Node.js Documentation

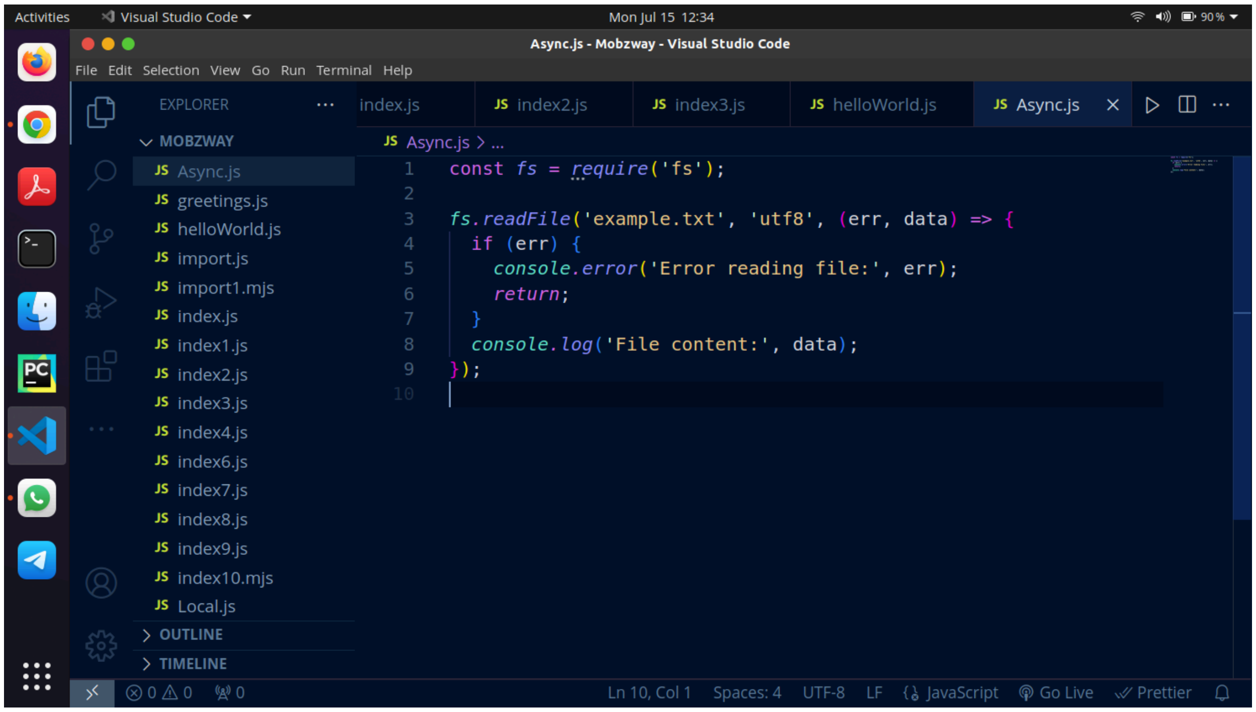
Module 5: File System

- **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

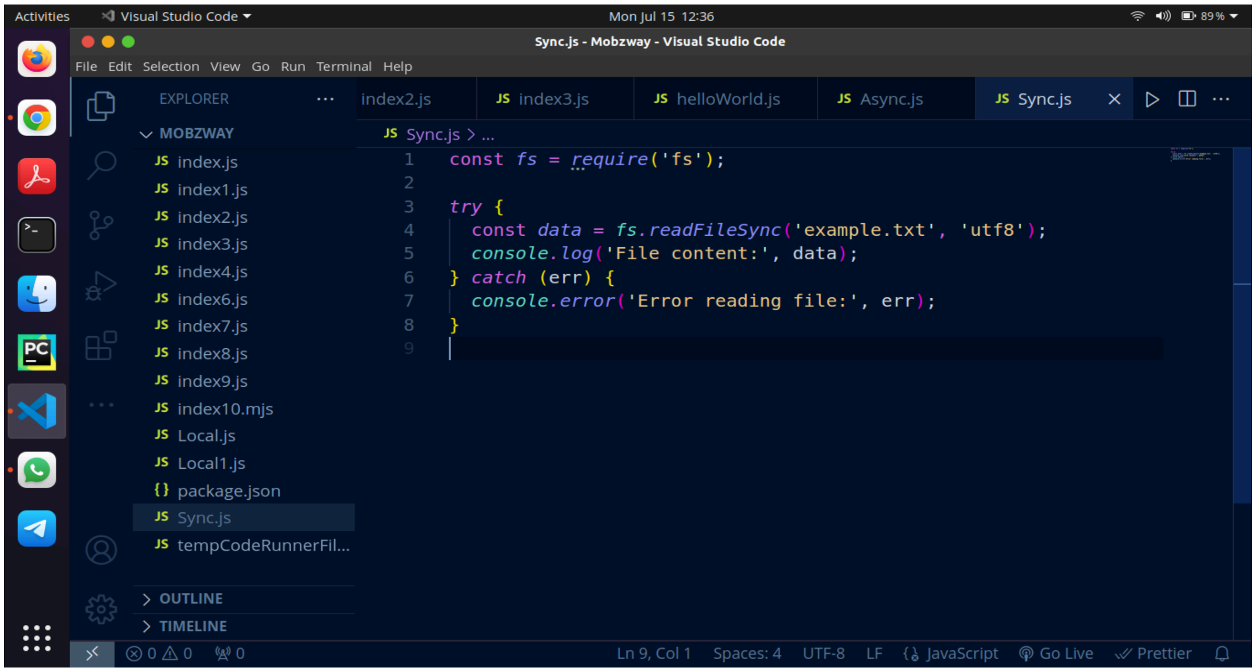
Using the asynchronous method is generally preferred for non-blocking code execution, which is crucial for performance in a Node.js environment.



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

fs.readFile('example.txt', 'utf8', (err, data) => {
  if (err) {
    console.error('Error reading file:', err);
    return;
  }
  console.log('File content:', data);
});
```

Synchronous File Reading

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



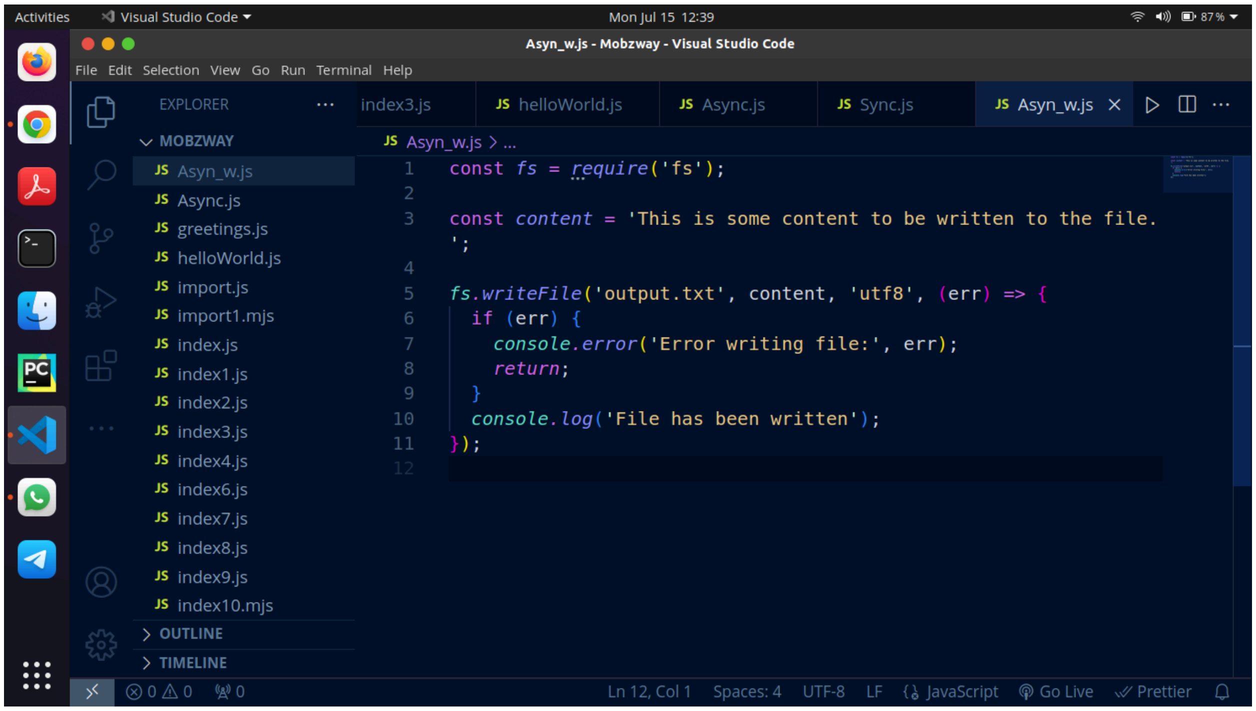
```
const fs = require('fs');

try {
  const data = fs.readFileSync('example.txt', 'utf8');
  console.log('File content:', data);
} catch (err) {
  console.error('Error reading file:', err);
}
```

- **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

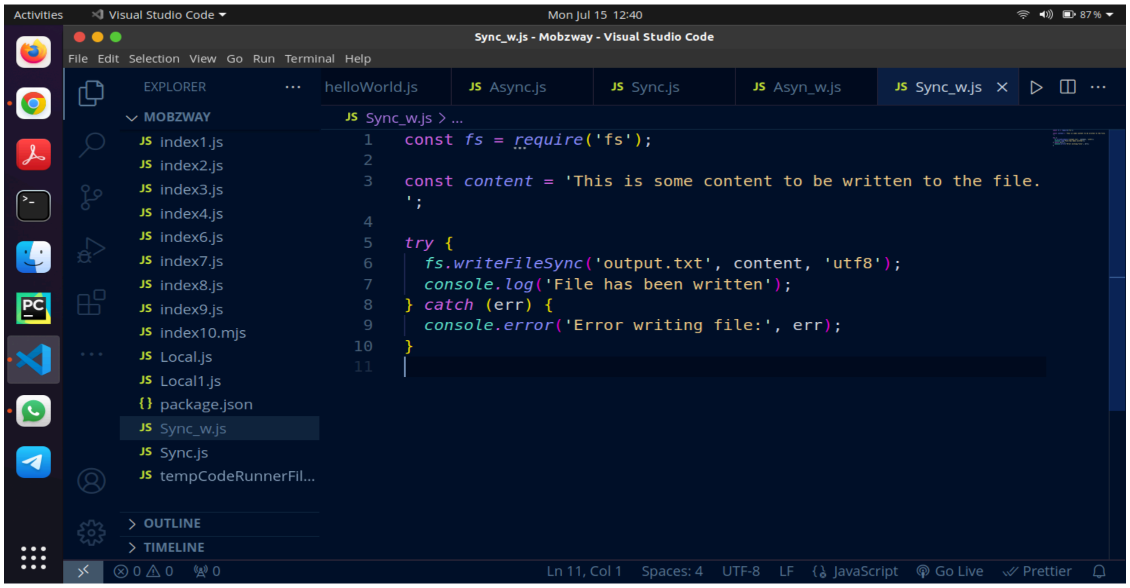
Asynchronous file writing is preferred to avoid blocking the event loop.



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

const content = 'This is some content to be written to the file.
';

fs.writeFile('output.txt', content, 'utf8', (err) => {
  if (err) {
    console.error('Error writing file:', err);
    return;
  }
  console.log('File has been written');
});
```

Synchronous File Writing

Synchronous methods are simpler but can block the event loop.



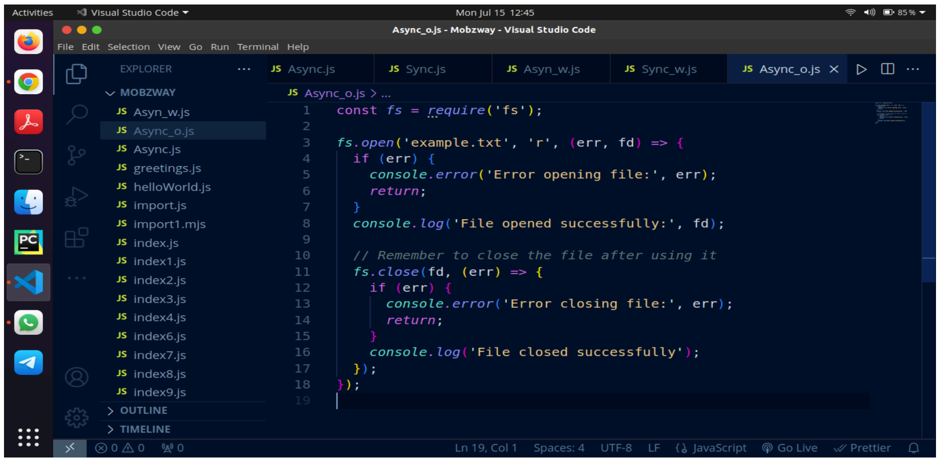
```javascript
const fs = require('fs');

const content = 'This is some content to be written to the file.
';

try {
  fs.writeFileSync('output.txt', content, 'utf8');
  console.log('File has been written');
} catch (err) {
  console.error('Error writing file:', err);
}
```

● 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

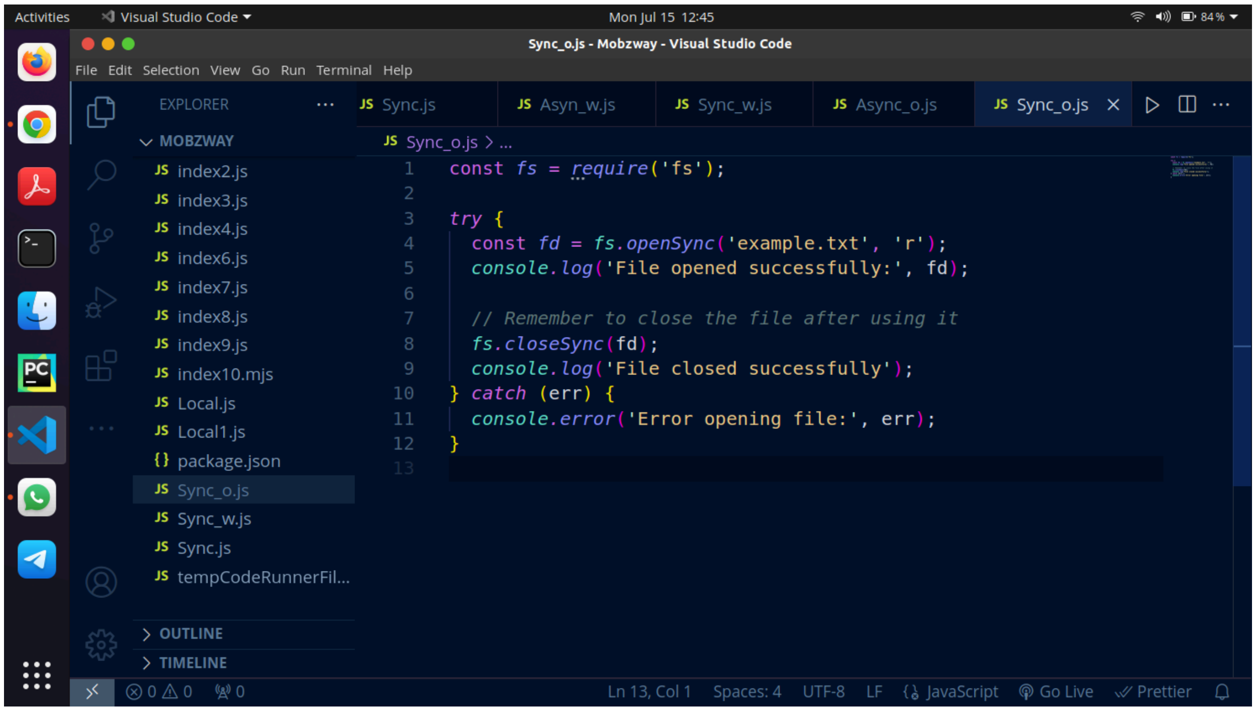
Using the asynchronous method is preferred for non-blocking code execution, which is crucial for performance in a Node.js environment.



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

fs.open('example.txt', 'r', (err, fd) => {
  if (err) {
    console.error('Error opening file:', err);
    return;
  }
  console.log('File opened successfully:', fd);

  // Remember to close the file after using it
  fs.close(fd, (err) => {
    if (err) {
      console.error('Error closing file:', err);
      return;
    }
    console.log('File closed successfully');
  });
});
```

Synchronous File Opening

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



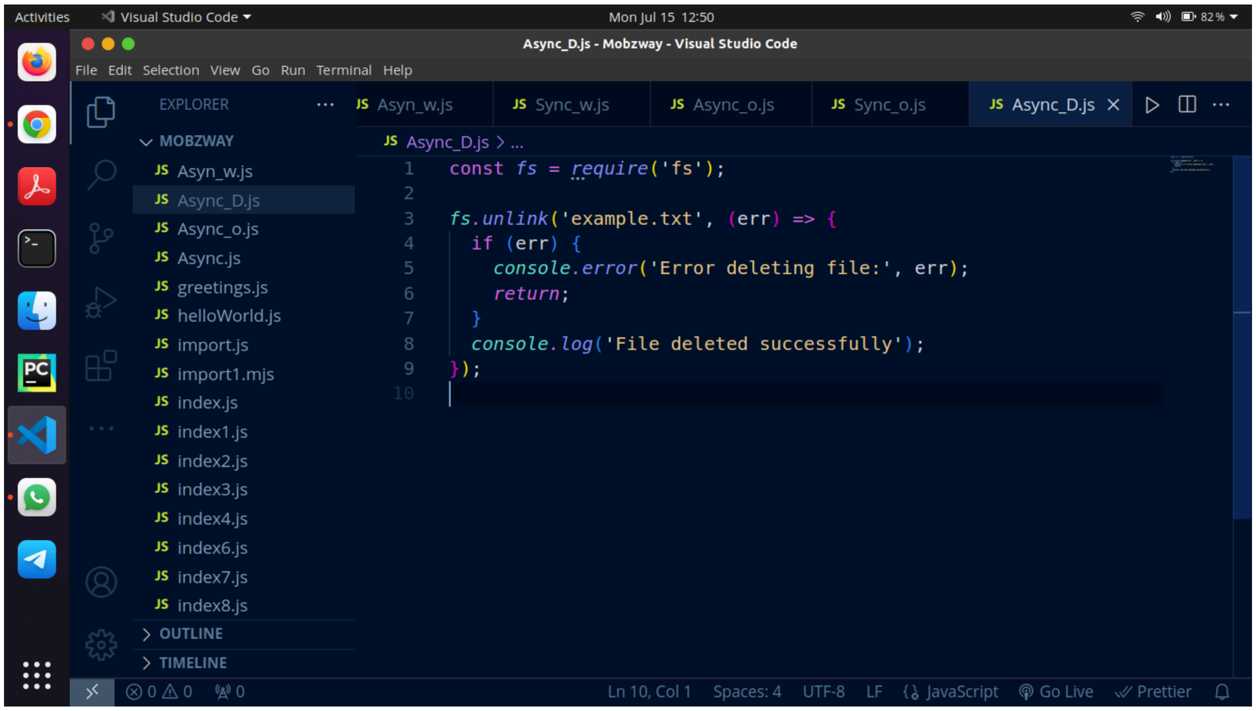
```
const fs = require('fs');

try {
  const fd = fs.openSync('example.txt', 'r');
  console.log('File opened successfully:', fd);

  // Remember to close the file after using it
  fs.closeSync(fd);
  console.log('File closed successfully');
} catch (err) {
  console.error('Error opening file:', err);
}
```

- ## **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

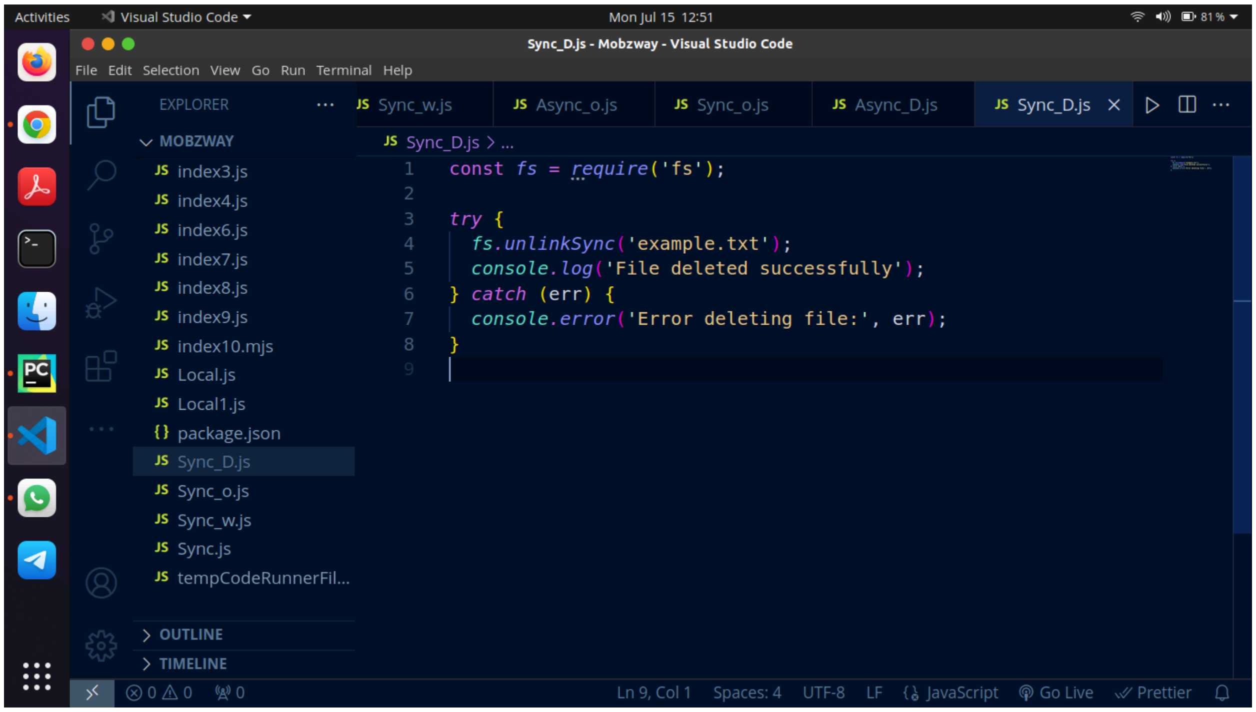
The asynchronous method is preferred to avoid blocking the event loop.



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

fs.unlink('example.txt', (err) => {
  if (err) {
    console.error('Error deleting file:', err);
    return;
  }
  console.log('File deleted successfully');
});
```

Synchronous File Deletion

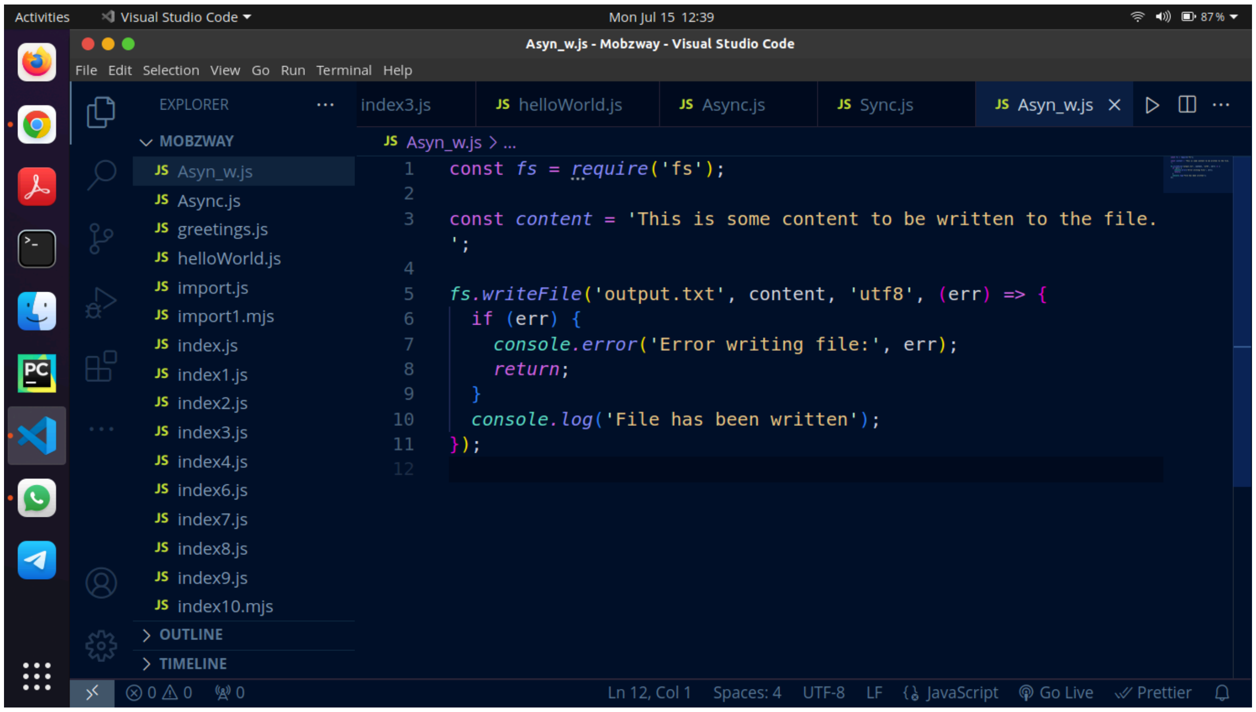
The synchronous method is simpler but can block the event loop.



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

try {
  fs.unlinkSync('example.txt');
  console.log('File deleted successfully');
} catch (err) {
  console.error('Error deleting file:', err);
}
```

- ## Writing a file asynchronously

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



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

const content = 'This is some content to be written to the file.';

fs.writeFile('output.txt', content, 'utf8', (err) => {
  if (err) {
    console.error('Error writing file:', err);
    return;
  }
  console.log('File has been written');
});
```

In this example:

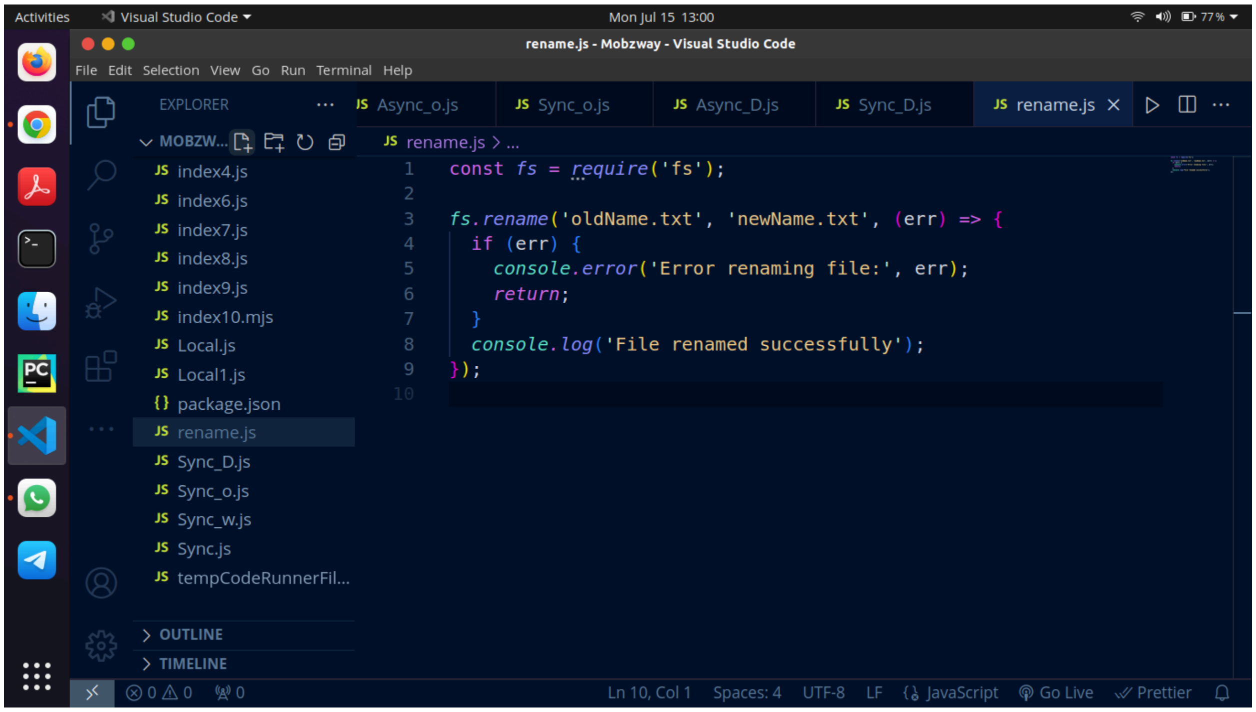
- `fs.writeFile` writes the file asynchronously.
- The first argument is the path to the file.
- The second argument is the content to be written.
- The third argument is the encoding (optional, but `utf8` is commonly used).
- The fourth argument is a callback function that handles any error that occurs.

- ## 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

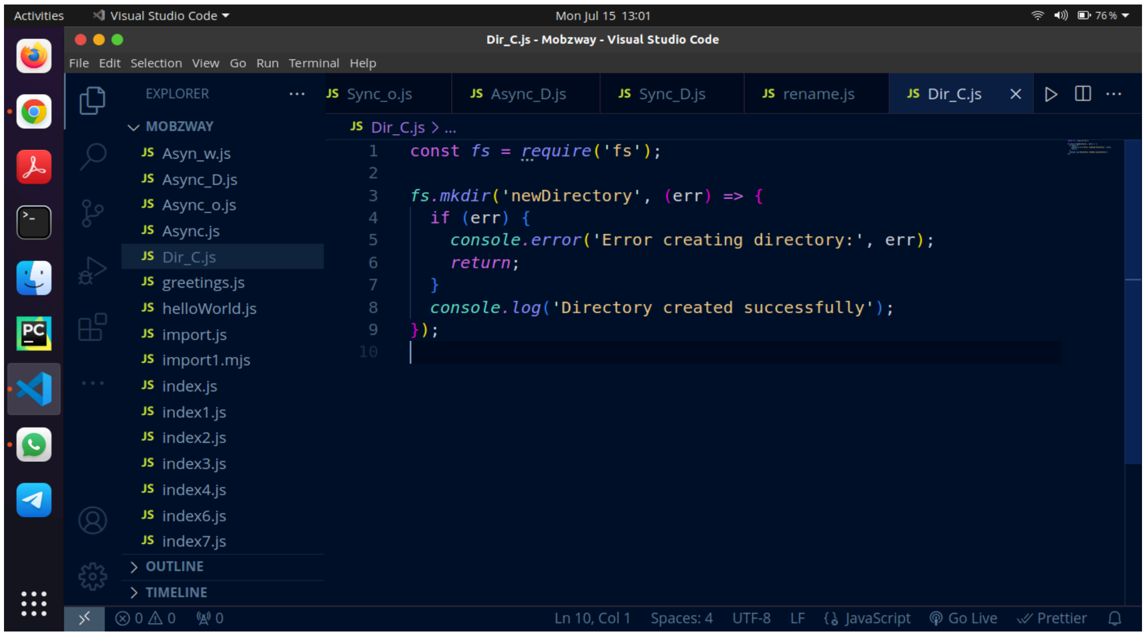
You can rename or move a file using the `fs.rename` method.



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

fs.rename('oldName.txt', 'newName.txt', (err) => {
  if (err) {
    console.error('Error renaming file:', err);
    return;
  }
  console.log('File renamed successfully');
});
```

Creating a Directory

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



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

fs.mkdir('newDirectory', (err) => {
  if (err) {
    console.error('Error creating directory:', err);
    return;
  }
  console.log('Directory created successfully');
});
```