe en un stream de escritura. Hay que enviarle un buffer y otra serie de parámetros opcionales como el tipo de codificación y una función callback a ejecutar cuando termine la operación de escritura.

```
process.stdout.write(objetoBuffer);
```

```
process.stdin.setEncoding('utf8');
process.stdout.write('¿Cómo estás hoy? ');
process.stdin.once('data', function(res) {
process.stdout.write('Has respondido: ');
process.stdout.write(res);
process.stdin.pause();
```

Ejercicio 8_bis

- Reescribir el ejercicio anterior utilizando la propiedad stdout en lugar de console.log
- Incorporar una petición mediante stdin para bajar el fichero.
- Combinar la salida por consola con los métodos de la propiedad process.stdin para entrar información por consola. Ej:

Streams: Ejemplo de Escritura

- Ejemplo con el método createWriteStream()

```
var fs = require("fs");
var data = 'Simply Easy Learning';

// Create a writable stream
var writerStream = fs.createWriteStream('output.txt');

// Write the data to stream with encoding to be utf8
writerStream.write(data,'UTF8');

// Mark the end of file
writerStream.end();

// Handle stream events --> finish, and error
writerStream.on('finish', function() {
   console.log("Write completed.");
});

writerStream.on('error', function(err){
  console.log(err.stack);
});

console.log("Program Ended");
```

Ejercicio 9

- Leer de la consola un texto y guardarlo en un fichero

Pipes:

- Conexión de un stream de lectura con un stream de escritura:

```
var streamLectura = fs.createReadStream('./archivo-texto.txt');
var streamEscritura = fs.createWriteStream('./otro-archivo.txt');
```

- Para conectar los dos streams se usa:

```
streamLectura.pipe(streamEscritura);
```

Ejercicio 9

- Copiar de un fichero a otro (ejercicio de pipes)

Streams: Encadenando streams

- Mecanismo para conectar la salida de un stream a múltiples streams mediante piping.
- Ejemplo generar archivo comprimido:

```
var fs = require("fs");
var zlib = require('zlib');

// Compress the file input.txt to input.txt.gz
fs.createReadStream('input.txt')
   .pipe(zlib.createGzip())
   .pipe(fs.createWriteStream('input.txt.gz'));

console.log("File Compressed.");
```

Ejercicio 10

- Descomprimir un fichero .gz creado con el ejemplo anterior. Buscar la documentación de la api 'zlib'

FileSystem

- En NodeJS todas las operaciones de acceso al sistema de archivos están englobadas dentro del módulo "fs" (File System)

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

- Los métodos del módulo fs existen en las dos alternativas: síncrona y asíncrona. Ej:

```
var fs = require("fs");


// Asynchronous read

fs.readFile('input.txt', function (err, data) {

   if (err) {

      return console.error(err);

   }

   console.log("Asynchronous read: " +
data.toString());

});
```

```
// Synchronous read

var data = fs.readFileSync('input.txt');

console.log("Synchronous read: " +
data.toString());



console.log("Program Ended");
```

FileSystem: Abrir un fichero

- Abrir un fichero en modo asíncrono: fs.open(path, flags[, mode], callback)

```
var fs = require("fs");


// Asynchronous - Opening File
console.log("Going to open file!");
fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
      return console.error(err);
   }
  console.log("File opened successfully!");
});
```

FileSystem: Sacar información de un fichero

- fs.stat(path, callback)

```
var fs = require("fs");


console.log("Going to get file info!");
fs.stat('input.txt', function (err, stats) {
   if (err) {
       return console.error(err);
   }
   console.log(stats);
   console.log("Got file info successfully!");


   // Check file type
   console.log("isFile ? " + stats.isFile());
   console.log("isDirectory ? " +
stats.isDirectory());
});
```

FileSystem: Escribir en un fichero

- fs.writeFile(filename, data[, options], callback)

```
var fs = require("fs");

console.log("Going to write into existing file");
fs.writeFile('input.txt', 'Simply Easy Learning!',
function(err) {
   if (err) {
      return console.error(err);
   }

   console.log("Data written successfully!");
   console.log("Let's read newly written data");
   fs.readFile('input.txt', function (err, data) {
      if (err) {
         return console.error(err);
      }
      console.log("Asynchronous read: " +
data.toString());
   });
});
```

FileSystem: Leer de un fichero

- fs.read(fd, buffer, offset, length, position, callback)

```
var fs = require("fs");
var buf = new Buffer(1024);

console.log("Going to open an existing file");
fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
      return console.error(err);
   }
   console.log("File opened successfully!");
   console.log("Going to read the file");
   fs.read(fd, buf, 0, buf.length, 0, function(err, bytes){
      if (err){
         console.log(err);
      }
      console.log(bytes + " bytes read");

      // Print only read bytes to avoid junk.
      if(bytes > 0){
         console.log(buf.slice(0, bytes).toString());
      }
   });
});
```

FileSystem: Cerrar un fichero

- fs.close(fd, callback)

```
var fs = require("fs");
var buf = new Buffer(1024);

console.log("Going to open an existing file");
fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
      return console.error(err);
   }
   console.log("File opened successfully!");
   console.log("Going to read the file");

   fs.read(fd, buf, 0, buf.length, 0, function(err, bytes){
      if (err){
         console.log(err);
      }

      // Print only read bytes to avoid junk.
      if(bytes > 0){
         console.log(buf.slice(0, bytes).toString());
      }

      // Close the opened file.
      fs.close(fd, function(err){
         if (err){
            console.log(err);
         }
         console.log("File closed successfully.");
      });
   });
});
```

FileSystem: Truncar un fichero

- fs.ftruncate(fd, len, callback)

```
var fs = require("fs");
var buf = new Buffer(1024);

console.log("Going to open an existing file");
fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
      return console.error(err);
   }
   console.log("File opened successfully!");
   console.log("Going to truncate the file after 10 bytes");

   // Truncate the opened file.
   fs.ftruncate(fd, 10, function(err){
      if (err){
         console.log(err);
      }
      console.log("File truncated successfully.");
      console.log("Going to read the same file");
      (...)
```

```
(...)
fs.read(fd, buf, 0, buf.length, 0, function(err, bytes){
         if (err){
            console.log(err);
         }

         // Print only read bytes to avoid junk.
         if(bytes > 0){
            console.log(buf.slice(0, bytes).toString());
         }

         // Close the opened file.
         fs.close(fd, function(err){
            if (err){
               console.log(err);
            }
            console.log("File closed successfully.");
         });
      });
   });
});
```

FileSystem: Borrar un fichero

- fs.unlink(path, callback)

```
var fs = require("fs");

console.log("Going to delete an existing file");
fs.unlink('input.txt', function(err) {
   if (err) {
      return console.error(err);
   }
   console.log("File deleted successfully!");
});
```

FileSystem: Crear un directorio

- fs.mkdir(path[, mode], callback)

```
var fs = require("fs");

console.log("Going to create directory /tmp/test");
fs.mkdir('/tmp/test',function(err){
   if (err) {
      return console.error(err);
   }
   console.log("Directory created successfully!");
});
```

FileSystem: Leer un directorio

- fs.readdir(path, callback)

```
var fs = require("fs");

console.log("Going to read directory /tmp");
fs.readdir("/tmp/",function(err, files){
   if (err) {
      return console.error(err);
   }
   files.forEach( function (file){
      console.log( file );
   });
});
```

FileSystem: Eliminar un directorio

- fs.rmdir(path, callback)

```
var fs = require("fs");

console.log("Going to delete directory /tmp/test");
fs.rmdir("/tmp/test",function(err){
   if (err) {
      return console.error(err);
   }
   console.log("Going to read directory /tmp");

   fs.readdir("/tmp/",function(err, files){
      if (err) {
         return console.error(err);
      }
      files.forEach( function (file){
         console.log( file );
      });
   });
});
```

Ejercicio A entregar

- Crear un módulo para empaquetar los ejercicios de node sin incluir los directorios node_modules.