

Aim:- Write a program in Node JS to

- a. Create a file
- b. Read the data from file
- c. Write the data to a file
- d. Rename a file
- e. Append data to a file
- f. Delete a file

LO Mapped:- LO6

Theory:-

Node.js is a powerful runtime environment for executing JavaScript on the server side. It provides built-in modules, including the 'fs' (File System) module, that allow developers to perform various file operations. Here, we will explore how to perform common file operations in Node.js:

a. Creating a File

To create a file in Node.js, you can use the 'fs' (File System) module. You typically use the `fs.writeFile()` method, specifying the file name and content. This method will create the file if it doesn't exist or overwrite its content if it does.

b. Reading Data from a File

Reading data from a file involves using the 'fs' module's `fs.readFile()` method. You provide the file name and an encoding (e.g., 'utf8' for text files) to read the file's content. The method then asynchronously reads the content and provides it as a callback argument.

c. Writing Data to a File

To write data to an existing file, you can use the 'fs' module's `fs.writeFile()` method. Similar to creating a file, you specify the file name and the content you want to write. This method overwrites the file's existing content with the new data.

d. Renaming a File

To rename a file in Node.js, you use the 'fs' module's `fs.rename()` method. This method accepts two arguments: the current file name and the new file name. It effectively renames the file by changing its name in the file system.

e. Appending Data to a File

Appending data to a file without overwriting its existing content can be done using the 'fs' module's `fs.appendFile()` method. You provide the file name and the data to append. This method appends the data to the end of the file, preserving the existing content.

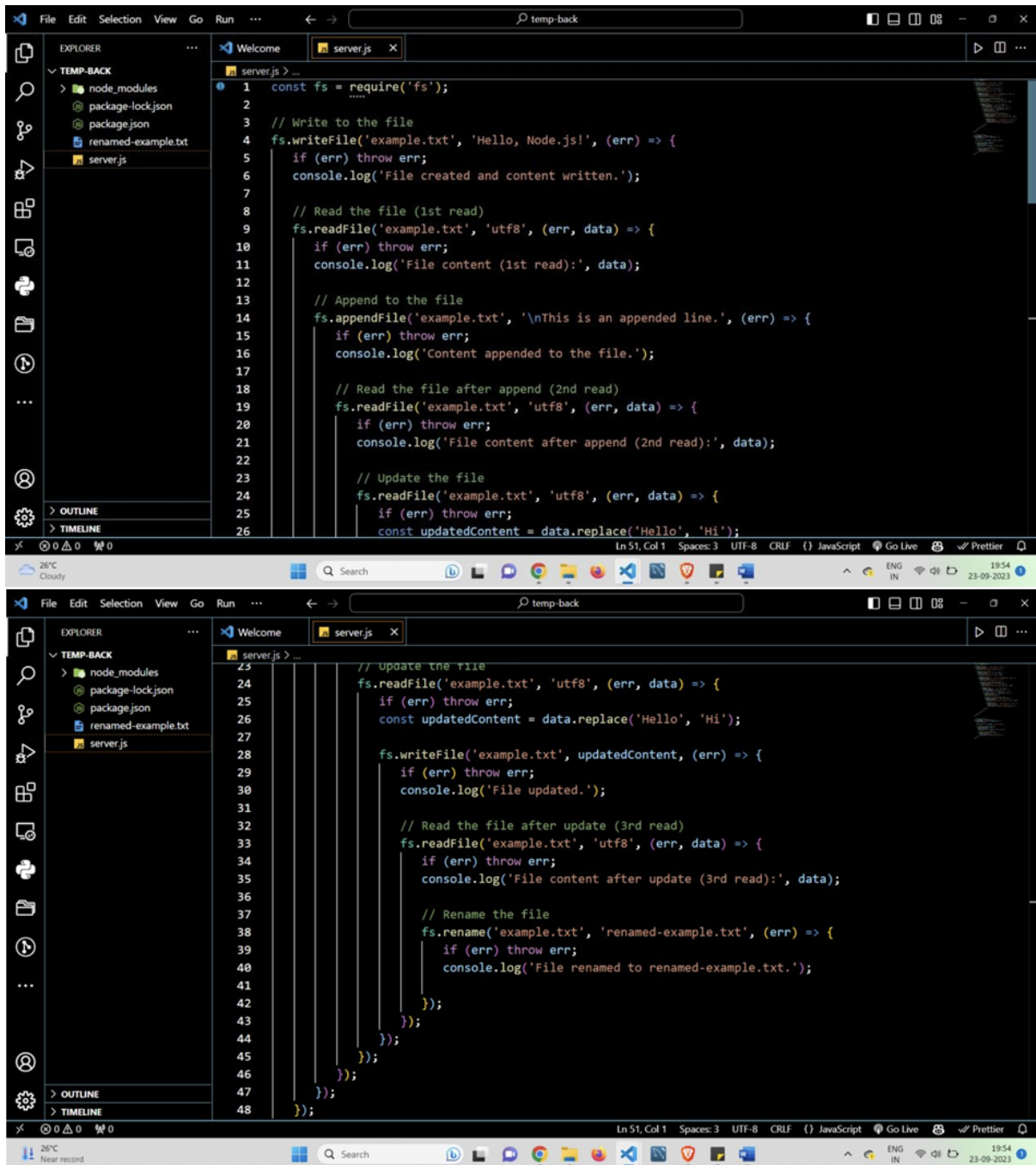
f. Deleting a File

To delete a file, you utilize the 'fs' module's `fs.unlink()` method. You specify the file name as the argument, and this method removes the file from the file system.

In Node.js, these file operations are typically performed asynchronously, allowing your application to continue executing other tasks while file operations are in progress. Proper error

handling is crucial to catch and manage any errors that may occur during these operations, ensuring the robustness and reliability of your Node.js applications.

Code :-



The image displays two screenshots of a Visual Studio Code editor window, showing the implementation of file operations in a project named 'temp-back'. The Explorer sidebar on the left shows the project structure, including 'node_modules', 'package-lock.json', 'package.json', 'renamed-example.txt', and 'server.js'.

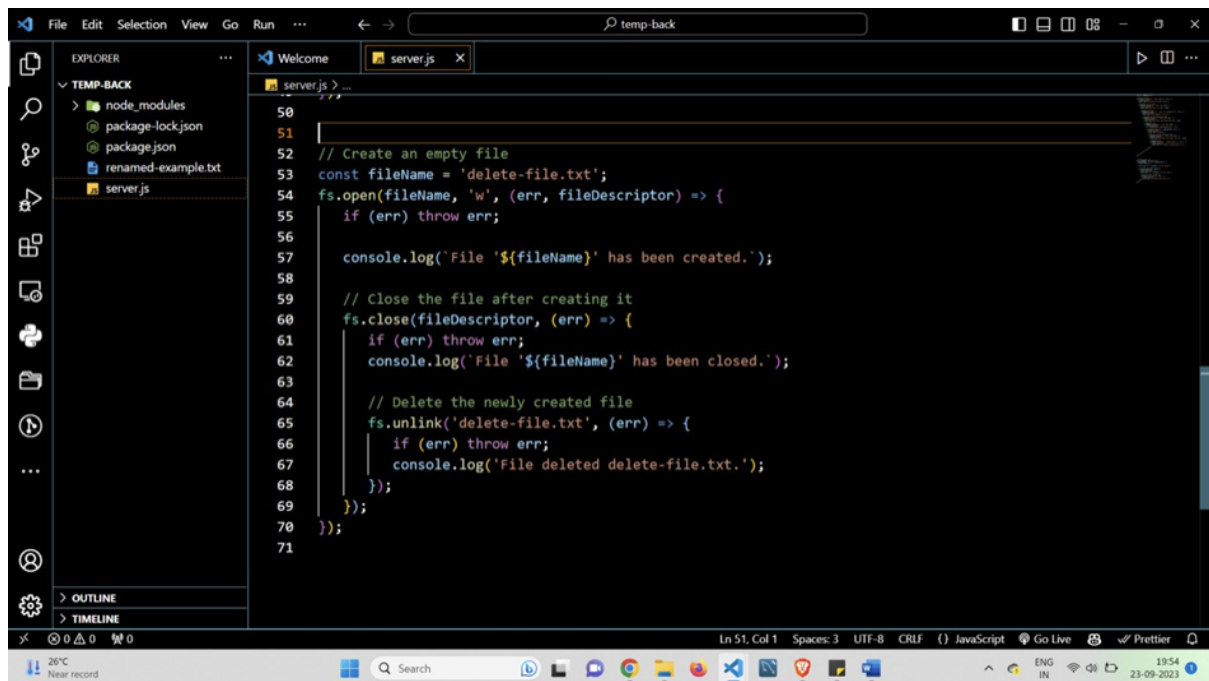
Top Screenshot: The 'server.js' file is open, showing the initial file operations. The code includes:

```
1 const fs = require('fs');
2
3 // Write to the file
4 fs.writeFile('example.txt', 'Hello, Node.js!', (err) => {
5   if (err) throw err;
6   console.log('File created and content written.');
```

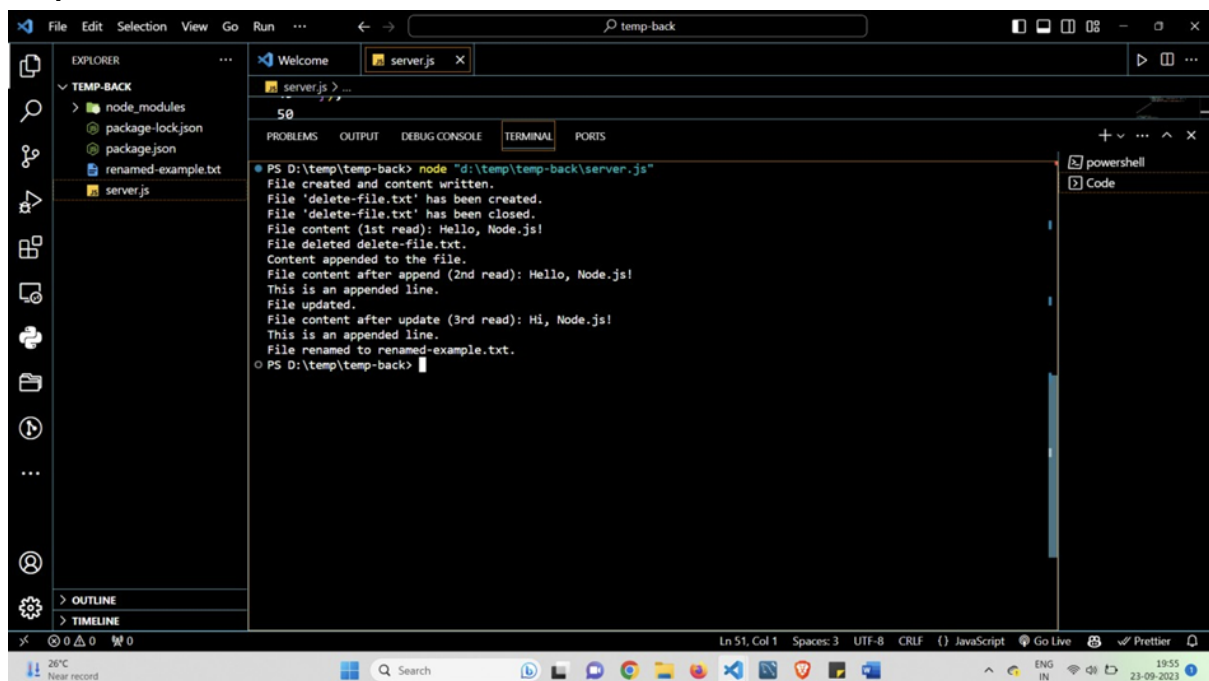
Bottom Screenshot: The 'server.js' file is open, showing the continuation of file operations. The code includes:

```
23 // update the file
24 fs.readFile('example.txt', 'utf8', (err, data) => {
25   if (err) throw err;
26   const updatedContent = data.replace('Hello', 'Hi');
```

The status bar at the bottom of the editor indicates the current file is 'server.js', the encoding is 'UTF-8', and the line/character count is 'Ln 51, Col 1'.



Output :-



Conclusion :-

Learnt about basic CRUD operations in node.js and also learnt about different ways in which these can be implemented , practically demonstrated CRUD operations using node.js