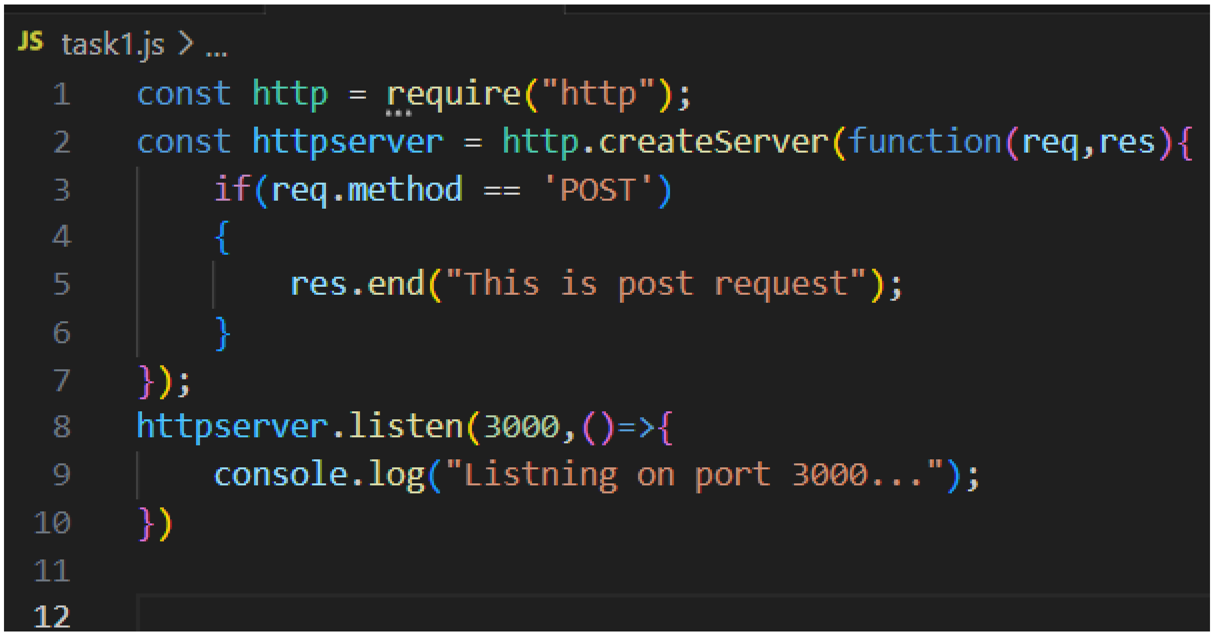
Week 2

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.

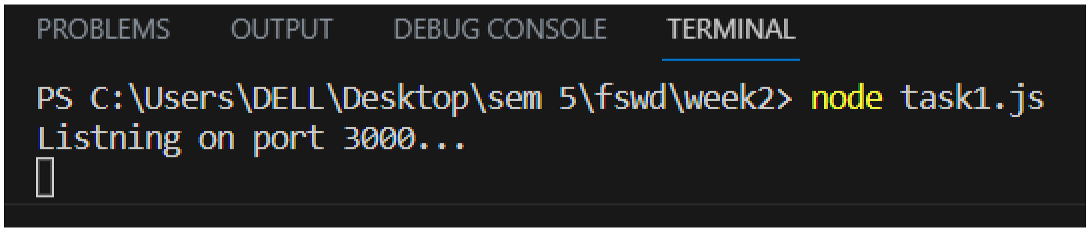
Theoretical Background:
An HTTP server is a software application that listens for incoming HTTP requests from clients and responds to those requests with appropriate HTTP responses. It acts as a communication bridge between clients (such as web browsers) and servers.

Source Code:



```
JS task1.js > ...
const http = require("http");
const httpserver = http.createServer(function(req,res){
    if(req.method == 'POST')
    {
        res.end("This is post request");
    }
});
httpserver.listen(3000,()=>{
    console.log("Listning on port 3000...");
})
```

Output:



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\DELL\Desktop\sem 5\fswd\week2> node task1.js
Listning on port 3000...
```

Task 2

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

Theoretical Background:
HTTP defines a set of request methods to indicate the desired action to be performed for a given resource. Although they can also be nouns, these request methods are sometimes referred to as HTTP verbs. Each of them implements a different semantic, but some common features are shared by a group of them: e.g. a request method can be safe, idempotent, or cacheable.

GET
The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.

HEAD
The HEAD method asks for a response identical to a GET request, but without the response body.

POST
The POST method submits an entity to the specified resource, often causing a change in state or side effects on the server.

PUT
The PUT method replaces all current representations of the target resource with the request payload.

DELETE
The DELETE method deletes the specified resource.

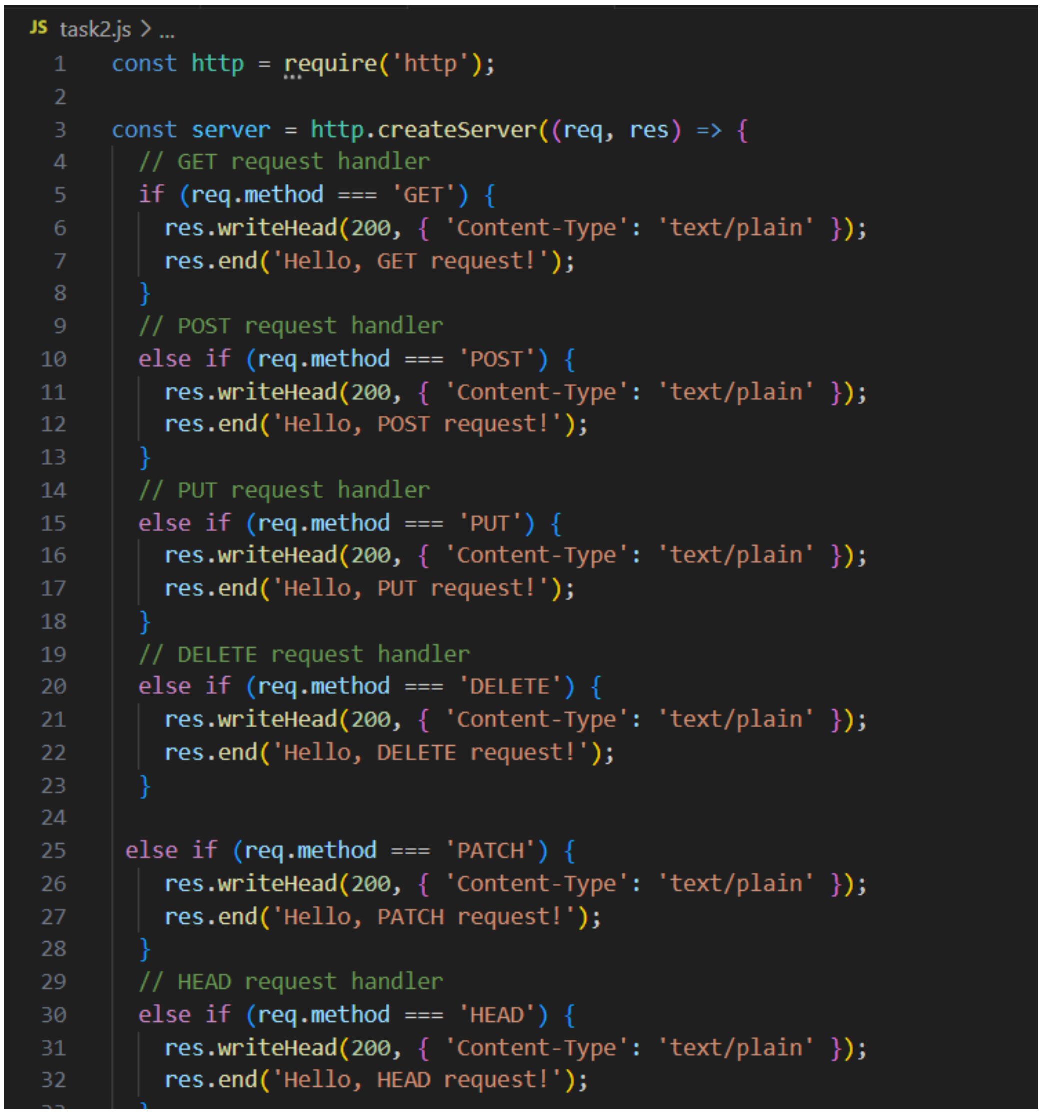
CONNECT
The CONNECT method establishes a tunnel to the server identified by the target resource.

OPTIONS
The OPTIONS method describes the communication options for the target resource.

TRACE
The TRACE method performs a message loop-back test along the path to the target resource.

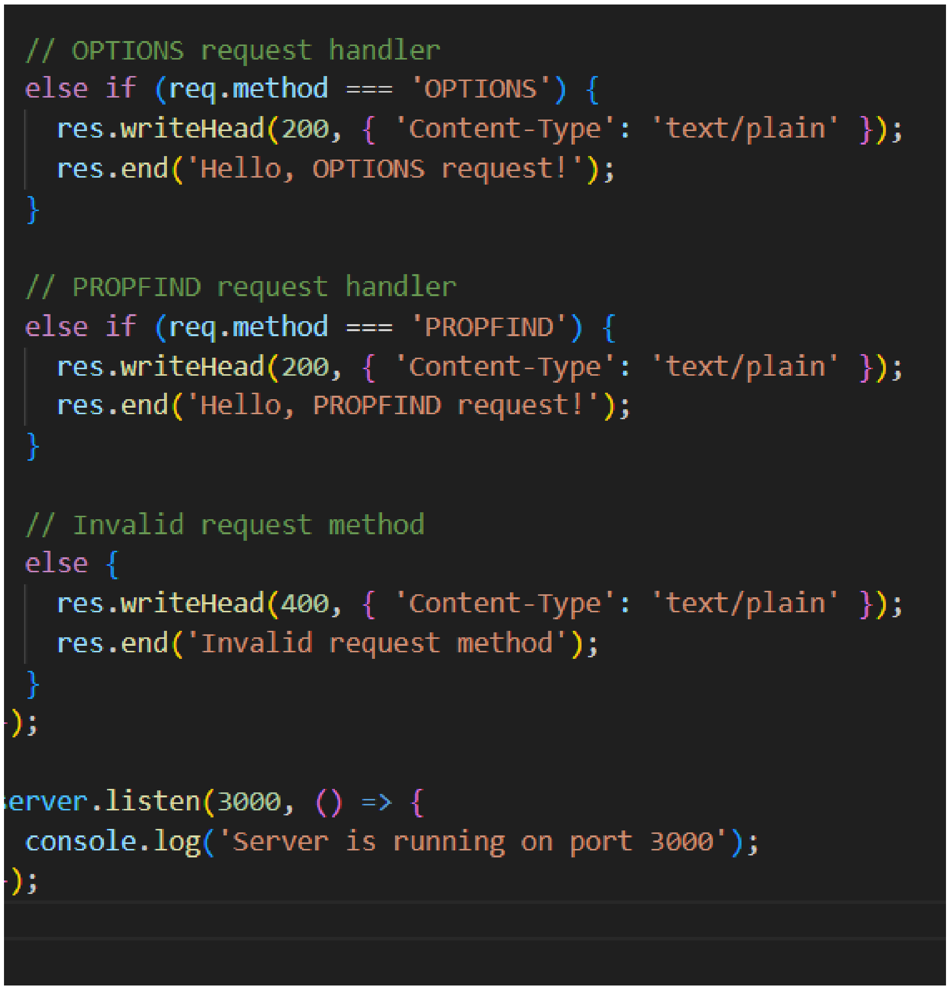
PATCH
The PATCH method applies partial modifications to a resource.

Source Code:



task2.js > ...

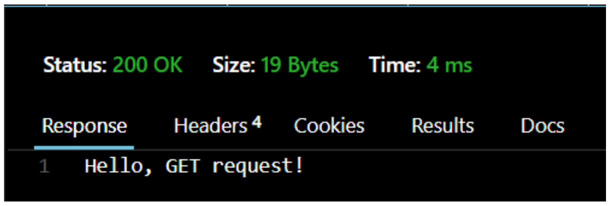
```
const http = require('http');

const server = http.createServer((req, res) => {
  // GET request handler
  if (req.method === 'GET') {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello, GET request!');
  }
  // POST request handler
  else if (req.method === 'POST') {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello, POST request!');
  }
  // PUT request handler
  else if (req.method === 'PUT') {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello, PUT request!');
  }
  // DELETE request handler
  else if (req.method === 'DELETE') {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello, DELETE request!');
  }

 else if (req.method === 'PATCH') {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello, PATCH request!');
  }
  // HEAD request handler
  else if (req.method === 'HEAD') {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello, HEAD request!');
```



```
  // OPTIONS request handler
  else if (req.method === 'OPTIONS') {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello, OPTIONS request!');
  }

  // PROPFIND request handler
  else if (req.method === 'PROPFIND') {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello, PROPFIND request!');
  }

  // Invalid request method
  else {
    res.writeHead(400, { 'Content-Type': 'text/plain' });
    res.end('Invalid request method');
  }
});

erver.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

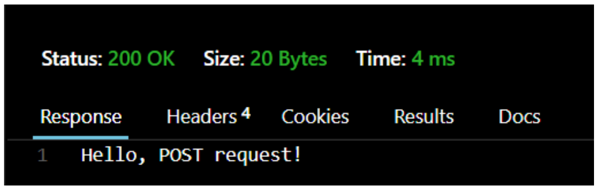
Output:

GET method:



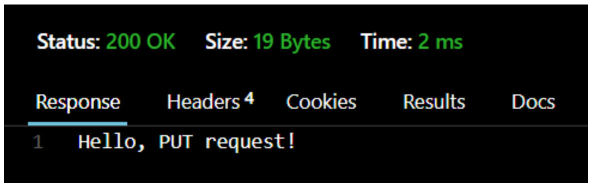
```
Status: 200 OK   Size: 19 Bytes   Time: 4 ms
Hello, GET request!
```

POST method:



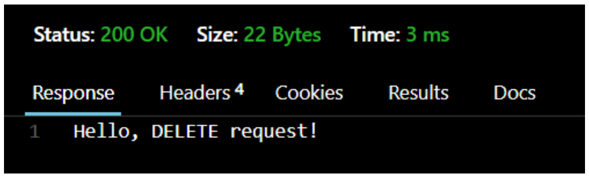
```
Status: 200 OK   Size: 20 Bytes   Time: 4 ms
Hello, POST request!
```

PUT method:



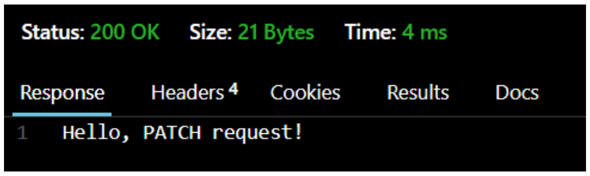
```
Status: 200 OK   Size: 19 Bytes   Time: 2 ms
Hello, PUT request!
```

DELETE method:



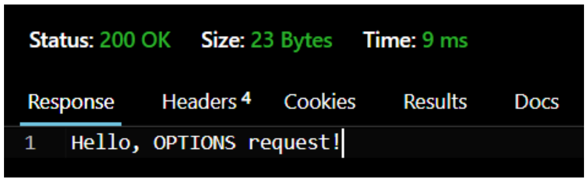
```
Status: 200 OK   Size: 22 Bytes   Time: 3 ms
Hello, DELETE request!
```

PATCH method:



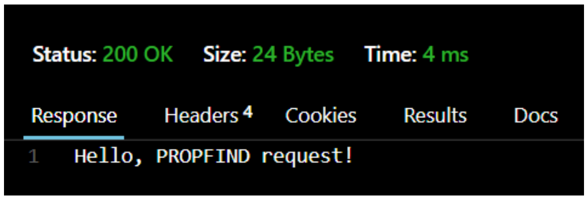
```
Status: 200 OK   Size: 21 Bytes   Time: 4 ms
Hello, PATCH request!
```

OPTION method:



```
Status: 200 OK   Size: 23 Bytes   Time: 9 ms
Hello, OPTIONS request!
```

PROFIND method:



```
Status: 200 OK   Size: 24 Bytes   Time: 4 ms
Hello, PROPFIND request!
```

Task 3

Aim:Test it using browser ,CLI and REST Client

Theoretical Background:

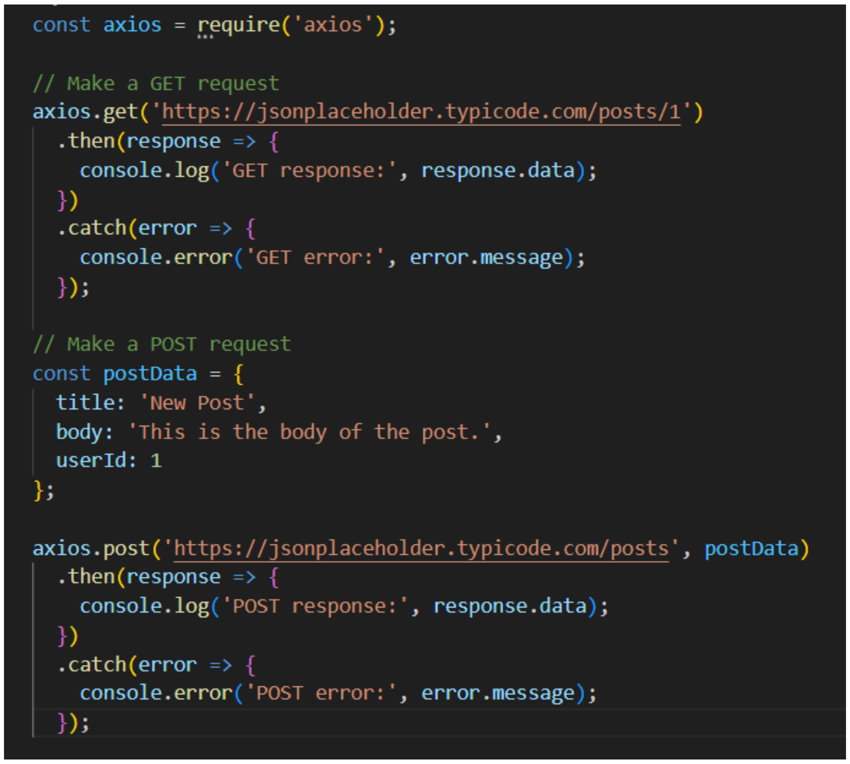
CLI:

Command-line interfaces (CLIs) built in Node.js allow you to automate repetitive tasks while leveraging the vast Node.js ecosystem. And thanks to package managers like npm and yarn, these can be easily distributed and consumed across multiple platforms.

REST Client:

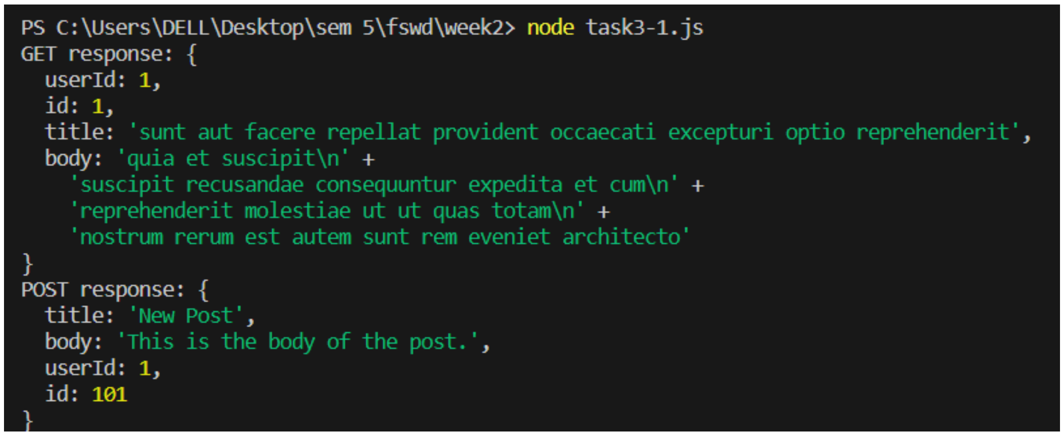
In a RESTful API, there are three main usage elements: the client call to the API, the API interface, and the server. The uniform interface is like a switchboard between the client and the server that also confirms that the client has authorization to send HTTP requests to the server.

Source Code:



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

// Make a GET request
axios.get('https://jsonplaceholder.typicode.com/posts/1')
  .then(response => {
    console.log('GET response:', response.data);
  })
  .catch(error => {
    console.error('GET error:', error.message);
  });

// Make a POST request
const postData = {
  title: 'New Post',
  body: 'This is the body of the post.',
  userId: 1
};

axios.post('https://jsonplaceholder.typicode.com/posts', postData)
  .then(response => {
    console.log('POST response:', response.data);
  })
  .catch(error => {
    console.error('POST error:', error.message);
  });
```

Output:



```
PS C:\Users\DELL\Desktop\sem 5\fswd\week2> node task3-1.js
GET response: {
  userId: 1,
  id: 1,
  title: 'sunt aut facere repellat provident occaecati excepturi optio reprehenderit',
  body: 'quia et suscipit\n' +
    'suscipit recusandae consequuntur expedita et cum\n' +
    'reprehenderit molestiae ut ut quas totam\n' +
    'nostrum rerum est autem sunt rem eveniet architecto'
}
POST response: {
  title: 'New Post',
  body: 'This is the body of the post.',
  userId: 1,
  id: 101
}
```

Task 4

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

Theoretical Background:

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

This line imports the built-in Node.js module fs (file system module), which provides methods for interacting with the file system. It allows you to read and write files, among other file-related operations.

2) const lines = data.split('\n');

Assuming no error occurred, this code splits the content of the file into an array of lines using the split method. Each line in the file is separated by a new line character ('\n').

3)
```
console.log('Filtered Students:');
 lines.forEach((line) => {
   const [name, id, cgpa] = line.split(' ');
   if (id.includes('MA') && parseFloat(cgpa) > 7) {
     console.log(line);
   }
 });
```

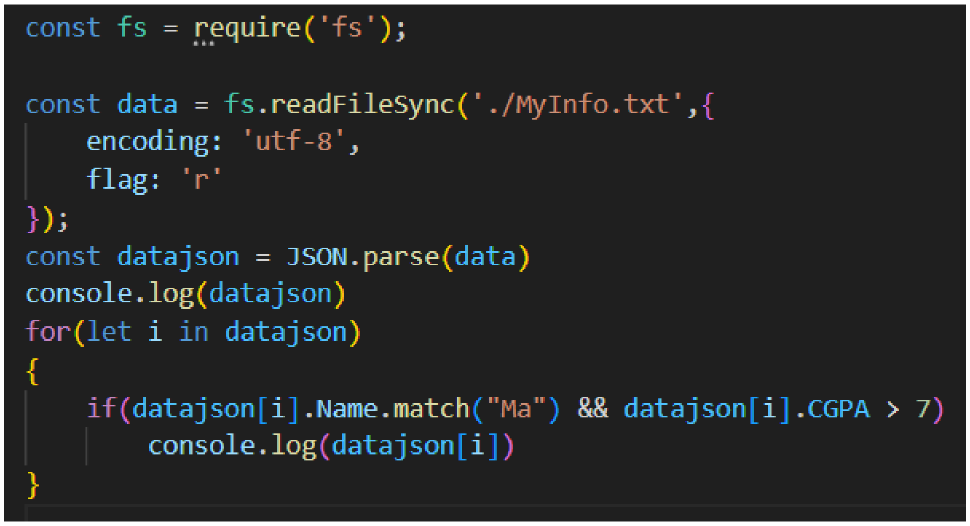
After splitting the content into lines, this code begins iterating over each line using the forEach method of the lines array. For each line, it splits the line into separate components (name, ID, and CGPA) using the split method, with the space character (' ') as the separator.

Then, it checks if the ID component (id) includes the string 'MA' and if the CGPA component (cgpa) parsed as a floating-point number is greater than 7. If both conditions are true, it prints the entire line using console.log(line).

This code filters and prints the lines that contain the substring 'MA' in the ID component and have a CGPA greater than 7.

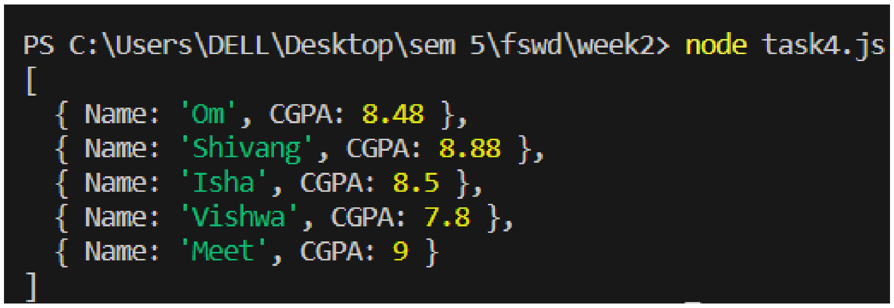
Please note that the explanation assumes the file 'student-data.txt' exists and contains data in the specified format.

Source Code:



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

const data = fs.readFileSync('./MyInfo.txt',{
    encoding: 'utf-8',
    flag: 'r'
});
const datajson = JSON.parse(data)
console.log(datajson)
for(let i in datajson)
{
    if(datajson[i].Name.match("Ma") && datajson[i].CGPA > 7)
        console.log(datajson[i])
}
```

Output:



```
PS C:\Users\DELL\Desktop\sem 5\fswd\week2> node task4.js
[
  { Name: 'Om', CGPA: 8.48 },
  { Name: 'Shivang', CGPA: 8.88 },
  { Name: 'Isha', CGPA: 8.5 },
  { Name: 'Vishwa', CGPA: 7.8 },
  { Name: 'Meet', CGPA: 9 }
]
```

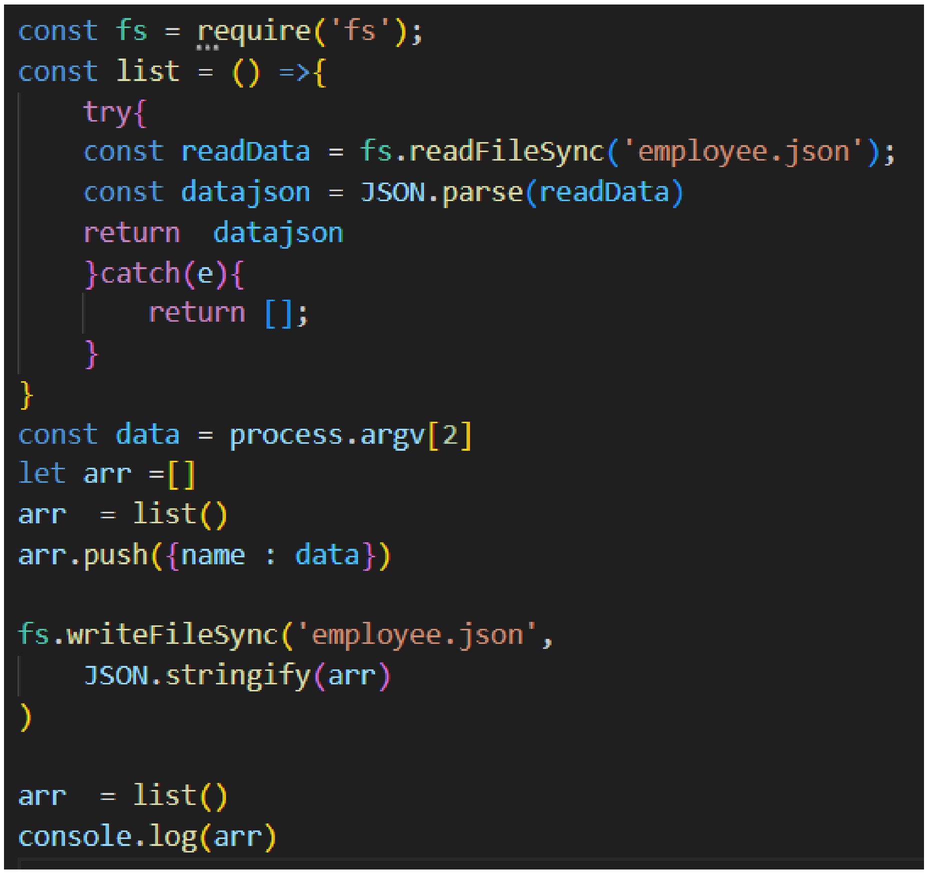
Task 5

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

Theoretical Background:

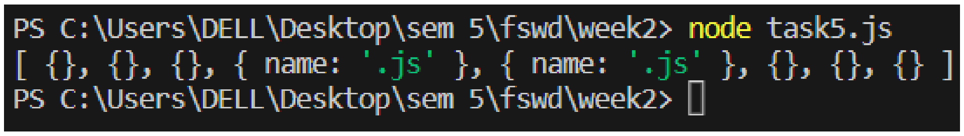
- The first line imports the fs module, which provides us with the functions we need to read and write files.
- The second line imports the JSON object, which provides us with the functions we need to work with JSON data.
- The third line creates a dictionary to store the employee information.

Source Code:



```
const fs = require('fs');
const list = () =>{
    try{
    const readData = fs.readFileSync('employee.json');
    const datajson = JSON.parse(readData)
    return  datajson
    }catch(e){
        return [];
    }
}
const data = process.argv[2]
let arr =[]
arr   = list()
arr.push({name : data})

fs.writeFileSync('employee.json',
    JSON.stringify(arr)
)

arr   = list()
console.log(arr)
```

Output:



```
PS C:\Users\DELL\Desktop\sem 5\fswd\week2> node task5.js
[ {}, {}, {}, { name: '.js' }, { name: '.js' }, {}, {}, {} ]
PS C:\Users\DELL\Desktop\sem 5\fswd\week2> 
```

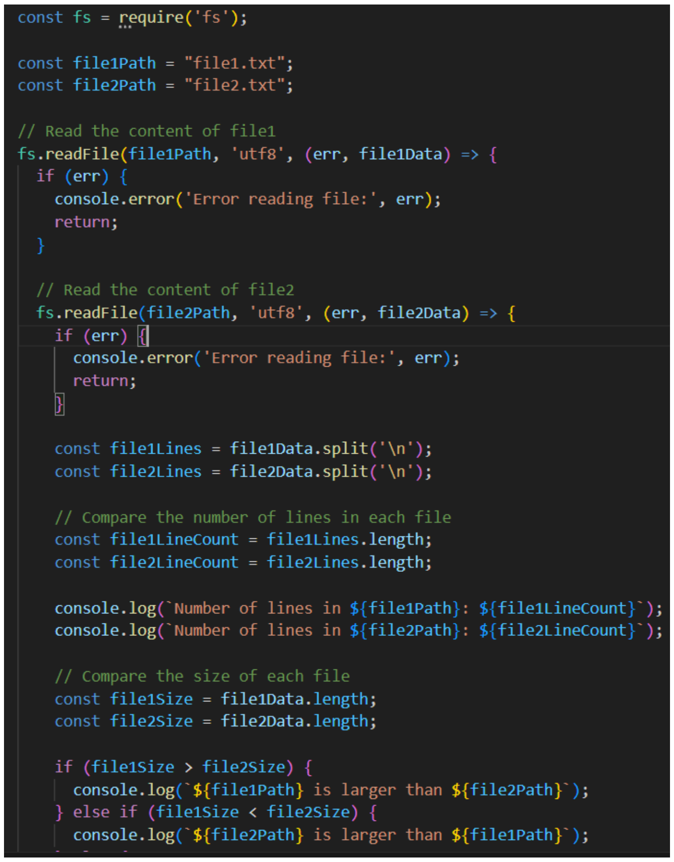
Task 6

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

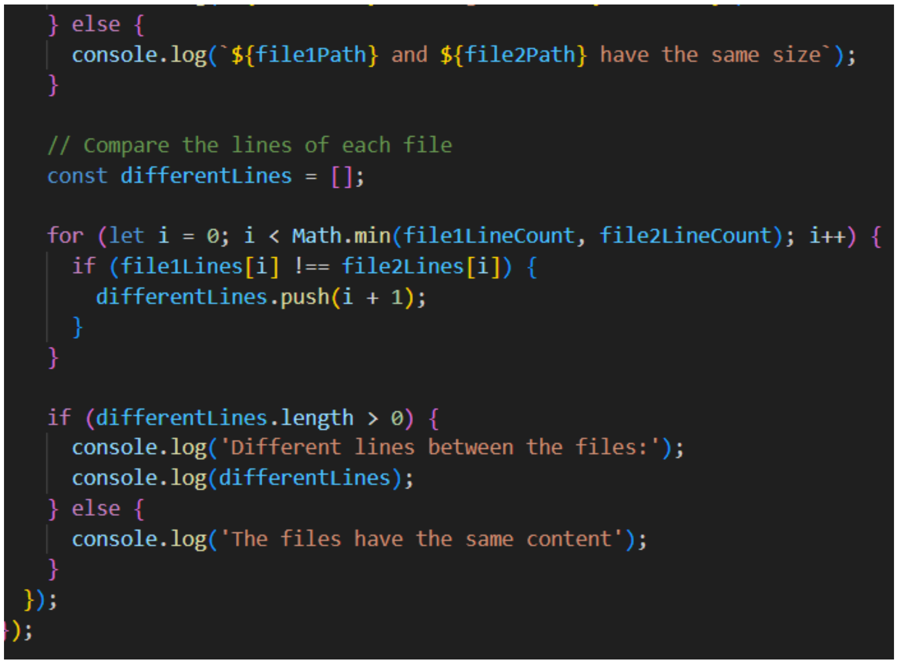
Theoretical Background:

- First, we need to import the fs module, which provides us with the functions we need to read and write files.
- Next, we need to open the two files in read mode.
- Then, we need to create a loop that will iterate through the lines of each file.
- In the loop, we need to compare the lines of the two files.
- If the lines are the same, we do nothing.
- If the lines are different, we need to print a message that indicates the line number and the difference between the two lines.
- After the loop, we need to close the two files.

Source Code:

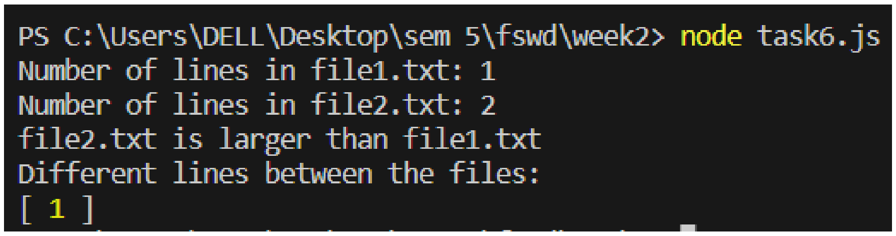


```
const fs = require('fs');

const file1Path = "file1.txt";
const file2Path = "file2.txt";

// Read the content of file1
fs.readFile(file1Path, 'utf8', (err, file1Data) => {
  if (err) {
    console.error('Error reading file:', err);
    return;
  }

  // Read the content of file2
  fs.readFile(file2Path, 'utf8', (err, file2Data) => {
    if (err) {
      console.error('Error reading file:', err);
      return;
    }

    const file1Lines = file1Data.split('\n');
    const file2Lines = file2Data.split('\n');

    // Compare the number of lines in each file
    const file1LineCount = file1Lines.length;
    const file2LineCount = file2Lines.length;

    console.log(`Number of lines in ${file1Path}: ${file1LineCount}`);
    console.log(`Number of lines in ${file2Path}: ${file2LineCount}`);

    // Compare the size of each file
    const file1Size = file1Data.length;
    const file2Size = file2Data.length;

    if (file1Size > file2Size) {
      console.log(`${file1Path} is larger than ${file2Path}`);
    } else if (file1Size < file2Size) {
      console.log(`${file2Path} is larger than ${file1Path}`);
```



```
  } else {
    console.log(`${file1Path} and ${file2Path} have the same size`);
  }

  // Compare the lines of each file
  const differentLines = [];

  for (let i = 0; i < Math.min(file1LineCount, file2LineCount); i++) {
    if (file1Lines[i] !== file2Lines[i]) {
      differentLines.push(i + 1);
    }
  }

  if (differentLines.length > 0) {
    console.log('Different lines between the files:');
    console.log(differentLines);
  } else {
    console.log('The files have the same content');
  }
 });
});
```

Output:



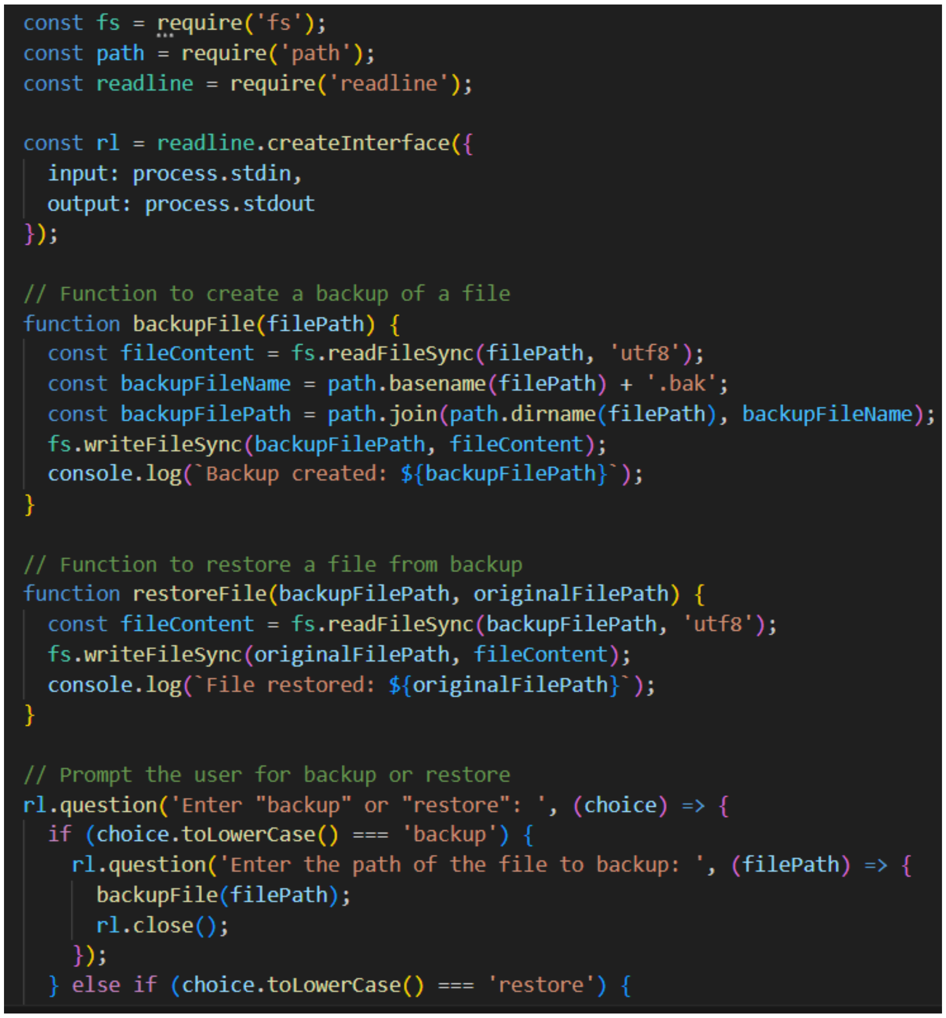
```
PS C:\Users\DELL\Desktop\sem 5\fswd\week2> node task6.js
Number of lines in file1.txt: 1
Number of lines in file2.txt: 2
file2.txt is larger than file1.txt
Different lines between the files:
[ 1 ]
```

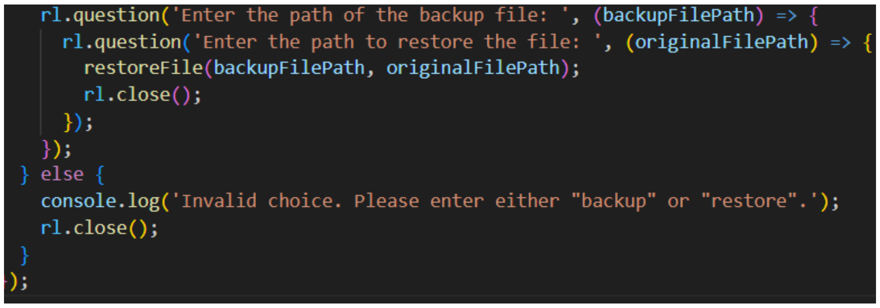
Task 7

Aim:Create File Backup and Restore Utility

Theoretical Background:we need to create a function that will take a file path as input and create a backup of the file. The function should create a new file with the same name as the original file, but with the suffix .bak. This code will prompt the user for a file path and then ask the user whether they want to backup or restore the file. If the user chooses to backup the file, the code will create a backup of the file. If the user chooses to restore the file, the code will delete the original file and then rename the backup file to the original file name.

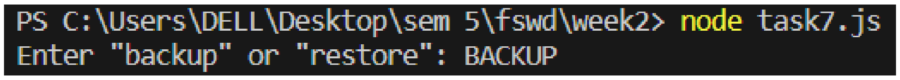
Source Code:


```
const fs = require('fs');
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 backupFile(filePath) {
  const fileContent = fs.readFileSync(filePath, 'utf8');
  const backupFileName = path.basename(filePath) + '.bak';
  const backupFilePath = path.join(path.dirname(filePath), backupFileName);
  fs.writeFileSync(backupFilePath, fileContent);
  console.log(`Backup created: ${backupFilePath}`);
}

// Function to restore a file from backup
function restoreFile(backupFilePath, originalFilePath) {
  const fileContent = fs.readFileSync(backupFilePath, 'utf8');
  fs.writeFileSync(originalFilePath, fileContent);
  console.log(`File restored: ${originalFilePath}`);
}

// Prompt the user for backup or restore
rl.question('Enter "backup" or "restore": ', (choice) => {
  if (choice.toLowerCase() === 'backup') {
    rl.question('Enter the path of the file to backup: ', (filePath) => {
      backupFile(filePath);
      rl.close();
    });
  } else if (choice.toLowerCase() === 'restore') {
```

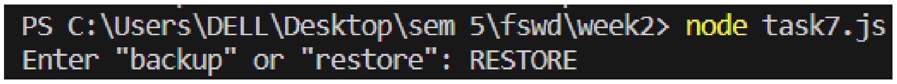


```
    rl.question('Enter the path of the backup file: ', (backupFilePath) => {
      rl.question('Enter the path to restore the file: ', (originalFilePath) => {
        restoreFile(backupFilePath, originalFilePath);
        rl.close();
      });
    });
  } else {
    console.log('Invalid choice. Please enter either "backup" or "restore".');
    rl.close();
  }
});
```

Output:



```
PS C:\Users\DELL\Desktop\sem 5\fswd\week2> node task7.js
Enter "backup" or "restore": BACKUP
```



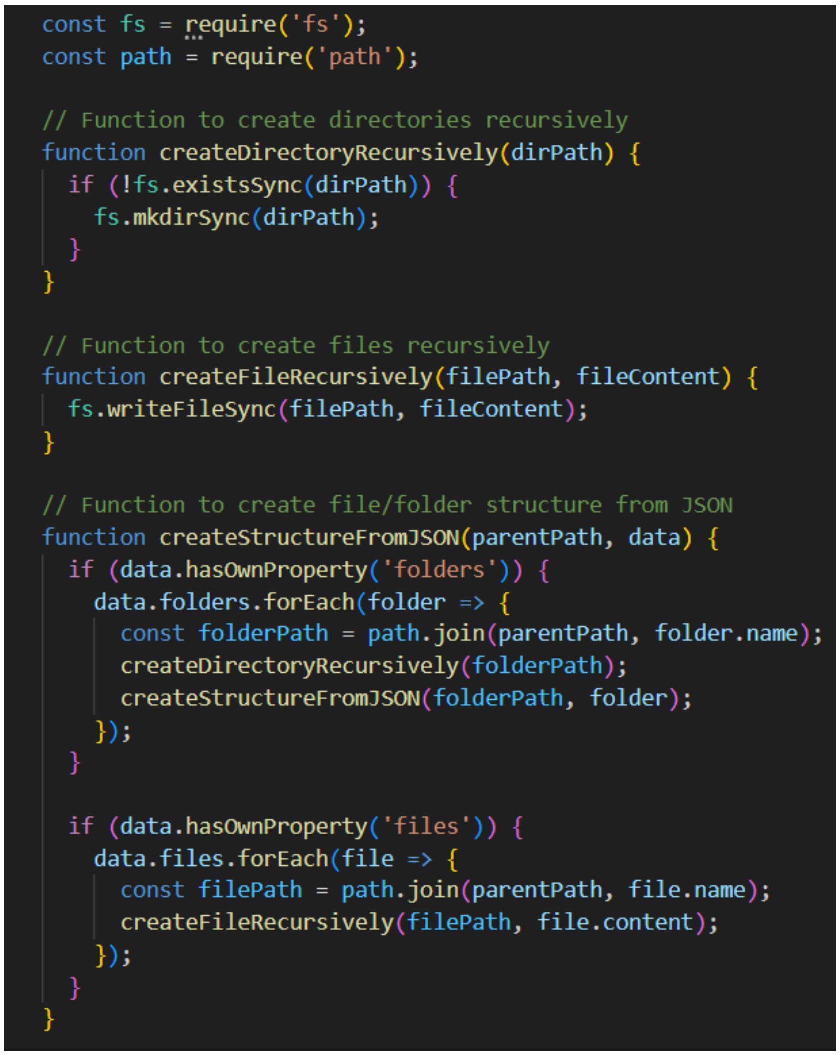
```
PS C:\Users\DELL\Desktop\sem 5\fswd\week2> node task7.js
Enter "backup" or "restore": RESTORE
```

Task 8

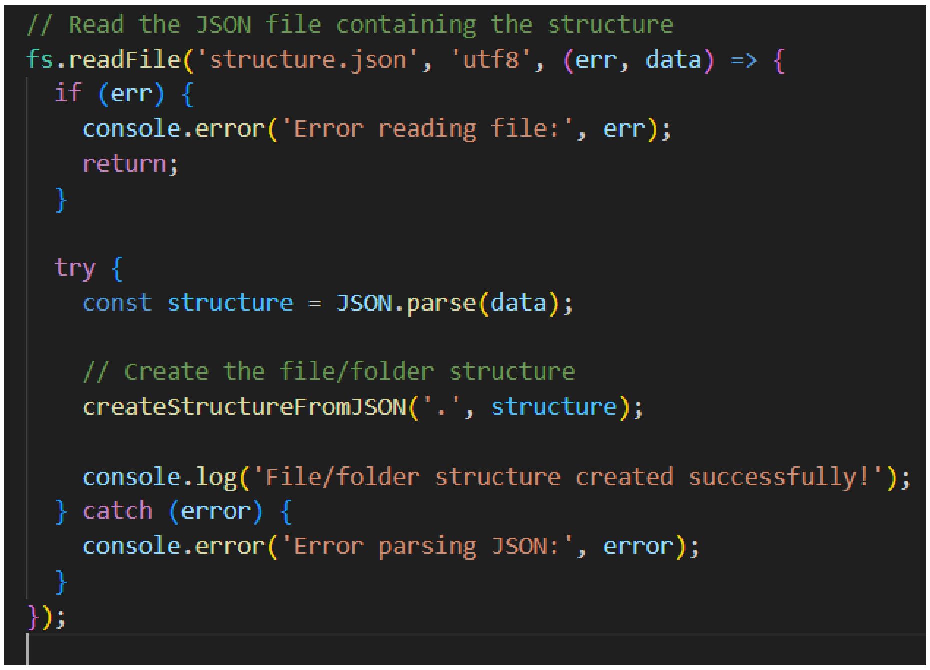
Aim:Create File/Folder Structure given in json file.

Theoretical Background:
we need to load the JSON file into a JavaScript object. Then, we need to iterate through the JSON object and create the corresponding files and folders.
The JSON object will contain a list of files and folders. The code will then iterate through the list and create the corresponding files and folders. Finally, the code will save the changes to the file system.

Source Code:

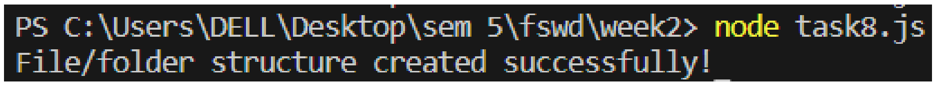


```
const fs = require('fs');
const path = require('path');

// Function to create directories recursively
function createDirectoryRecursively(dirPath) {
  if (!fs.existsSync(dirPath)) {
    fs.mkdirSync(dirPath);
  }
}

// Function to create files recursively
function createFileRecursively(filePath, fileContent) {
  fs.writeFileSync(filePath, fileContent);
}

// Function to create file/folder structure from JSON
function createStructureFromJSON(parentPath, data) {
  if (data.hasOwnProperty('folders')) {
    data.folders.forEach(folder => {
      const folderPath = path.join(parentPath, folder.name);
      createDirectoryRecursively(folderPath);
      createStructureFromJSON(folderPath, folder);
    });
  }

  if (data.hasOwnProperty('files')) {
    data.files.forEach(file => {
      const filePath = path.join(parentPath, file.name);
      createFileRecursively(filePath, file.content);
    });
  }
}
```



```
// Read the JSON file containing the structure
fs.readFile('structure.json', 'utf8', (err, data) => {
  if (err) {
    console.error('Error reading file:', err);
    return;
  }

  try {
    const structure = JSON.parse(data);

    // Create the file/folder structure
    createStructureFromJSON('.', structure);

    console.log('File/folder structure created successfully!');
  } catch (error) {
    console.error('Error parsing JSON:', error);
  }
});
```

Output:



```
PS C:\Users\DELL\Desktop\sem 5\fswd\week2> node task8.js
File/folder structure created successfully!
```

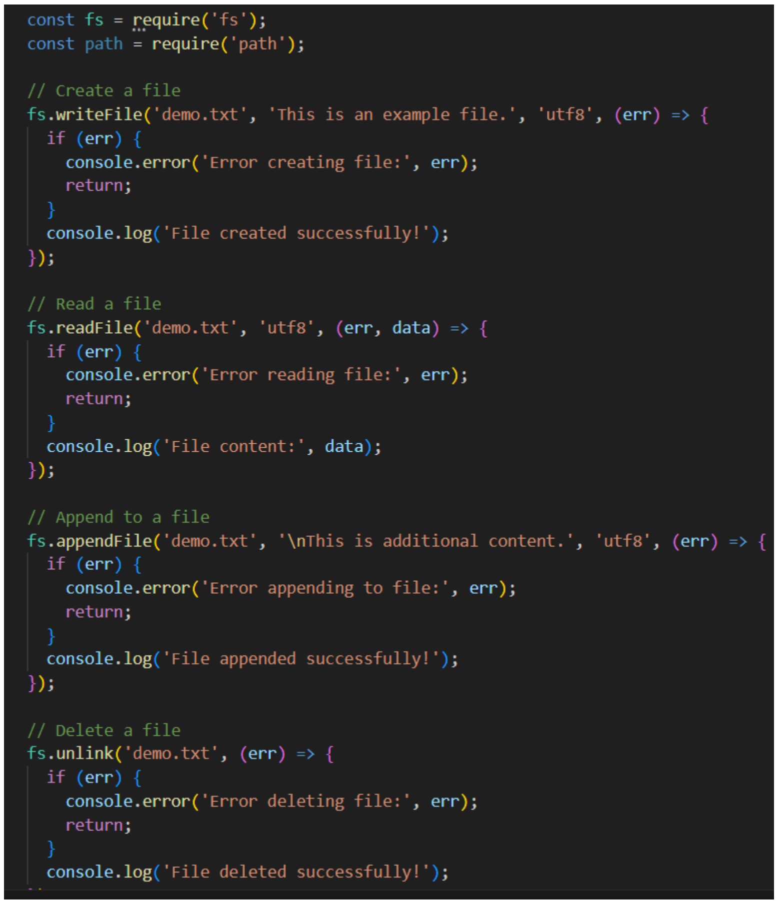
Task 9

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

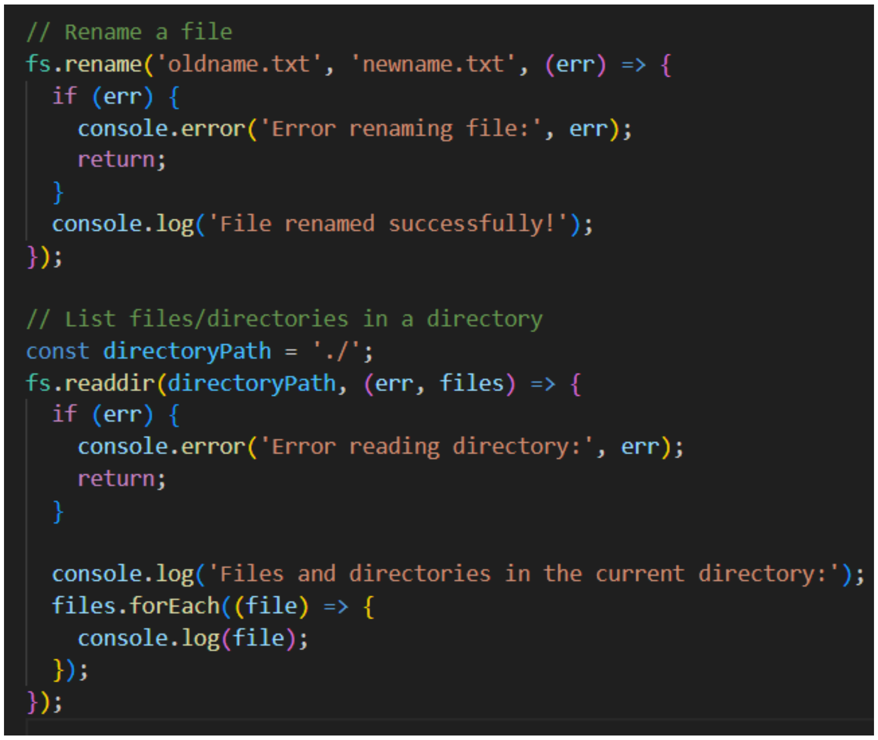
Theoretical Background:

- Create File: This operation creates a new file.
- Read File: This operation reads the contents of a file.
- Append File: This operation appends data to the end of a file.
- Delete File: This operation deletes a file.
- Rename File: This operation renames a file.
- List Files/Dirs: This operation lists the files and directories in a directory.

Source Code:

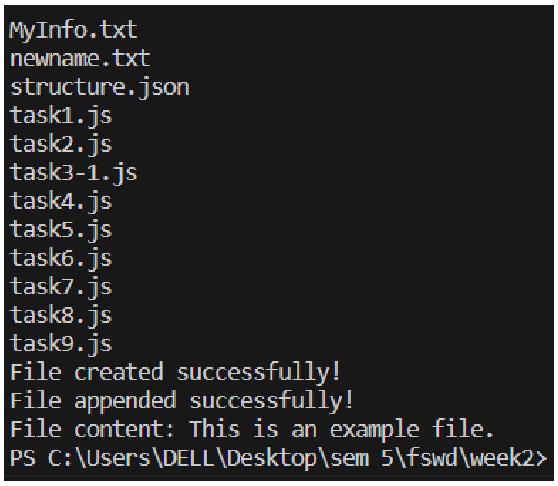


```
const fs = require('fs');
const path = require('path');

// Create a file
fs.writeFile('demo.txt', 'This is an example file.', 'utf8', (err) => {
  if (err) {
    console.error('Error creating file:', err);
    return;
  }
  console.log('File created successfully!');
});

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

// Append to a file
fs.appendFile('demo.txt', '\nThis is additional content.', 'utf8', (err) => {
  if (err) {
    console.error('Error appending to file:', err);
    return;
  }
  console.log('File appended successfully!');
});

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



```
// Rename a file
fs.rename('oldname.txt', 'newname.txt', (err) => {
  if (err) {
    console.error('Error renaming file:', err);
    return;
  }
  console.log('File renamed successfully!');
});

// List files/directories in a directory
const directoryPath = './';
fs.readdir(directoryPath, (err, files) => {
  if (err) {
    console.error('Error reading directory:', err);
    return;
  }

  console.log('Files and directories in the current directory:');
  files.forEach((file) => {
    console.log(file);
  });
});
```

Output:



```
MyInfo.txt
newname.txt
structure.json
task1.js
task2.js
task3-1.js
task4.js
task5.js
task6.js
task7.js
task8.js
task9.js
File created successfully!
File appended successfully!
File content: This is an example file.
PS C:\Users\DELL\Desktop\sem 5\fswd\week2>
```

Course Outcome:

CO1 : Understand various technologies and trends impacting single page web applications.

CO4 : Demonstrate the use of JavaScript to fulfill the essentials of front-end development To back-end development.

Conclusion:

From the above performed tasks, we learnt about:

1. Installing and configuring NodeJS, NPM, NPX
2. Installing and starting with nodemon package
3. Setting Up the Development Environment and Working with Node.js Script