4. LECTURA DE FICHEROS

1. Importar el módulo fs.

Lo primero que se debe hacer es importar el módulo fs.

const fs = require('fs');

2. Función fs.readFile:

La función fs.readFile se utiliza para leer el contenido de un archivo, tiene 3 argumentos: la ruta del archivo, la codificación del archivo ('utf8') y la función de devolución de llamada que se ejecutará una vez que la lectura del archivo esté completa.

```
const archivo = 'archivo.txt';

fs.readFile(archivo, 'utf8', (error, contenido) => {
  if (error) {
    console.error(`Error al leer el archivo: ${error.message}`);
    return;
  }
 console.log('Contenido del archivo:');
  console.log(contenido);
});
```

5. CREATESERVER

Esta función pertenece al módulo **http** en Node.js. Se utiliza para crear un servidor HTTP, es decir, cuando se realiza la llamada de la creación se genera una instancia de un servidor HTTP que podrá ser configurado para manejar solicitudes y enviar respuestas.

```
const http = require('http'); //IMPORTAR EL MÓDULO PARA CREACIÓN DE SERVIDORES HTTP

const servidor = http.createServer((solicitud, respuesta) => {
```

LÓGICA DE LA SOLICITUD Y RESPUESTA. Esta función se ejecutará cada vez que el servidor reciba una solicitud. A través de la respuesta se puede enviar datos al cliente

```
});
```

Se inicia el servidor con LISTEN

```
const puerto = 3000;

servidor.listen(puerto, () => {

  console.log(`Servidor corriendo en http://localhost:${puerto}/`);

});
```

Información a la que se puede acceder sobre la solicitud (https://nodejs.org/dist/latest-v14.x/docs/api/http.html#http_class_http_incomingmessage) :

1. request.url: Contiene la URL de la solicitud.

```
  const urlSolicitada = request.url;

  console.log(`URL solicitada: ${urlSolicitada}`);
```

2. request.method: Contiene el método HTTP utilizado en la solicitud.

```
  const metodoSolicitado = request.method;

  console.log(`Método solicitado: ${metodoSolicitado}`);
```

3. request.headers: Contiene los encabezados de la solicitud como un objeto.

```
  const headers = request.headers;

  console.log('Encabezados de la solicitud:', headers);
```

4. request.httpVersion: Contiene la versión de HTTP utilizada en la solicitud (por ejemplo, '1.1').

```
  const versionHTTP = request.httpVersion;

  console.log(`Versión de HTTP: ${versionHTTP}`);
```

6. FUNCIÓN URL

La función principal del módulo url es facilitar el análisis y la manipulación de componentes de una URL.

1. url.parse(urlStr[, parseQueryString): Toma la cadena de URL (urlStr) y devuelve un objeto que contiene diferentes componentes de la URL.

- **parseQueryString**: Un booleano que indica si se deben analizar los parámetros de la cadena de consulta. Por defecto, es false.

```
const url = require('url');
const urlString = 'https://www.ejemplo.com:8080/ruta?param1=valor1&param2=valor2';

const parsedUrl = url.parse(urlString, true);
console.log(parsedUrl);
```

parsedUrl contendrá propiedades como:

1. protocol: epresenta el protocolo utilizado en la URL (por ejemplo, 'http'.
2. auth: representa las credenciales de autenticación (usuario y contraseña) en la URL.
3. pathname: representa la ruta del recurso en la URL.
4. host: representa la parte del dominio de la URL, incluyendo el puerto si está presente. (**www.ejemplo.com:8080**).
5. port: representa el número de puerto de la URL. (8080).
6. hostname: como host, pero excluye el número de puerto si está presente. (**www.ejemplo.com**).
7. href: representa la URL completa como una cadena (**https:// www.ejemplo.com:8080/ruta1/ruta2?param1=valor1**).

2. url.format(urlObject): Esta función toma un objeto de URL y lo convierte en una cadena de URL.

```
const url = require('url');
const urlObject = {
  protocol: 'https',
  host: 'www.ejemplo.com',
  pathname: '/ruta',
  query: { param1: 'valor1', param2: 'valor2' }
};

const formattedUrl = url.format(urlObject);
console.log(formattedUrl);
```

El resultado será la cadena de URL completa.

3. url.resolve(from, to): Combina dos partes de una URL.
- **from:** La URL base.
- **to:** La porción de URL que se va a combinar con la URL base.

```
const url = require('url');
const baseUrl = 'https://www.ejemplo.com/ruta1/';
const relativePath = 'subruta';

const resolvedUrl = url.resolve(baseUrl, relativePath);
console.log(resolvedUrl);
```

El resultado será la URL combinada: `https://www.ejemplo.com/ruta1/subruta`.

7. FUNCIÓN PATH

Permite trabajar con rutas de archivos y directorios.

1. path.join(…): se utiliza para unir segmentos de ruta en una ruta completa. Toma uno o más argumentos que representan los segmentos de la ruta y devuelve la ruta completa resultante.

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

const rutaCompleta = path.join('/ruta1', 'ruta2', 'archivo.txt');

console.log(rutaCompleta);
```

Resultado: /ruta1/ruta2/archivo.txt.

2. path.basename(path[, ext]): devuelve el último segmento de la ruta (nombre de archivo).

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

const nombreArchivo = path.basename('/ruta1/ruta2/archivo.txt');

console.log(nombreArchivo)
```

Resultado: archivo.txt.

3**. path.dirname(path):** devuelve el directorio de una ruta.

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

const directorio = path.dirname('/ruta1/ruta2/archivo.txt');

console.log(directorio);

Resultado: /ruta1/ruta2.

4. path.extname(path): devuelve la extensión de un archivo en la ruta.

const path = require('path');

const extension = path.extname('/ruta1/ruta2/archivo.txt');

console.log(extension);

Resultado: .txt.

5. path.parse(pathString): descompone una ruta en un objeto que contiene propiedades del elemento path (se exponen más abajo).

const path = require('path');

const infoRuta = path.parse('/ruta1/ruta2/archivo.txt');

console.log(infoRuta);

Resultado: Un objeto con propiedades como { root: '/', dir: '/ruta1/ruta2', base: 'archivo.txt', ext: '.txt', name: 'archivo' }.

1. root: representa el componente raíz de la ruta. Es la porción de la ruta antes de cualquier directorio o nombre de archivo.

2. dir: representa el directorio de la ruta. Es la porción de la ruta que contiene todos los directorios, pero no el nombre del archivo.

const path = require('path');

const infoRuta = path.parse('/ruta1/ruta2/archivo.txt');

console.log(infoRuta.dir);

// Resultado: '/ruta1/ruta2'

3. base: representa el nombre de archivo con su extensión. Es la combinación del nombre del archivo y la extensión.

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

const infoRuta = path.parse('/ruta1/ruta2/archivo.txt');

console.log(infoRuta.base);

// Resultado: 'archivo.txt'
```

4. ext: representa la extensión del archivo. Es la porción de la ruta después del último punto (`.`).

5. name: representa el nombre del archivo sin la extensión. Es el nombre del archivo antes del último punto (`.`).

8. __dirname

Variable global en Node.js que representa el nombre del directorio del archivo actualmente en ejecución. Proporciona la ruta completa del directorio que contiene el script en el que se está utilizando.

Cuando se ejecuta este código en un script de Node.js, `__dirname` imprimirá la ruta completa del directorio que contiene el script.

9. FUNCIÓN RES.END

Se utiliza para enviar la respuesta HTTP al cliente. En el contexto de un servidor web, la respuesta se envía al navegador o a la entidad que realizó la solicitud HTTP.

- res.end(contenido):

- res: Es el objeto de respuesta que se pasa como argumento a la función de devolución de llamada del servidor HTTP. Representa la respuesta que se enviará al cliente.
- end: Es un método de `res` que indica que la respuesta se ha completado y se debe enviar al cliente.
- contenido: Es el contenido que se enviará como parte del cuerpo de la respuesta. En este contexto, `contenido` suele ser el contenido del archivo que se leyó.

La función `res.end` es fundamental para finalizar la respuesta HTTP y enviarla al cliente. Cuando se llama a `res.end`, Node.js enviará la respuesta al cliente, completando la transacción HTTP.

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

const server = http.createServer((req, res) => {

    // Código para manejar la solicitud

   // Supongamos que se ha leído el contenido de un archivo y se almacena en la variable 'contenido'

  const contenido = "¡Hola, mundo! Este es el contenido del archivo.";

  // Enviar el contenido como respuesta al cliente

  res.writeHead(200, { 'Content-Type': 'text/plain' });

  res.end(contenido);

});


const puerto = 3000;

server.listen(puerto, () => {

  console.log(`Servidor corriendo en http://localhost:${PORT}/`);

});
```