# **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.

**Description:**

1).Initialize a new Node.js project by running npm init in your project directory and installing the required dependencies (e.g., Express.js) using npm install.

2. Create a new JavaScript file (e.g., server.js) and import the necessary modules, such as http and express.

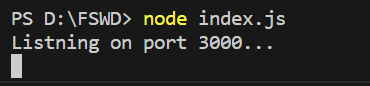
3. Create an instance of the HTTP server and define a route handler to respond to incoming requests with a simple message.

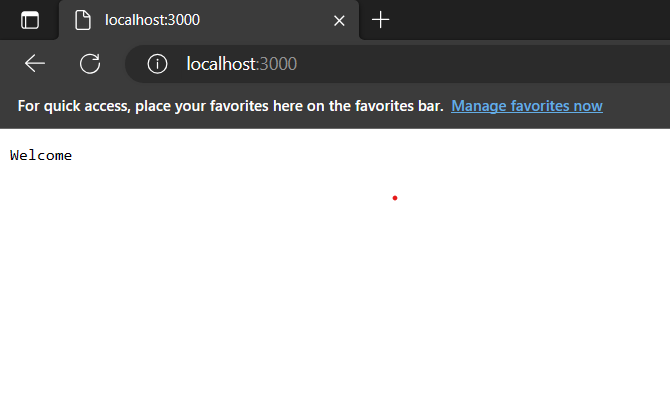
4.Start the server by specifying a port number (e.g., 3000) and listen for incoming HTTP requests.

**Source Code:**



**Output:**

****

****

# **Task-2**

**Aim:**

Experiment with Various HTTP Methods,Content Types and Status Code

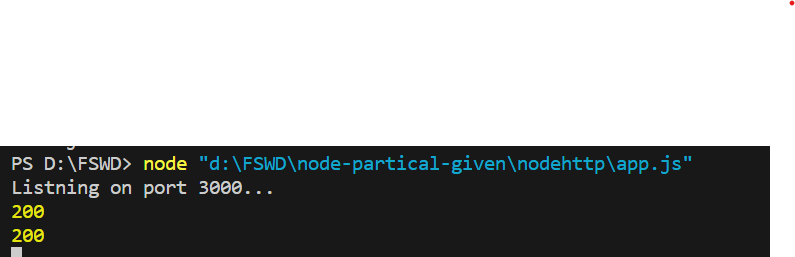
**Description:**

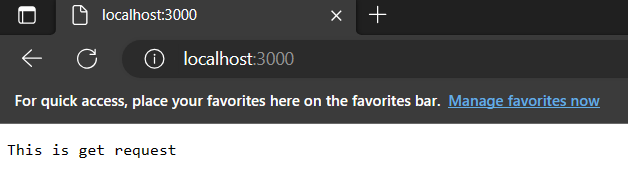
* 100: Continue
* 101: Switching Protocols
* 200: OK
* 201: Created
* 204: No Content
* 301: Moved Permanently
* 302: Found (Moved Temporarily)
* 304: Not Modified
* 400: Bad Request
* 401: Unauthorized
* 403: Forbidden
* 404: Not Found
* 409: Conflict
* 500: Internal Server Error
* 501: Not Implemented
* 503: Service Unavailable
* GET: Requests a representation of the specified resource.
* POST: Submits data to be processed to the specified resource.
* PUT: Updates the specified resource with the provided data.
* DELETE: Deletes the specified resource.
* PATCH: Partially updates the specified resource.
* HEAD: Requests the headers of the specified resource without the response body. OPTIONS: Requests information about the communication options available for the specified resource.

**Source code:**



**Output:**

****



# **Task-3**

**Aim:**

Test it using browser ,CLI and REST Client

**Description:**

A REST client is a tool or application used to interact with RESTful web services. It allows users to send HTTP requests, such as GET, POST, PUT, or DELETE, to perform operations on server resources. REST clients can be accessed through web browsers, enabling easy testing and visualization of responses. Command-line tools like CURL offer flexibility for scripting and automation. Dedicated REST client applications, such as Postman or Insomnia, provide a comprehensive environment for testing, managing collections, and advanced authentication options. These different methods cater to various testing needs and enhance the development and consumption of RESTful APIs.

**Source Code:**

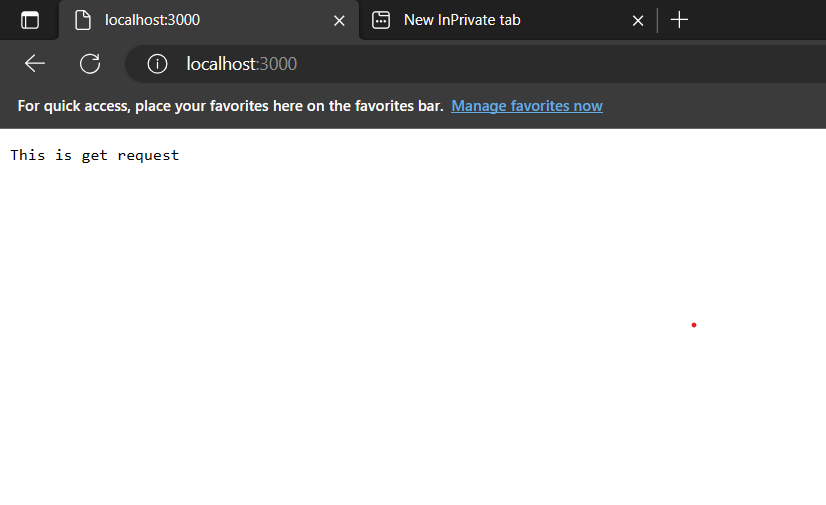
1)Typing localhost:3000 in web browser.

2)cmd:curl -X GET http://localhost:3000

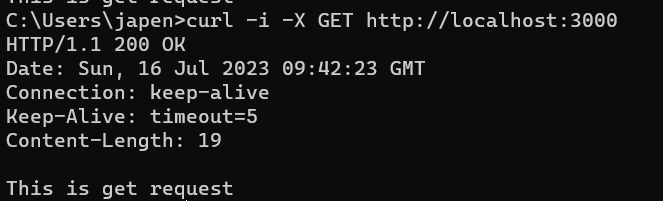
3) Typing localhost:3000 in ThunderClient

**Output:**

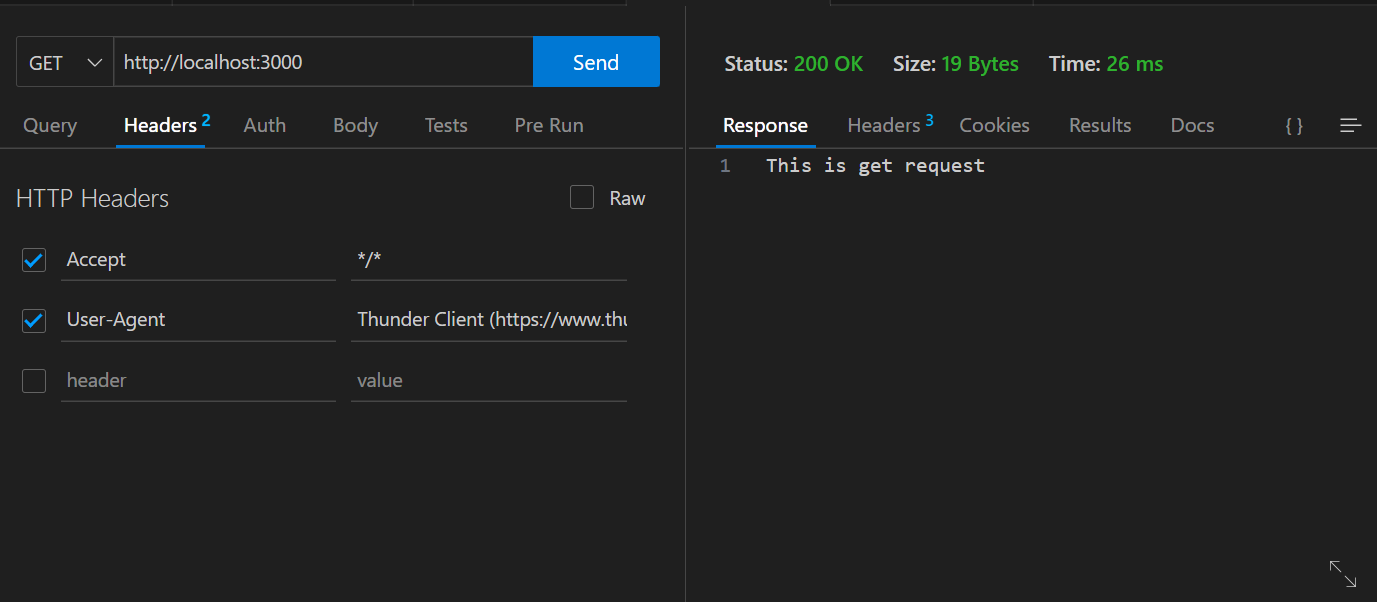
**Web Browser:**



**CLI:**

****

**Client:(ThunderClient)**

****

# **Task-4**

**Aim:**

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

**Description:**

In this i create a file student\_data.txt which contains the data

Name CGPA

manish 6.0

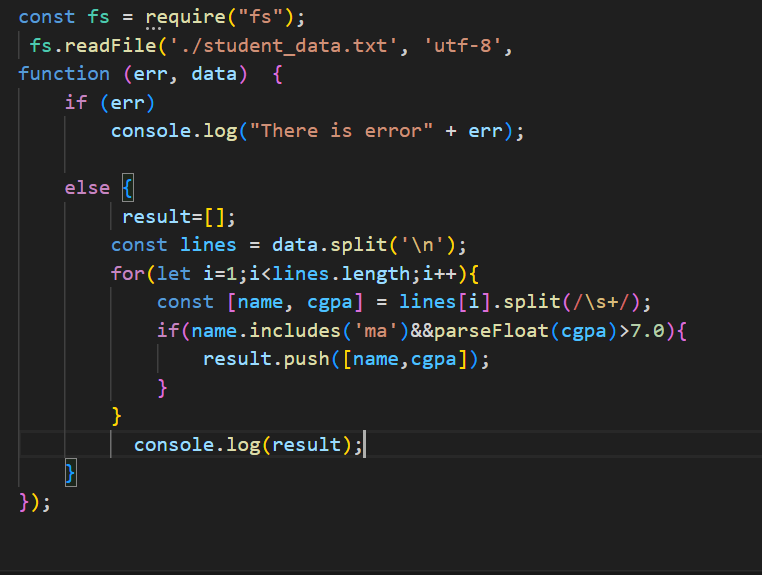
suresh 10.0

mahesh 9.99

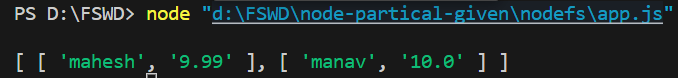
manav 10.0

And then i split the lines of the file by \n and then split each line by blank space so i got name and cgpa of each student then checked the given condition(whose name contains ‘MA’ and CGPA > 7.)

**Source code:**



**Output:**

****

# **Task-5**

**Aim:**

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

**Description:**

In this i used the readline module in nodejs to take input from user and used fs module to write the data into the json file by using the JSON>stringify which converts the object into the json format and using write and providing the file name the data is entered into the employee-data.json file

**Source code:**

const fs = require('fs');

const readline = require('readline');

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

rl.question('Enter employee name: ', (name) => {

rl.question('Enter employee ID: ', (id) => {

rl.question('Enter employee position: ', (position) => {

const employee = {

name,

id,

position

};

const employeeData = JSON.stringify(employee);

fs.writeFile('employee-data.json', employeeData, 'utf8', (err) => {

if (err) {

console.error(err);

return;

}

console.log('Added Sucessfully');

rl.close();

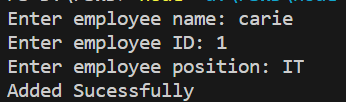
});

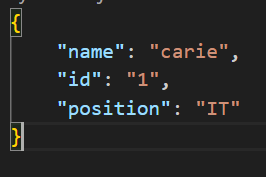
});

});

});

**Output:**





# **Task-6**

**Aim:**

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

**Description:**

In this i installed the diff module for getting different lines in two files and i used readfilesync for reading the file so that i could get length and compare them.

**Source Code:**

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

**const diff = require('diff');**

**function compareFiles(file1Path, file2Path) {**

**const file1 = fs.readFileSync(file1Path, 'utf8');**

**const file2 = fs.readFileSync(file2Path, 'utf8');**

**if (file1.length > file2.length) {**

**console.log(`${file1Path} is larger than ${file2Path}`);**

**} else if (file1.length < file2.length) {**

**console.log(`${file2Path} is larger than ${file1Path}`);**

**} else {**

**console.log('Both files have the same size');**

**}**

**const differences = diff.diffLines(file1, file2);**

**differences.forEach((part) => {**

**const indicator = part.added ? '+' : part.removed ? '-' : ' ';**

**console.log(`${indicator} ${part.value.trim()}`);**

**});**

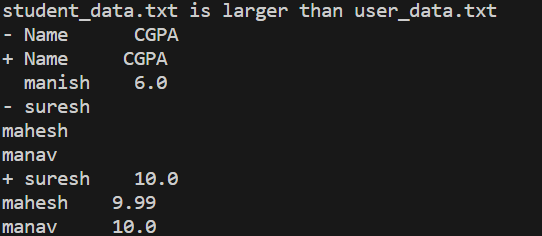
**}**

**const file2Path = 'student\_data.txt';**

**const file1Path = 'user\_data.txt';**

**compareFiles(file1Path, file2Path);**

**output:**

****

# **Task-7**

# **Aim:**

Create File Backup and Restore Utility

**Description:**

I Created a backup of the file by copying it with a .bak extension. Modified the file by overwriting its content with the desired changes. Restored the file from the backup by copying the contents of the backup file back to the original file. Deleted the backup file.

**Source Code:**

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

**const path = require('path');**

**function createBackup(filePath) {**

**const backupPath = filePath + '.bak';**

**fs.copyFileSync(filePath, backupPath);**

**console.log(`Backup created: ${backupPath}`);**

**}**

**function restoreBackup(filePath) {**

**const backupPath = filePath + '.bak';**

**if (fs.existsSync(backupPath)) {**

**fs.copyFileSync(backupPath, filePath);**

**console.log(`Backup restored: ${filePath}`);**

**fs.unlinkSync(backupPath);**

**console.log(`Backup deleted: ${backupPath}`);**

**} else {**

**console.log(`Backup not found: ${backupPath}`);**

**}**

**}**

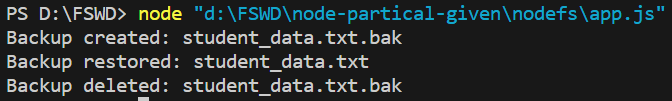
**const filePath = 'student\_data.txt';**

**createBackup(filePath);**

**fs.writeFileSync(filePath, 'Modified content');**

**restoreBackup(filePath);**

**output:**

****

# **Task-8**

# **Aim:**

Create File/Folder Structure given in json file.

**Description:**

The created FileStructure function takes a directory and a structure as parameters. It iterates over each item in the structure array. For each item, it checks whether it is a file or a folder. If it's a file, it creates the file with the specified name and content in the given directory. If it's a folder, it creates the folder and recursively calls createFileStructure with the folder path and its children. The directoryPath variable represents the base directory where the file/folder structure will be created. The fileStructure variable is an array that defines the structure and contents of the files and folders to be created. It contains objects representing files and folders, with properties like name, type, and optionally content. When the code is executed, it will create the file/folder structure defined in fileStructure within the directory Path directory

**Source Code:**

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

**const path = require('path');**

**function createFileStructure(directory, structure) {**

**for (const item of structure) {**

**const itemPath = path.join(directory, item.name);**

**if (item.type === 'file') {**

**fs.writeFileSync(itemPath, item.content || '');**

**console.log(`Created file: ${itemPath}`);**

**} else if (item.type === 'folder') {**

**fs.mkdirSync(itemPath);**

**console.log(`Created folder: ${itemPath}`);**

**if (item.children && item.children.length > 0) {**

**createFileStructure(itemPath, item.children);**

**}**

**}**

**}**

**}**

**const directoryPath = '';**

**const fileStructure = [**

**{**

**"name": "folder1",**

**"type": "folder",**

**"children": [**

**{**

**"name": "file1.txt",**

**"type": "file",**

**"content": "File 1 content"**

**},**

**{**

**"name": "folder2",**

**"type": "folder",**

**"children": [**

**{**

**"name": "file2.txt",**

**"type": "file",**

**"content": "File 2 content"**

**}**

**]**

**}**

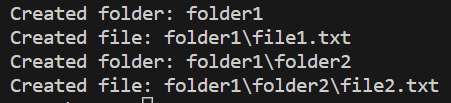
**]**

**}**

**];**

**createFileStructure(directoryPath, fileStructure);**

**output:**

****

# **Task-9**

# **Aim:**

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

**Description:**

In this i used the fs module for the creating,reading Appending and deleting,renaming and performing serval operation on it explored the various function such as read,write etc.

**Source Code:**

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

**const path = require('path');**

**function createFile(filePath, content) {**

**fs.writeFileSync(filePath, content);**

**console.log(`File created: ${filePath}`);**

**}**

**function readFile(filePath) {**

**const content = fs.readFileSync(filePath, 'utf8');**

**console.log(`File content: ${content}`);**

**}**

**function appendToFile(filePath, content) {**

**fs.appendFileSync(filePath, content);**

**console.log(`Appended to file: ${filePath}`);**

**}**

**function deleteFile(filePath) {**

**fs.unlinkSync(filePath);**

**console.log(`File deleted: ${filePath}`);**

**}**

**function renameFile(oldFilePath, newFilePath) {**

**fs.renameSync(oldFilePath, newFilePath);**

**console.log(`File renamed: ${oldFilePath} to ${newFilePath}`);**

**}**

**function listFilesAndDirs(directoryPath) {**

**const filesAndDirs = fs.readdirSync(directoryPath);**

**console.log(`Files and directories in ${directoryPath}:`);**

**for (const item of filesAndDirs) {**

**const itemPath = path.join(directoryPath, item);**

**const isDir = fs.statSync(itemPath).isDirectory();**

**const itemType = isDir ? 'directory' : 'file';**

**console.log(`${item} (${itemType})`);**

**}**

**}**

**const filePath = 'temp.txt';**

**const newFilePath = 'temp1.txt';**

**const directoryPath = 'new';**

**createFile(filePath, 'Hello, world!');**

**readFile(filePath);**

**appendToFile(filePath, '\nThis is an appended line.');**

**readFile(filePath);**

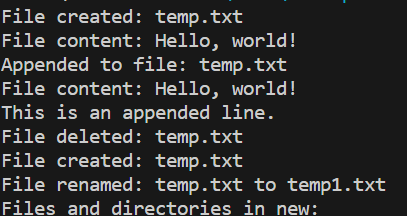
**deleteFile(filePath);**

**createFile(filePath, 'Hello again!');**

**renameFile(filePath, newFilePath);**

**listFilesAndDirs(directoryPath);**

**output:**

****

**Conclusion:**

I created files, read file content, appended content to files, deleted files, renamed files, and listed files and directories using the built-in fs (file system) module in Node.js. These operations were tested using different methods: through a browser by making HTTP requests to the Node.js server, through the command-line interface (CLI) by executing the Node.js application, and using a REST client tool such as Postman or Insomnia to simulate HTTP requests. Additionally,i read the content of a file (student-data.txt), filtered the student records based on certain criteria (name containing 'MA' and CGPA > 7), and extracted the desired results. I also implemented a functionality to read employee information from the user and write the data to a file called employee-data.json. Furthermore, I compared two files and determined which file is larger, as well as identified the differing lines between them. Lastly, i created a file backup and restore utility that allowed for creating a backup of a file, modifying the file, and restoring it from the backup.Overall i learnt about the file management in nodejs and some of the modules such http

Diff.