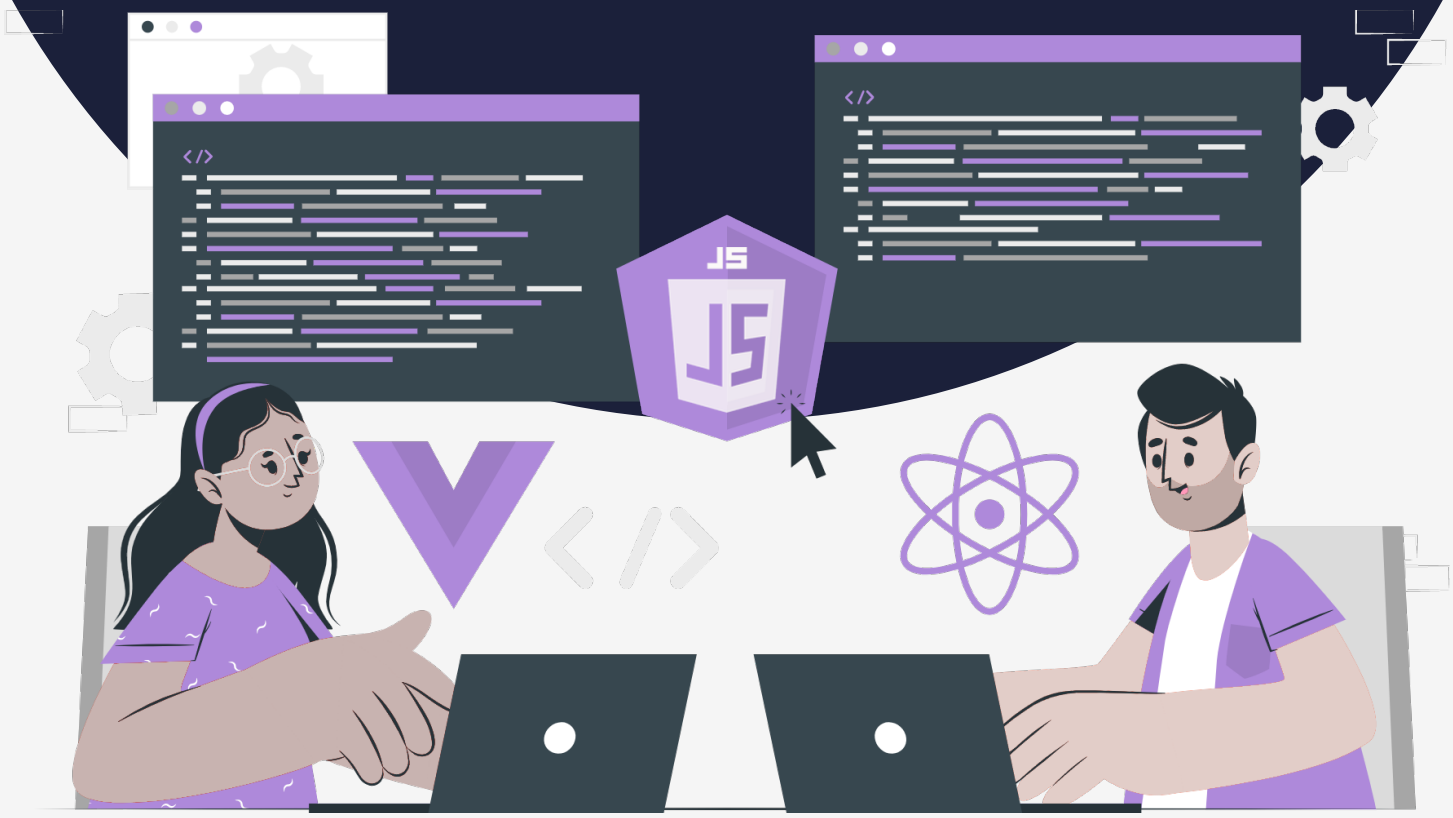
Lesson:

File system Module



List of content:

Node.js is a platform that uses Chrome's V8 JavaScript engine, allowing developers to use JavaScript to create server-side applications that generate dynamic content for web clients. What sets Node.js apart are its event-driven and non-blocking I/O features. Node.js includes a built-in module called FS (File System) that allows users to manage files by creating, reading, deleting, and performing other file operations. To use this module, developers can call the "require()" method, which provides access to POSIX functions wrapped by Node.js to **enable both synchronous and asynchronous file system operations**, depending on the user's requirements.

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

Common use for File System module:

- **Read Files**
- **Write Files**
- **Append Files**
- **Close Files**
- **Delete Files**

Synchronous vs Asynchronous approach :

The ***Synchronous approach*** **involves blocking functions** that wait for each operation to complete before executing the next one. This means that a command won't execute until the query has finished executing and all the results from previous commands have been obtained. On the other hand, the ***Asynchronous approach*** **involves non-blocking functions** that execute all operations at once, without waiting for each operation to complete. The results of each operation are handled when they become available, and each command is executed after the previous one. If the operations involve **querying large amounts of data from a database, the Asynchronous approach is recommended.** In this approach, progress indicators can be shown to the user while the heavy work continues in the background, which is ideal for GUI-based applications.

Example : Create a text file named input.txt with the following content:
PW Skills

Asynchronous read :

```
var fs = require("fs");
fs.readFile('input.txt', function (err, data) {
   if (err) {
      return console.error(err);
   }
   console.log("Asynchronous read: " + data.toString());
});
```

Output :

```
Asynchronous read:PW Skills
```

Synchronous Read :

```
var fs = require("fs");
var data = fs.readFileSync('input.txt');
console.log("Synchronous read: " + data.toString());
```

Output :

```
Synchronous read: PW Skills
```

Opening a file:
To perform operations like creating, reading, or writing a file in Node.js, the "fs" module provides several methods, including the "fs.open()" method. Unlike the "fs.readFile()" and "fs.writeFile()" methods, which are limited to reading or writing files, the "fs.open()" method can perform multiple operations on a file. To use this method, you first need to load the "fs" module, which provides access to the physical file system.

```
fs.open(path, flags, mode, callback)
```

Parameters:
path: It holds the name of the file to read or the entire path if stored at other locations.
flags: Flags indicate the behavior of the file to be opened. All possible values are (r, r+, rs, rs+, w, wx, w+, wx+, a, ax, a+, ax+).
mode: Sets the mode of file i.e. r-read, w-write, r+ -readwrite. It sets to default as readwrite.
err: If any error occurs.
data: Contents of the file. It is called after the open operation is executed.

Example: Create a js file named main.js having the following code to open a file input.txt for reading and writing.

Asynchronous - Opening File

```
var fs = require("fs");
console.log("opening…");
fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
      return console.error(err);
   }
   console.log("File open successfully");
});
```

Output :

```
opening...File open successfully
```

Reading a File:
The fs.read() method is used to read the file specified by fd. This method reads the entire file into the buffer.

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

Parameters:

fd: This is the file descriptor returned by fs.open() method.

buffer: This is the buffer that the data will be written to.

offset: This is the offset in the buffer to start writing at.

length: This is an integer specifying the number of bytes to read.

position: This is an integer specifying where to begin reading from in the file. If the position is null, data will be read from the current file position.

callback: It is a callback function that is called after reading of the file. It takes two parameters:

err: If any error occurs.

data: Contents of the file.

Example: Create a js file named main.js having the following code.

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

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

Writing to a file:
The "fs.writeFile()" method is used to write data to a file asynchronously in Node.js. If the file already exists, this method will overwrite it by default. However, the behavior of this method can be modified using the "options" parameter.

```
fs.writeFile(path, data, options, callback)
```

Parameters:

path: It is a string, Buffer, URL, or file description integer that denotes the path of the file where it has to be written. Using a file descriptor will make it behave similarly to fs.write() method.

data: It is a string, Buffer, TypedArray, or DataView that will be written to the file.

options: It is a string or object that can be used to specify optional parameters that will affect the output. It has three optional parameters:

encoding: It is a string value that specifies the encoding of the file. The default value is 'utf8'.

mode: It is an integer value that specifies the file mode. The default value is 0o666.

flag: It is a string value that specifies the flag used while writing to the file. The default value is 'w'.

callback: It is the function that would be called when the method is executed.

err: It is an error that would be thrown if the operation fails.

Example: Create a js file named main.js having the following code:

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

console.log("writing into existing file");
fs.writeFile('input.txt', 'PW Skills'', 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());
   });
});
```

Output:

```
writing into existing fileData written
successfully!Let's read newly written
dataAsynchronous read: PW Skills
```

Appending to a File

The fs.appendFile() method is used to synchronously append the data to the file.

```
fs.appendFile(filepath, data, options, callback);
```

or

```
fs.appendFileSync(filepath, data, options);
```

Parameters explanation:

filepath: String that specifies the file path.

data: Mandatory and contains the data that you append to the file.

options: Optional parameter that specifies the encoding/mode/flag.

Callback: Mandatory function and is called when appending data to file is completed.

Example : Create a js file named main.js having the following code:

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

var data = "\nLearn Node.js";

// Append data to file
fs.appendFile('input.txt', data, 'utf8',

    // Callback function
    function(err) {
        if (err) throw err;

        //  If no error
        console.log("Data is appended to file successfully.")
});
```

Output:

```
Data is appended to file successfully.
```

Example : For synchronously appending

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

var data = "\nLearn Node.js";

// Append data to file
fs.appendFileSync('input.txt', data, 'utf8');
console.log("Data is appended to file successfully.")
```

Output:

```
Data is appended to file successfully.
```

Before Appending Data to input.txt file:

```
PW Skills
```

After Appending Data to input.txt file:

```
PW Skills
Learn Node.js
```

Closing the file

The function fs.close() is utilized to close a specific file descriptor in an asynchronous manner, which effectively erases the file linked with it. Once the file is closed, the file descriptor can be used for other files. However, if fs.close() is executed on a file descriptor while another operation is still in progress, the outcome may be uncertain.

```
fs.close(fd, callback)
```

Parameters:

fd: It is an integer that denotes the file descriptor of the file for which to be closed.

callback: It is a function that would be called when the method is executed.

err: It is an error that would be thrown if the method fails.

Example: Create a js file named main.js having the following code

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

Output:

```
File closed successfully.
```

Delete a File

The fs.unlink() method is used to remove a file or symbolic link from the filesystem. This function does not work on directories, therefore it is recommended to use fs.rmdir() to remove a directory.

```
fs.unlink(path, callback)
```

Parameters:

path: It is a string, Buffer or URL which represents the file or symbolic link which has to be removed.

callback: It is a function that would be called when the method is executed.

err: It is an error that would be thrown if the method fails.

Example: Create a js file named main.js having the following code:

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

console.log("deleting...");
fs.unlink('input.txt', function(err) {
   if (err) {
      return console.error(err);
   }
   console.log("File deleted successfully!");
});
```

Output:

```
deleting...
File deleted successfully!
```