Task-1

Aim:

Setting up a basic HTTP server: Create a Node.js application that listens for incoming HTTP requests and responds with a simple message.

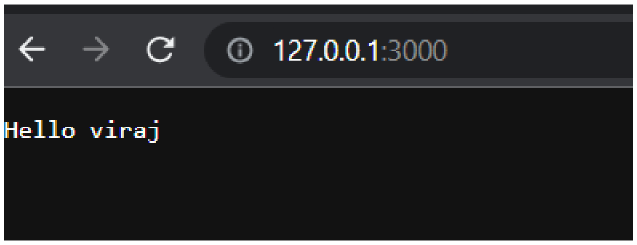
Description:

- Initialize a new Node.js project: Begin by setting up a new Node.js project. Open your terminal and navigate to the desired directory. Run the command npm init and follow the prompts to create a package.json file, which will track your project's dependencies and configuration.

Source Code:

```
const http = require("http");
const httpserver = http.createServer(function (req, res) {
    if (req.method == 'GET') {
        res.end("viraj pankhaniya");
    }
});
httpserver.listen(3000, () => {
    console.log("Listning on port 3000...");
})
```

Output:



127.0.0.1:3000

```
Hello viraj
```

Task-2

Aim: Experiment with Various HTTP Methods,Content Types and Status Code

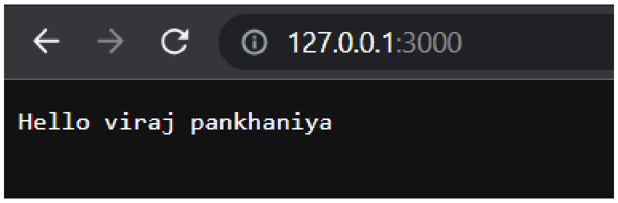
Description:

- To run this snippet, save it as a server.js file and run node server.js in your terminal. ○ This code first includes the Node.js http module.
- Node.js has a fantastic standard library, including first-class support for networking.
- The createServer () method of http creates a new HTTP server and returns it.
- The server is set to listen on the specified port and host name. When the server is ready, the callback function is called, in this case informing us that the server is running.

Source Code:

```
const http = require('http');
const hostname = '127.0.0.1';
const port = 3000;
const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Hello viraj pankhaniya \n');
});
server.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname}:${port}/`);
});
```

Output:



```
Hello viraj pankhaniya
```

Task-3

Aim: Test it using browser ,CLI and REST Client

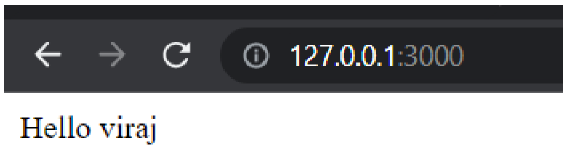
Description:

1. **Testing with a web browser:** ○ Start your Node.js server by running the command node server.js in the terminal. ○ Open a web browser and navigate to http://localhost:<port>, where <port> is the port number specified in your Node.js server code.
2. The browser will send an HTTP GET request to the server, and you should see the response message "Hello, world!" displayed on the page.

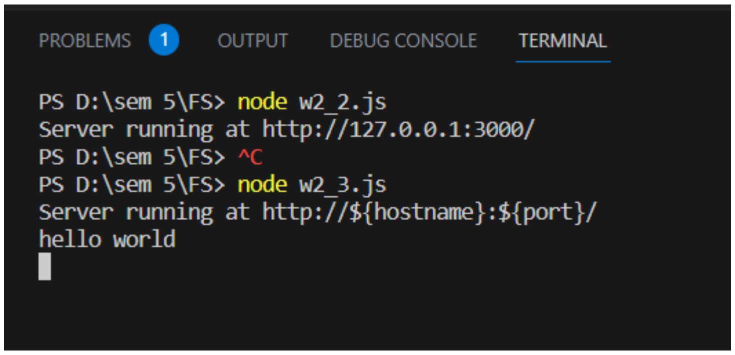
Source Code:

```
const http = require('http');
const hostname = '127.0.0.1';
const port = 3000;
const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/html');
    res.end('Hello viraj\n');
});
server.listen(port, hostname, () => {
    console.log("Server running at http://${hostname}:${port}/");
    console.log("hello world");
});
```

Output:



Hello viraj



```
PS D:\sem 5\FS> node w2_2.js
Server running at http://127.0.0.1:3000/
PS D:\sem 5\FS> ^C
PS D:\sem 5\FS> node w2_3.js
Server running at http://${hostname}:${port}/
hello world
```

Task-4

Aim: Read File student-data.txt file and find all students whose name contains ‘MA’ and CGPA > 7.

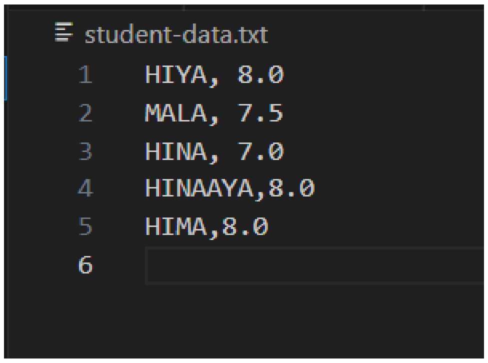
Description:

- Next, you can initialize an empty array, let's call it **matchingStudents**, to store the names of the students that meet the criteria. You will iterate over each line using a loop, and for each line, you will split it further using a separator like a comma (','). This will give you an array of values representing different attributes of a student, such as name and CGPA.
- You can then check if the name contains 'MA' by using the **includes()** method of the name string. Additionally, you can convert the CGPA value to a number using **parseFloat()** and compare it to 7 to check if it is greater.
- If the conditions are met, you can add the student's name to the **matchingStudents** array.

Source Code:

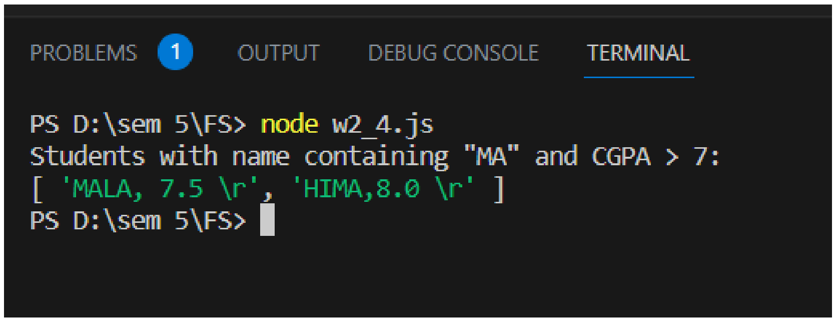
```javascript
const fs = require('fs');
fs.readFile('student-data.txt', 'utf8', (err, data) => {
    if (err) {
        console.error('Error reading the file:', err);
        return;
    }
    const students = data.split('\n');
    const filteredStudents = students.filter((student) => {
        const [name, cgpa] = student.split(',');
        return name.includes('MA') && parseFloat(cgpa) > 7;
    });
    console.log('Students with name containing "MA" and CGPA > 7:');
    console.log(filteredStudents);
})
```

Output:



student-data.txt

```
HIYA, 8.0
MALA, 7.5
HINA, 7.0
HINAAYA,8.0
HIMA,8.0

```



```
PS D:\sem 5\FS> node w2_4.js
Students with name containing "MA" and CGPA > 7:
[ 'MALA, 7.5 \r', 'HIMA,8.0 \r' ]
PS D:\sem 5\FS>
```

Task-5

Aim: Read Employee Information from User and Write Data to file called ‘employee-data.json’

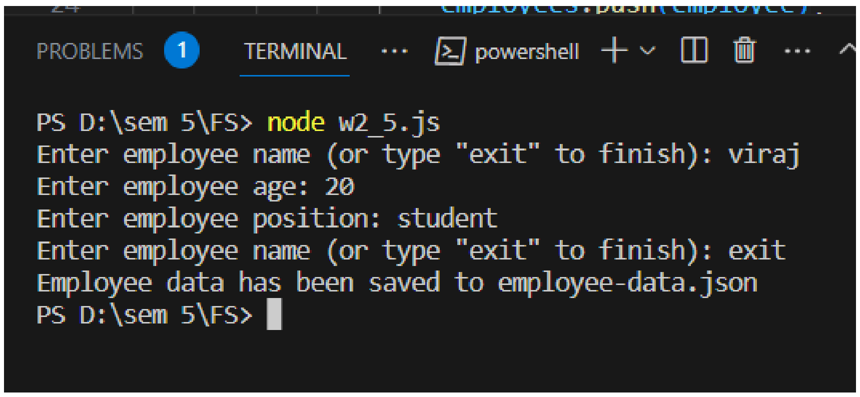
Description:

1. Set up a user interface for reading input from the user. This can be done through a command-line interface, a web form, or any other means suitable for your application.
2. Create an empty array to store the employee data. This array will hold objects, where each object represents an employee and contains their information.
3. Prompt the user to enter the employee's information. This typically includes fields such as name, age, position, salary, etc. Collect this information and create an employee object using the entered values.

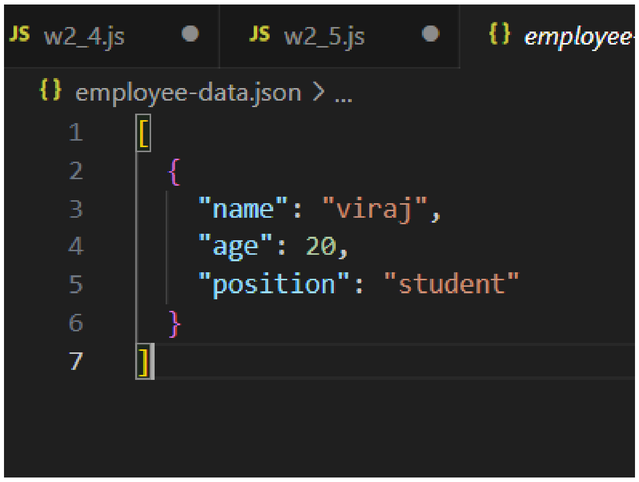
Source Code:

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

// Create an interface for reading user input
const rl = readline.createInterface({
    input: process.stdin, output: process.stdout
});

const employees = [];

function promptForEmployeeInfo() {
    rl.question('Enter employee name (or type "exit" to finish): ', (name) => {
        if (name.toLowerCase() === 'exit') {

            writeEmployeeDataToFile();
        } else {
            rl.question('Enter employee age: ', (age) => {
                rl.question('Enter employee position: ', (position) => {

                    const employee = {
                        name: name, age: parseInt(age),
                        position: position
                    };

                    employees.push(employee);

                    promptForEmployeeInfo();
                });
            });
        }
    });
}

function writeEmployeeDataToFile() {
    const data = JSON.stringify(employees, null, 2);

    fs.writeFile('employee-data.json', data, (err) => {
        if (err) {
            console.error('Error writing to file:', err);
        } else {
            console.log('Employee data has been saved to employee-data.json');
        }
        rl.close();
    });
}

promptForEmployeeInfo();
```

Output:



```
PS D:\sem 5\FS> node w2_5.js
Enter employee name (or type "exit" to finish): viraj
Enter employee age: 20
Enter employee position: student
Enter employee name (or type "exit" to finish): exit
Employee data has been saved to employee-data.json
PS D:\sem 5\FS>
```



```
[
  {
    "name": "viraj",
    "age": 20,
    "position": "student"
  }
]
```

Task-6

Aim: Compare Two file and show which file is larger and which lines are different

Description:

1. Read the contents of the two files you want to compare. You can use file I/O operations to read the contents into separate variables or buffers.
2. Calculate the sizes of the files. This can be done by measuring the byte length of the file contents. Compare the file sizes to determine which file is larger.
3. Split the contents of each file into lines. You can split the file contents by newline characters to create arrays of lines for each file.
4. Iterate over the lines of the files simultaneously. Compare the corresponding lines between the two files, line by line.
5. Identify the lines that are different. Whenever a mismatch is found between the lines of the two files, store the line number and the contents of the mismatched lines.
6. Repeat the comparison process until all lines of the files have been compared.
7. Display the results. Output the information about which file is larger and display the line numbers and contents of the different lines.

Source Code:

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

function compareFiles(FILE1Path, FILE2Path) {   const FILE1Lines =
fs.readFileSync(FILE1Path, 'utf8').split('\n');   const FILE2Lines =
fs.readFileSync(FILE2Path, 'utf8').split('\n');

  if (FILE1Lines.length > FILE2Lines.length) {
    console.log(`${FILE1Path} is larger than ${FILE2Path}`);
  } else if (FILE2Lines.length > FILE1Lines.length)
{     console.log(`${FILE2Path} is larger than ${FILE1Path}`);
  } else {
    console.log('Both files have the same number of lines.');   }

  console.log('Differing lines:');
  for (let i = 0; i < FILE1Lines.length && i < FILE2Lines.length; i++) {     if
(FILE1Lines[i] !== FILE2Lines[i]) {
      console.log(`Line ${i + 1}:`);
      console.log(`${FILE1Path}:
${FILE1Lines[i].trim()}`);       console.log(`${FILE2Path}:
${FILE2Lines[i].trim()}`);       console.log();
    }
  }

  if (FILE1Lines.length !== FILE2Lines.length) {
    const additionalLines = FILE1Lines.length > FILE2Lines.length ? FILE1Lines :
FILE2Lines;      const FILE = FILE1Lines.length > FILE2Lines.length ? FILE1Path :
FILE2Path;     console.log('Additional lines:');
    for (let i = Math.min(FILE1Lines.length, FILE2Lines.length); i <
additionalLines.length; i++) {       console.log(`Line ${i + 1}:`);
      console.log(`${FILE}: ${additionalLines[i].trim()}`);       console.log();
    }
  }
}
```

Output:

Task-7

Aim: Create File Backup and Restore Utility

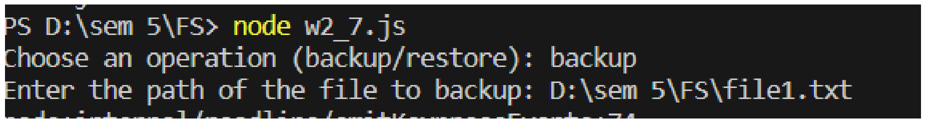
Description:

1. Import the necessary modules for file operations. In Node.js, you can use the **fs** module for file-related operations and the **path** module for handling file paths.
2. Set up a user interface for interacting with the utility. This can be done through a command-line interface, a web form, or any other suitable means for your application.
3. Implement a function to create a backup of a file. This function should take the file path as an input and generate a backup file with a unique name, such as appending a timestamp or a version number to the original file name. Use the **fs.copyFile()** function to copy the original file to the backup file.

Source Code:

```
const path = require('path');
const readline = require('readline');
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});
// Function to create a backup of a file
function createBackup(filePath) {
    // Generate a timestamp for the backup file
    const timestamp = new Date().toISOString().replace(/:/g, '');
    // Generate the backup file name
    const backupFileName = path.basename(filePath) + '-' + timestamp + '.bak';
    const backupFilePath = path.join(path.dirname(filePath), backupFileName);
    // Copy the original file to the backup file
    fs.copyFile(filePath, backupFilePath, (err) => {
        if (err) {
            console.error('Error creating backup:', err);
        } else {
            console.log(`Backup created: ${backupFilePath}`);
        }
        rl.close();
    });
}
// Function to restore a file from backup
function restoreBackup(backupFilePath, originalFilePath) {
    fs.copyFile(backupFilePath, originalFilePath, (err) => {
        if (err) {
            console.error('Error restoring backup:', err);
        } else {
            console.log(`Backup restored: ${originalFilePath}`);
        }
        rl.close();
    });
}
// Prompt the user for backup or restore operation
rl.question('Choose an operation (backup/restore): ', (operation) => {
    if (operation === 'backup') {
        rl.question('Enter the path of the file to backup: ', (filePath) => {
            createBackup(filePath);
        });
    } else if (operation === 'restore') {
        rl.question('Enter the path of the backup file: ', (backupFilePath) => {
            rl.question('Enter the path to restore the file: ',
(originalFilePath) => {
                restoreBackup(backupFilePath, originalFilePath);
            });
        });
    } else {
        console.log('Invalid operation.');
        rl.close();
    }
});
```

Output:



```
PS D:\sem 5\FS> node w2_7.js
Choose an operation (backup/restore): backup
Enter the path of the file to backup: D:\sem 5\FS\file1.txt
```

Task-8

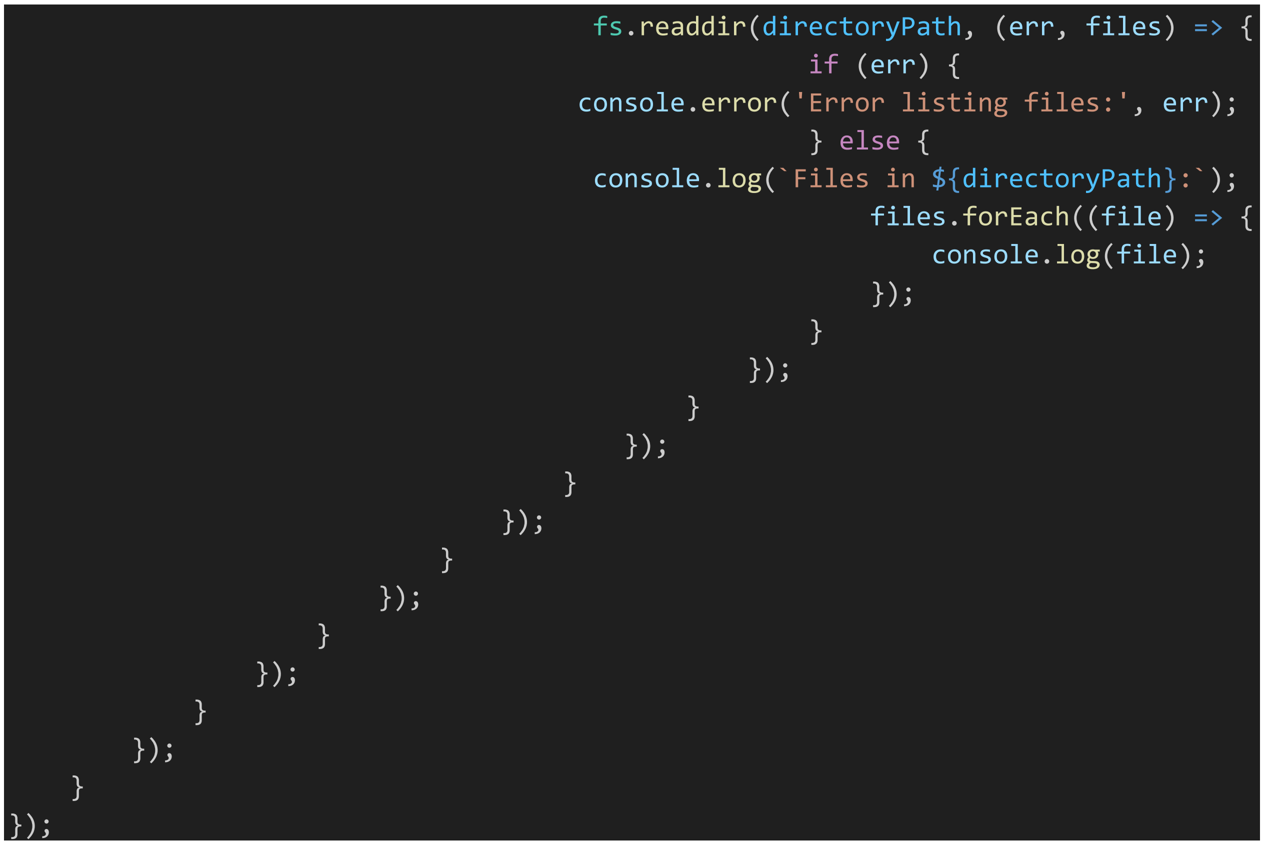
Aim: Create File/Folder Structure given in json file.

Description:

1. Read the JSON file that contains the desired file/folder structure. The JSON file should have an array of objects, where each object represents a file or folder in the desired structure. Each object should have a **name** property to specify the name of the file or folder and a **type** property to indicate whether it's a file or folder. For folders, there may be a nested **children** property containing the child files and folders.
2. Specify the root directory where you want to create the file/folder structure.

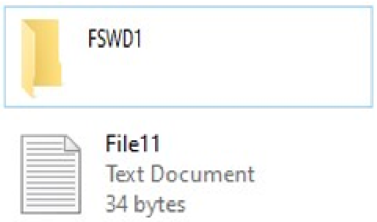
Source Code:

```
const fs = require('fs');
const path = require('path');
// Create a file
fs.writeFile('example.txt', 'This is a sample file.', (err) => {
    if (err) {
        console.error('Error creating file:', err);
    } else {
        console.log('File created successfully.');
        // Read the file
        fs.readFile('example.txt', 'utf-8', (err, data) => {
            if (err) {
                console.error('Error reading file:', err);
            } else {
                console.log('File content:', data);
                // Append content to the file
           fs.appendFile('example.txt', '\nThis is appended content.', (err) => {
                    if (err) {
                        console.error('Error appending to file:', err);
                    } else {
                        console.log('Content appended successfully.');
                        // Read the file again to see the appended content
                        fs.readFile('example.txt', 'utf-8', (err, data) => {
                            if (err) {
                                console.error('Error reading file:', err);
                            } else {
                                console.log('Updated file content:', data);
                                // Rename a file
                           fs.rename('example.txt', 'new-example.txt', (err) => {
                                    if (err) {
                                     console.error('Error renaming file:', err);
                                    } else {
                                      console.log('File renamed successfully.');
                                        // Delete the file
                                        fs.unlink('new-example.txt', (err) => {
                                            if (err) {
                                      console.error('Error deleting file:', err);
                                            } else {
                             console.log('File deleted successfully.');
                                        // List files/directories in a directory
                                                const directoryPath = '.';
```



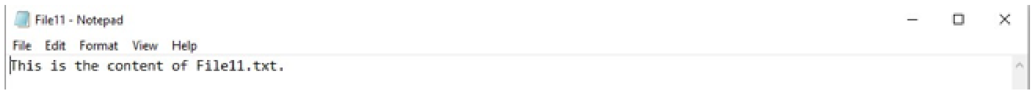
```
                                      fs.readdir(directoryPath, (err, files) => {
                                                    if (err) {
                                     console.error('Error listing files:', err);
                                                    } else {
                                      console.log(`Files in ${directoryPath}:`);
                                                        files.forEach((file) => {
                                                            console.log(file);
                                                        });
                                                    }
                                                });
                                            }
                                        });
                                    }
                                });
                            }
                        });
                    }
                });
            }
        });
    }
});
```

Output:



FSWD1

File11
Text Document
34 bytes



```
This is the content of File11.txt.
```

Task-9

Aim: Experiment with : Create File,Read File,Append File,Delete File,Rename File,List Files/Dirs

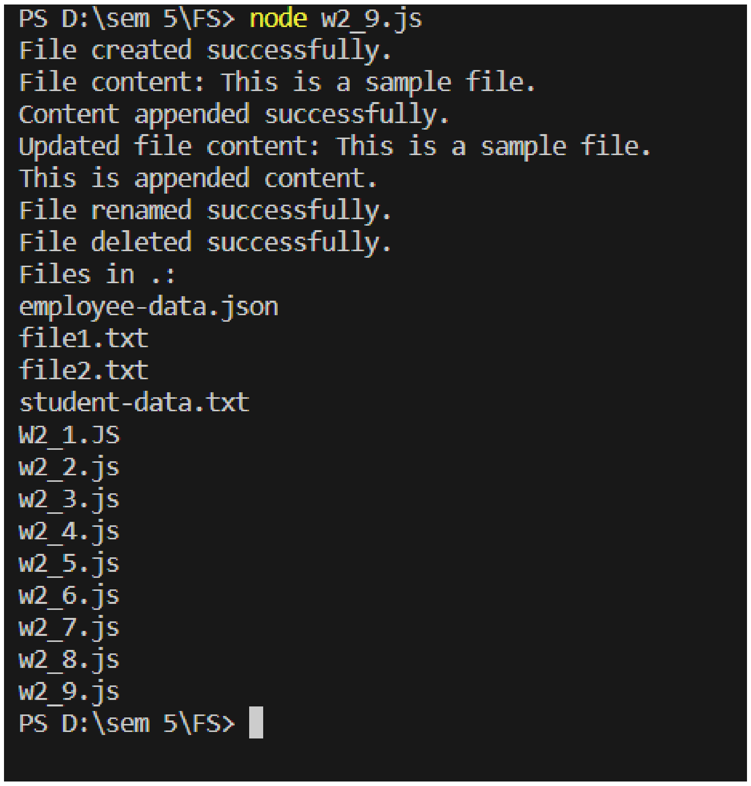
Description:

- **fs.writeFile()** creates a file with the provided content.
- **fs.readFile()** reads the contents of a file.
- **fs.appendFile()** appends content to an existing file.
- **fs.unlink()** deletes a file.
- **fs.rename()** renames a file.
- **fs.readdir()** lists the files and directories in a given directory.

Source Code:

```
const fs = require('fs');
const path = require('path');
// Create a file
fs.writeFile('example.txt', 'This is a sample file.', (err) => {
    if (err) {
        console.error('Error creating file:', err);
    } else {
        console.log('File created successfully.');
        // Read the file
        fs.readFile('example.txt', 'utf-8', (err, data) => {
            if (err) {
                console.error('Error reading file:', err);
            } else {
                console.log('File content:', data);
                // Append content to the file
     fs.appendFile('example.txt', '\nThis is appended content.', (err) => {
                    if (err) {
                        console.error('Error appending to file:', err);
                    } else {
                        console.log('Content appended successfully.');
                        // Read the file again to see the appended content
                        fs.readFile('example.txt', 'utf-8', (err, data) => {
                            if (err) {
                                console.error('Error reading file:', err);
                            } else {
                                console.log('Updated file content:', data);
                                // Rename a file
                          fs.rename('example.txt', 'new-example.txt', (err) => {
                                    if (err) {
                             console.error('Error renaming file:', err);
                                    } else {
                            console.log('File renamed successfully.');
                                        // Delete the file
                                  fs.unlink('new-example.txt', (err) => {
                                   if (err) {
                                     console.error('Error deleting file:', err);
                                            } else {
                                     console.log('File deleted successfully.');
                                      // List files/directories in a directory
                                                const directoryPath = '.';
                                     fs.readdir(directoryPath, (err, files) => {
                                         if (err) {
                                    console.error('Error listing files:', err);
                                                    } else {
                               console.log(`Files in ${directoryPath}:`);
                                           files.forEach((file) => {
                                                            console.log(file);
                                                        });
                                                    }
                                                });
                                            }
                                        });
                                    }
                                });
                            }
                        });
                    }
                });
            }
        });
    }
});
```

Output:



```
PS D:\sem 5\FS> node w2_9.js
File created successfully.
File content: This is a sample file.
Content appended successfully.
Updated file content: This is a sample file.
This is appended content.
File renamed successfully.
File deleted successfully.
Files in .:
employee-data.json
file1.txt
file2.txt
student-data.txt
W2_1.JS
w2_2.js
w2_3.js
w2_4.js
w2_5.js
w2_6.js
w2_7.js
w2_8.js
w2_9.js
PS D:\sem 5\FS>
```

Learning Outcome: Demonstrate the use of JavaScript to fulfill the essentials of front-end development To back-end development.