

# Node.js Documentation

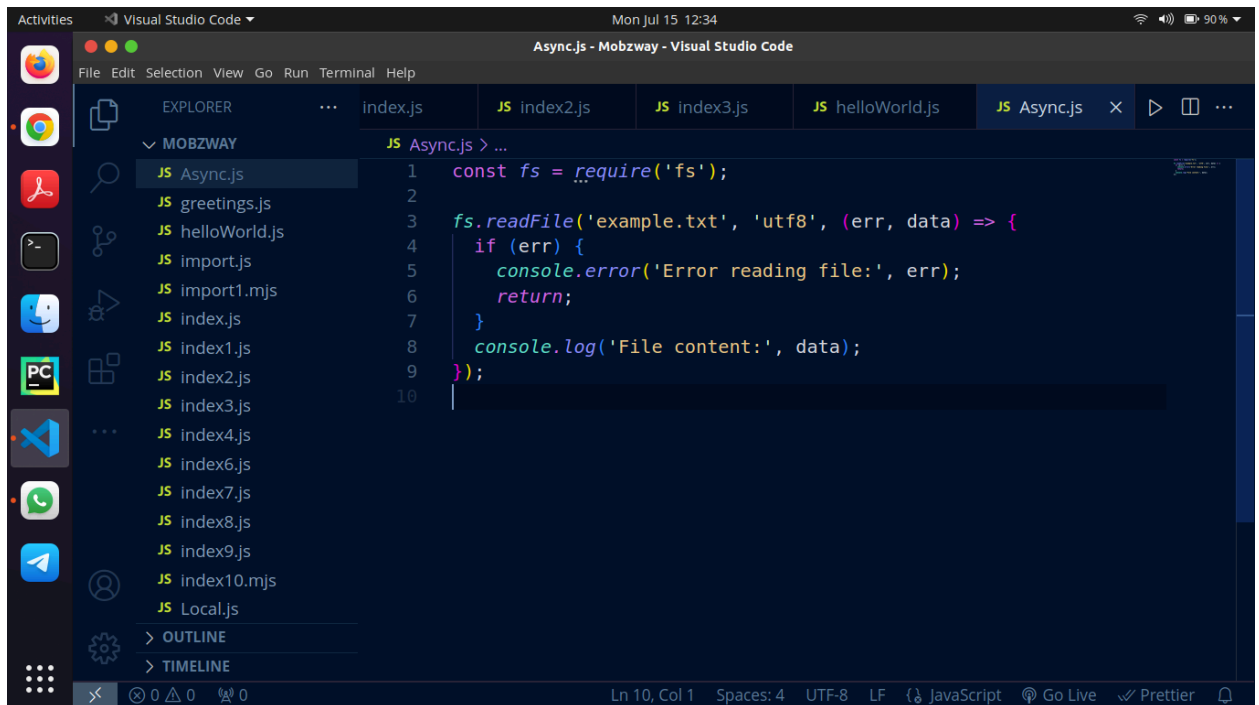
## Module 5: File System

- **Read File**

In Node.js, you can read files using the built-in `fs` (File System) module. There are multiple ways to read files, including synchronous and asynchronous methods. Here's how you can read files using both approaches:

### Asynchronous File Reading

Using the asynchronous method is generally preferred for non-blocking code execution, which is crucial for performance in a Node.js environment.

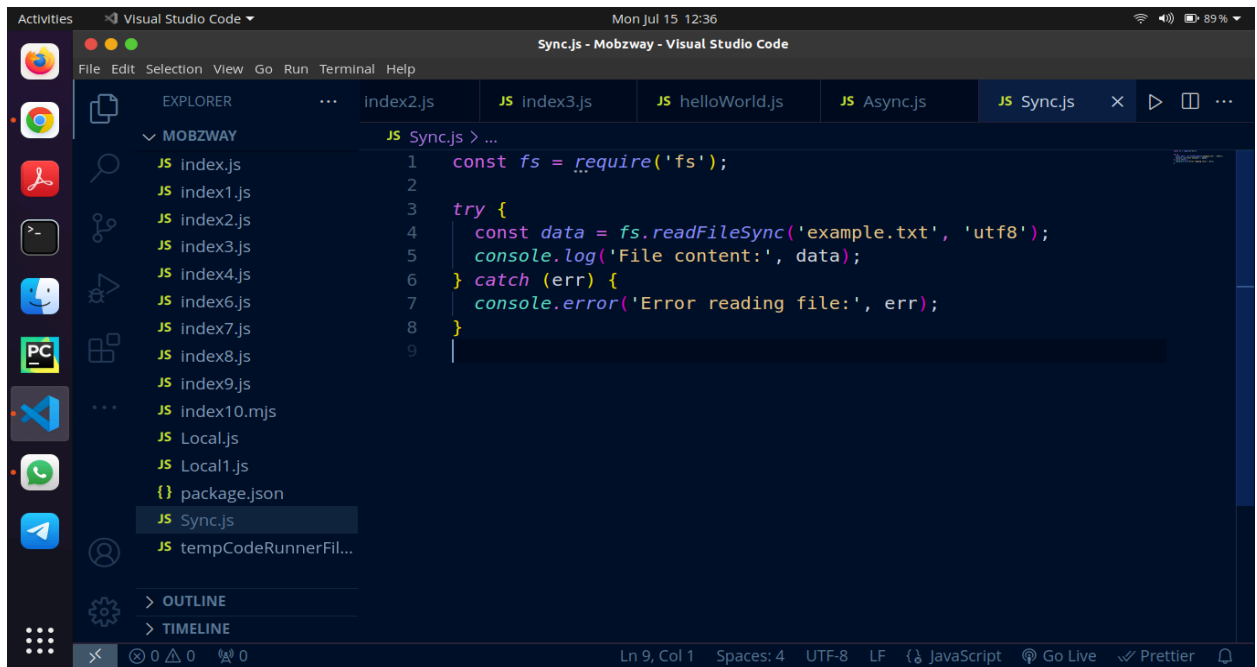
A screenshot of the Visual Studio Code editor interface. The title bar shows 'Visual Studio Code' and 'Mon Jul 15 12:34'. The menu bar includes 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help'. The Explorer sidebar on the left shows a project named 'MOBZWAY' with a list of files: 'JS Async.js', 'JS greetings.js', 'JS helloWorld.js', 'JS import.js', 'JS import1.mjs', 'JS index.js', 'JS index1.js', 'JS index2.js', 'JS index3.js', 'JS index4.js', 'JS index6.js', 'JS index7.js', 'JS index8.js', 'JS index9.js', 'JS index10.mjs', and 'JS Local.js'. The main editor area displays the code for 'JS Async.js' with the following content:

```
1  const fs = require('fs');
2
3  fs.readFile('example.txt', 'utf8', (err, data) => {
4    if (err) {
5      console.error('Error reading file:', err);
6      return;
7    }
8    console.log('File content:', data);
9  });
10
```

The status bar at the bottom indicates 'Ln 10, Col 1', 'Spaces: 4', 'UTF-8', 'LF', 'JavaScript', 'Go Live', and 'Prettier'.

### Synchronous File Reading

Synchronous methods are simpler but can block the event loop, making them less suitable for performance-critical applications.



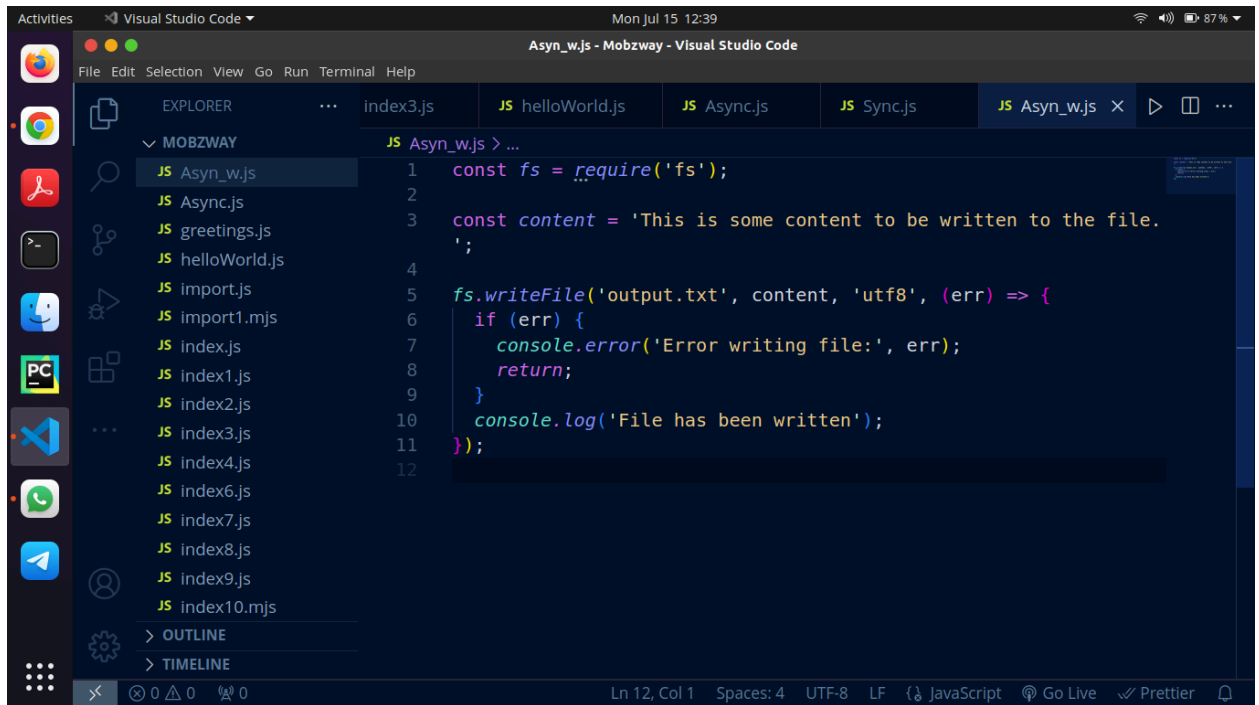
```
1 const fs = require('fs');
2
3
4 try {
5   const data = fs.readFileSync('example.txt', 'utf8');
6   console.log('File content:', data);
7 } catch (err) {
8   console.error('Error reading file:', err);
9 }
```

- **Writing a File**

Writing a file in Node.js can be done using the built-in **fs** (File System) module. Like reading files, you have multiple options to write files, including synchronous and asynchronous methods. Here's how you can write files using both approaches:

### **Asynchronous File Writing**

Asynchronous file writing is preferred to avoid blocking the event loop.



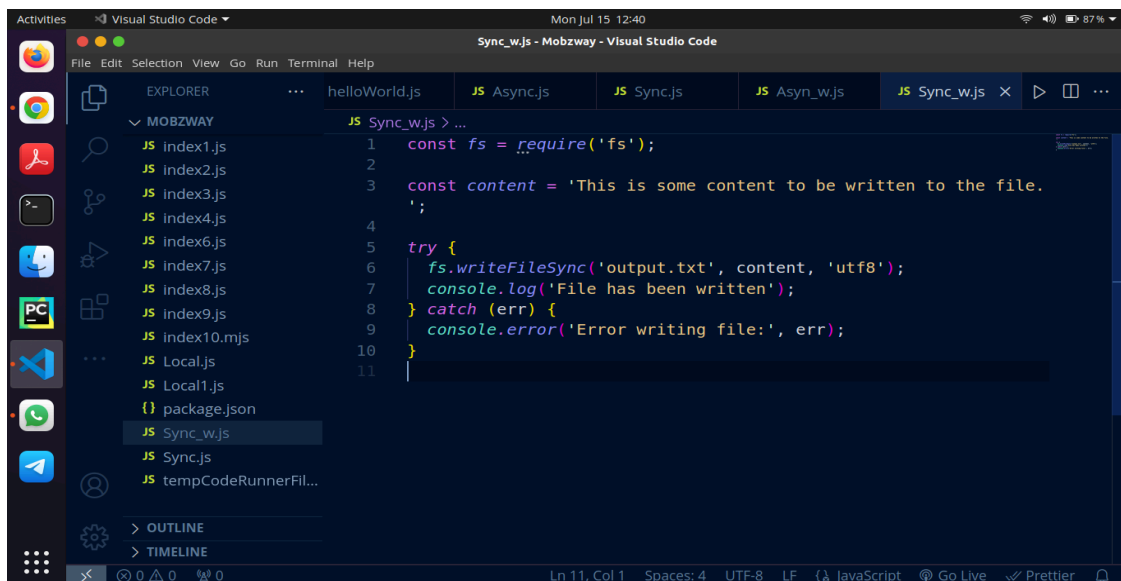
The screenshot shows the Visual Studio Code interface with the file explorer on the left displaying a project named 'MOBZWAY'. The file 'Asyn\_w.js' is selected. The main editor shows the following JavaScript code:

```
1 const fs = require('fs');
2
3 const content = 'This is some content to be written to the file.';
4
5 fs.writeFile('output.txt', content, 'utf8', (err) => {
6   if (err) {
7     console.error('Error writing file:', err);
8     return;
9   }
10  console.log('File has been written');
11 });
12
```

The status bar at the bottom indicates 'Ln 12, Col 1', 'Spaces: 4', 'UTF-8', 'LF', and 'JavaScript'.

## Synchronous File Writing

Synchronous methods are simpler but can block the event loop.



The screenshot shows the Visual Studio Code interface with the file explorer on the left displaying a project named 'MOBZWAY'. The file 'Sync\_w.js' is selected. The main editor shows the following JavaScript code:

```
1 const fs = require('fs');
2
3 const content = 'This is some content to be written to the file.';
4
5 try {
6   fs.writeFileSync('output.txt', content, 'utf8');
7   console.log('File has been written');
8 } catch (err) {
9   console.error('Error writing file:', err);
10 }
11
```

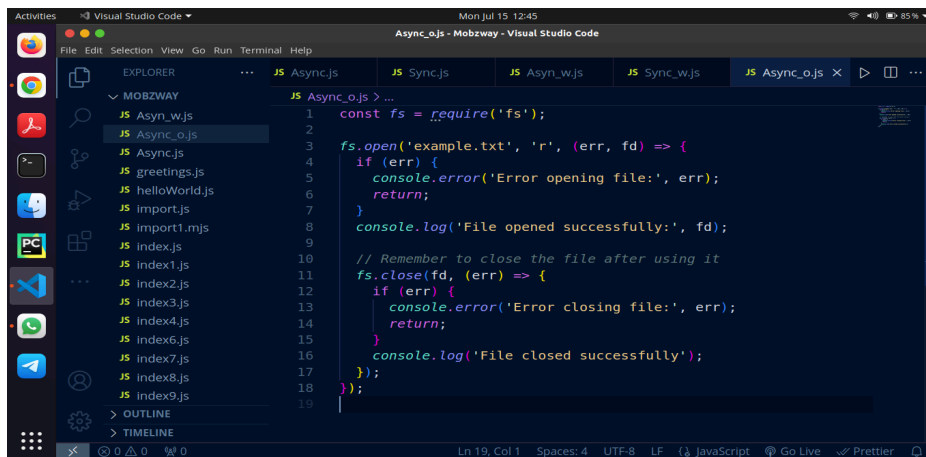
The status bar at the bottom indicates 'Ln 11, Col 1', 'Spaces: 4', 'UTF-8', 'LF', and 'JavaScript'.

- **Opening a File**

In Node.js, opening a file involves using the built-in `fs` (File System) module. You can open a file to read from or write to it using asynchronous or synchronous methods. Opening a file provides you with a file descriptor, which you can then use for various file operations.

### **Asynchronous File Opening**

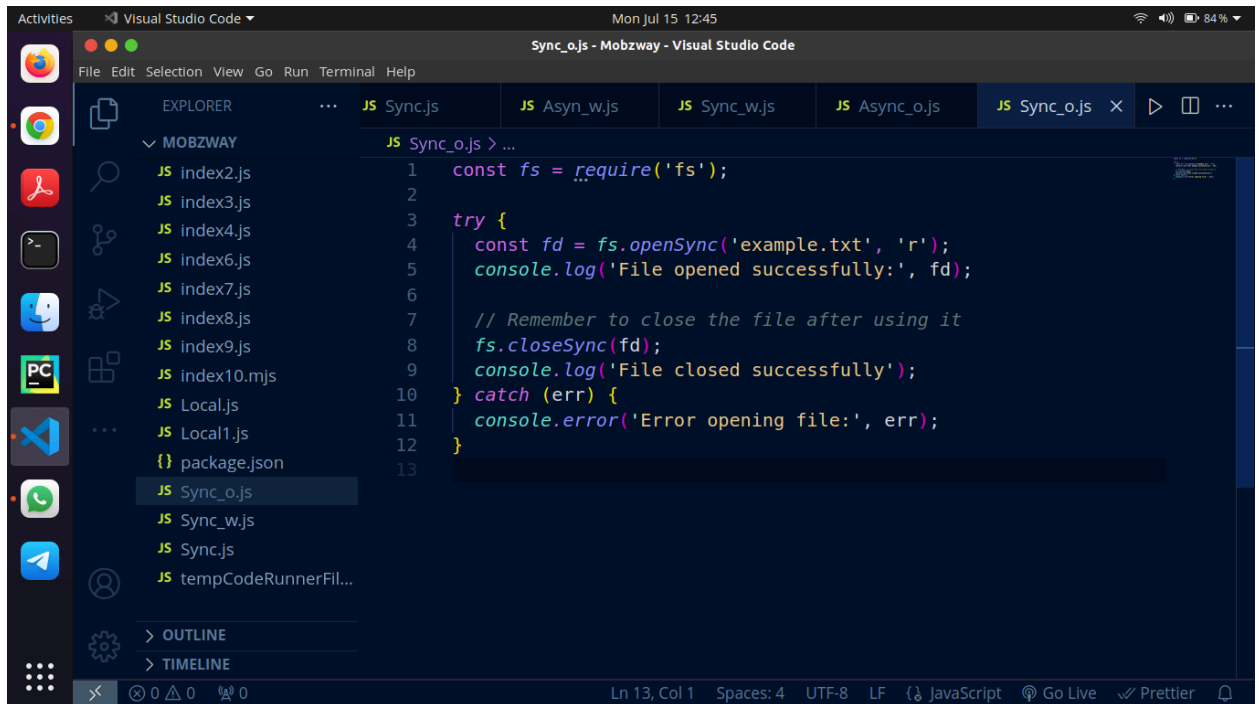
Using the asynchronous method is preferred for non-blocking code execution, which is crucial for performance in a Node.js environment.

A screenshot of the Visual Studio Code editor interface. The Explorer sidebar on the left shows a project named 'MOBZWAY' with several JavaScript files. The main editor window displays the code for 'JS Async\_o.js'. The code uses the 'fs' module's asynchronous 'open' and 'close' methods with callback functions. It includes error handling with 'console.error' and success logging with 'console.log'. The status bar at the bottom indicates the current position is Line 19, Column 1, with 4 spaces, UTF-8 encoding, and LF line endings. The file is identified as JavaScript and uses the Prettier formatter.

```
1 const fs = require('fs');
2
3 fs.open('example.txt', 'r', (err, fd) => {
4   if (err) {
5     console.error('Error opening file:', err);
6     return;
7   }
8   console.log('File opened successfully:', fd);
9
10  // Remember to close the file after using it
11  fs.close(fd, (err) => {
12    if (err) {
13      console.error('Error closing file:', err);
14      return;
15    }
16    console.log('File closed successfully');
17  });
18 });
19
```

### **Synchronous File Opening**

Synchronous methods are simpler but can block the event loop, making them less suitable for performance-critical applications.

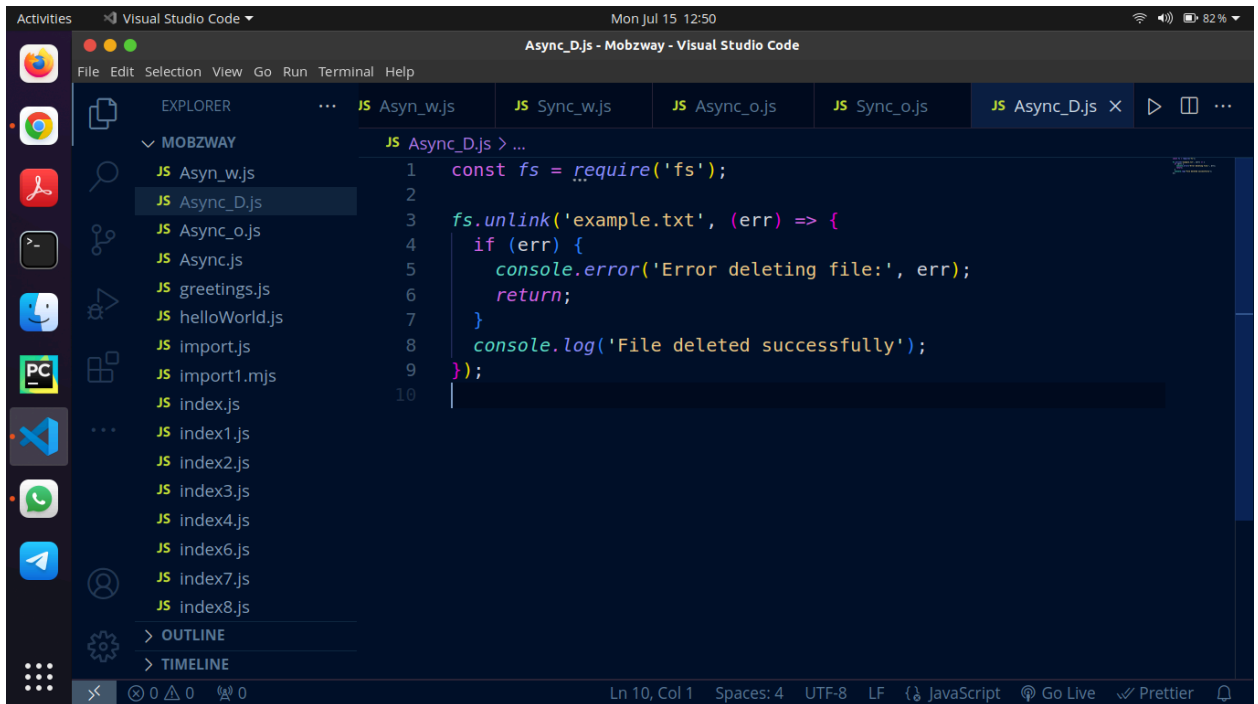


- **Deleting a File**

Deleting a file in Node.js can be done using the `fs` (File System) module. There are both asynchronous and synchronous methods available for this task.

### **Asynchronous File Deletion**

The asynchronous method is preferred to avoid blocking the event loop.



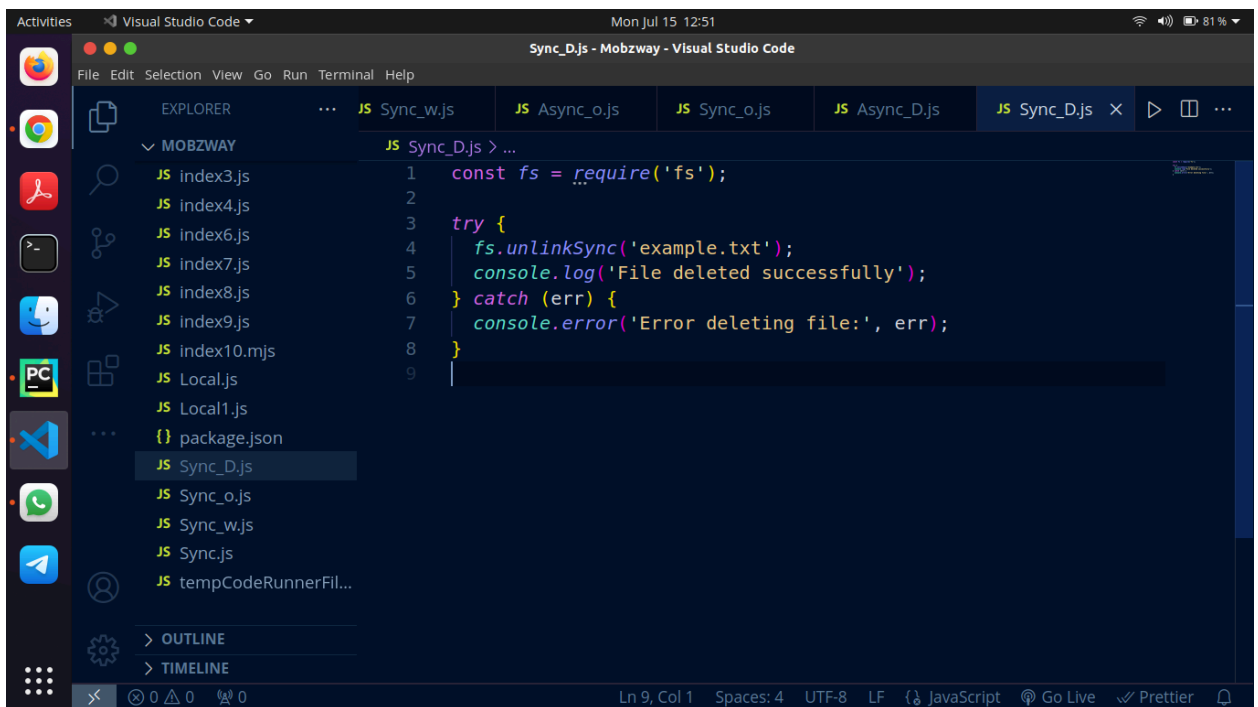
The screenshot shows the Visual Studio Code interface with the file explorer on the left displaying a project named 'MOBZWAY'. The file 'JS Async\_Djs' is selected. The code editor shows the following JavaScript code:

```
1 const fs = require('fs');
2
3 fs.unlink('example.txt', (err) => {
4   if (err) {
5     console.error('Error deleting file:', err);
6     return;
7   }
8   console.log('File deleted successfully');
9 });
10
```

The status bar at the bottom indicates 'Ln 10, Col 1', 'Spaces: 4', 'UTF-8', 'LF', and 'JavaScript'.

## Synchronous File Deletion

The synchronous method is simpler but can block the event loop.



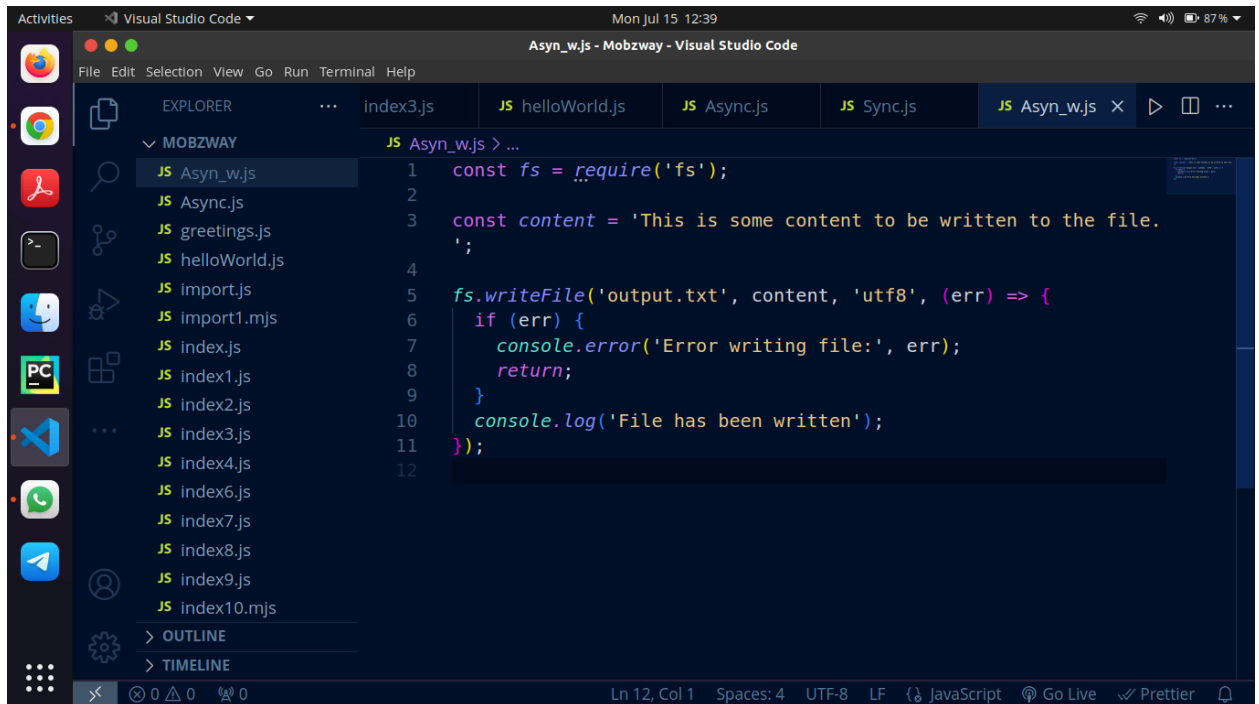
The screenshot shows the Visual Studio Code interface with the file explorer on the left displaying a project named 'MOBZWAY'. The file 'JS Sync\_Djs' is selected. The code editor shows the following JavaScript code:

```
1 const fs = require('fs');
2
3 try {
4   fs.unlinkSync('example.txt');
5   console.log('File deleted successfully');
6 } catch (err) {
7   console.error('Error deleting file:', err);
8 }
9
```

The status bar at the bottom indicates 'Ln 9, Col 1', 'Spaces: 4', 'UTF-8', 'LF', and 'JavaScript'.

- **Writing a file asynchronously**

Asynchronous file writing is preferred to avoid blocking the event loop.



```
1  const fs = require('fs');
2
3  const content = 'This is some content to be written to the file.';
4
5  fs.writeFile('output.txt', content, 'utf8', (err) => {
6    if (err) {
7      console.error('Error writing file:', err);
8      return;
9    }
10   console.log('File has been written');
11 });
12
```

In this example:

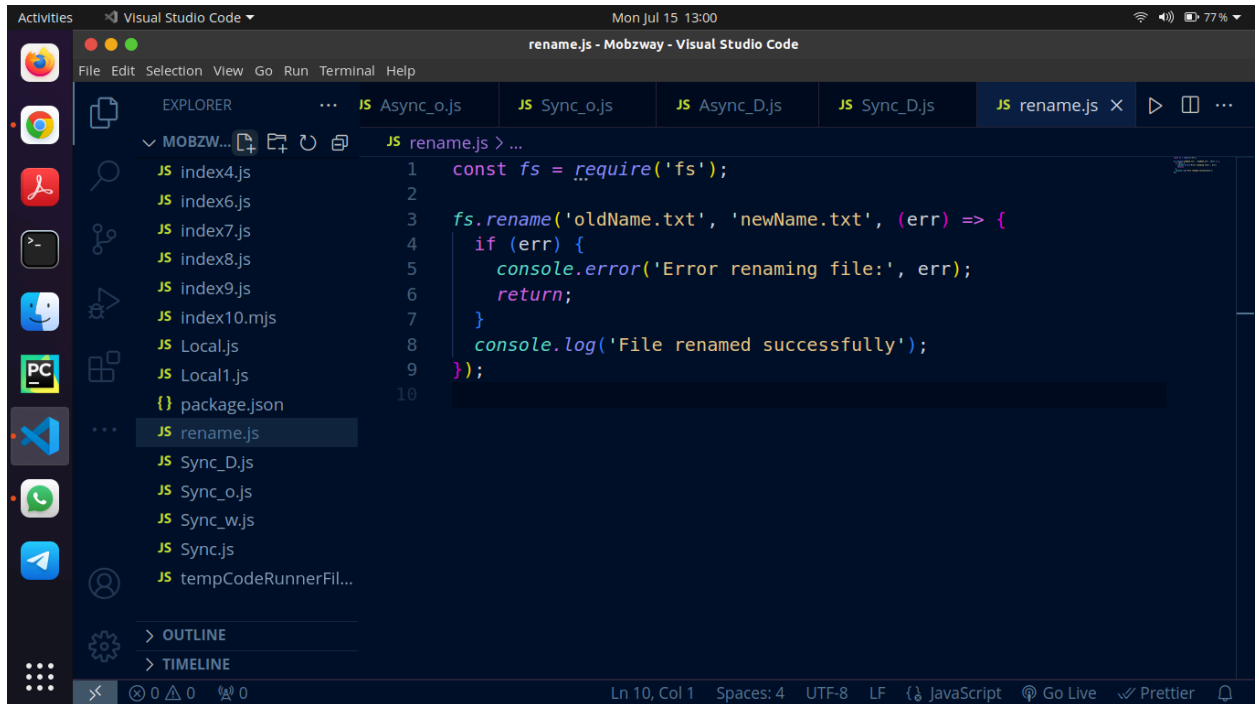
- `fs.writeFile` writes the file asynchronously.
- The first argument is the path to the file.
- The second argument is the content to be written.
- The third argument is the encoding (optional, but `utf8` is commonly used).
- The fourth argument is a callback function that handles any error that occurs.

- **Other I/O Operations**

In Node.js, the `fs` (File System) module provides various I/O operations for working with the file system. Here are some common I/O operations apart from reading, writing, and deleting files:

## Renaming a File

You can rename or move a file using the `fs.rename` method.



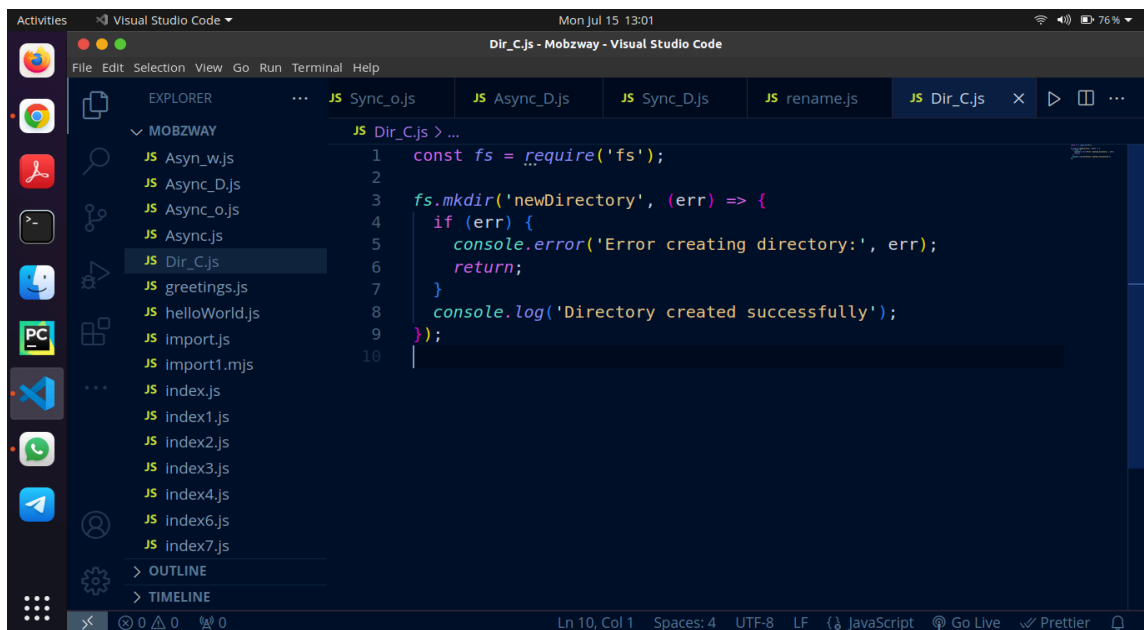
The screenshot shows the Visual Studio Code editor with the file `rename.js` open. The Explorer sidebar on the left shows a project named `MOBZW...` with various files including `index4.js` through `index10.mjs`, `Local.js`, `Local1.js`, `package.json`, `rename.js`, `Sync_D.js`, `Sync_o.js`, `Sync_w.js`, `Sync.js`, and `tempCodeRunnerFil...`. The main editor area displays the following JavaScript code:

```
1 const fs = require('fs');
2
3 fs.rename('oldName.txt', 'newName.txt', (err) => {
4   if (err) {
5     console.error('Error renaming file:', err);
6     return;
7   }
8   console.log('File renamed successfully');
9 });
10
```

The status bar at the bottom indicates the cursor is at line 10, column 1, with 4 spaces, UTF-8 encoding, and LF line endings. It also shows the active language as JavaScript and the Prettier formatter.

## Creating a Directory

You can create a new directory using the `fs.mkdir` method.



The screenshot shows the Visual Studio Code editor with the file `Dir_C.js` open. The Explorer sidebar on the left shows a project named `MOBZWAY` with various files including `Asyn_w.js`, `Asyn_D.js`, `Asyn_o.js`, `Asyn.js`, `Dir_C.js`, `greetings.js`, `helloWorld.js`, `import.js`, `import1.mjs`, `index.js`, `index1.js`, `index2.js`, `index3.js`, `index4.js`, `index6.js`, and `index7.js`. The main editor area displays the following JavaScript code:

```
1 const fs = require('fs');
2
3 fs.mkdir('newDirectory', (err) => {
4   if (err) {
5     console.error('Error creating directory:', err);
6     return;
7   }
8   console.log('Directory created successfully');
9 });
10
```

The status bar at the bottom indicates the cursor is at line 10, column 1, with 4 spaces, UTF-8 encoding, and LF line endings. It also shows the active language as JavaScript and the Prettier formatter.