Week - 4
Task-1

Aim: URL Parsing and Manipulation:

- Write a program that accepts a URL as user input and uses the url module to parse it. Display the protocol, host, path, and query parameters separately.
- Implement a function that takes a base URL and a relative path as input, and uses the url module to resolve and display the absolute URL.

Description:

- URL parsing and manipulation in Node.js is facilitated by the built-in `url` module. This module provides several functions to work with URLs, allowing developers to parse, construct, and manipulate URLs easily.

1. **Parsing URLs:**
   The `url.parse()` function is used to parse a URL string and extract its different components such as the protocol, host, path, query parameters, etc. It returns an object containing these components.

2. **Constructing URLs:**
   The `url.format()` function is used to construct a URL string from its components. It takes an object containing the components and returns a formatted URL string.

3. **URL Class:**
   Node.js also provides a `URL` class (introduced in Node.js v7.5.0) for working with URLs. It offers an improved and more modern API compared to the legacy `url.parse()` and `url.format()` functions.

4. **Resolving URLs:**
   The `url.resolve()` function is used to resolve a relative URL against a base URL and construct the absolute URL. This is useful when you want to navigate from one URL to another.

With these capabilities provided by the `url` module, developers can easily work with URLs in Node.js, parse them into their components, construct new URLs, and resolve relative URLs to their absolute forms.

Source Code & Output:

Task1.1

Welcome | Task_1.js

Task_1 > Task_1.js > rl.question('Enter the URL: ') callback

```javascript
const readline = require('readline');
const urlmod = require('url');

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

function parseURL(input) {
  const parsed = new urlmod.URL(input);

  console.log('Protocol:', parsed.protocol);
  console.log('Host:', parsed.host);
  console.log('Path:', parsed.pathname);
  console.log('Query Parameters:');
  const query = parsed.searchParams;
  query.forEach((value, name) => {
    console.log(`${name}:${value}`);
  });
}
rl.question('Enter the URL: ', (input) => {
  parseURL(input);
  rl.close();
});
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

```
D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_1>node Task_1.js
Enter the URL: http://google.com/default?name=rushi&id=21it145
Protocol: http:
Host: google.com
Path: /default
Query Parameters:
name:rushi
id:21it145

D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_1>
```

Task 1.2

Welcome | Task_1.js

Task_1 > Task_1.js > resolveAbsoluteURL

```javascript
const urlmod = require('url');

function resolveAbsoluteURL(baseUrl, relativePath) {
  const resolvedURL = urlmod.resolve(baseUrl, relativePath);
  console.log(' URL:', resolvedURL);
}

const Url = 'https://www.googlecloud.com/default/';
const Path = '../homepage.html';
resolveAbsoluteURL(Url, Path);
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL

```
D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_1>node Task_1.js
 URL: https://www.googlecloud.com/homepage.html

D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_1>
```

Task-2

Aim: Query String Operation:

- Write a Node.js program that takes a URL with a query string as input and extracts the key-value pairs from the query string using the querystring module. The program should display the extracted key-value pairs as output.

Description:

- In Node.js, the "querystring" module provides utilities for working with query strings. Query strings are commonly used in URLs to pass data as key-value pairs, typically for parameters in HTTP requests. The "querystring" module allows you to parse and stringify query strings, making it easy to work with the data they contain.

1. **Parsing a Query String:** The `querystring.parse()` function is used to parse a query string and convert it into a JavaScript object.
2. **Stringifying an Object to a Query String:** The `querystring.stringify()` function is used to convert a JavaScript object into a query string.
3. **Encoding and Decoding:** The `querystring` module also provides the `querystring.escape()` and `querystring.unescape()` functions for encoding and decoding special characters in a query string.

Source Code & Output:

Task_2.js

Task_2 > Task_2.js > ...

```javascript
const url = require('url');
const querystring = require('querystring');
const readline = require('readline');

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

function extractQueryParameters(inputURL) {
  const parsedURL = url.parse(inputURL);
  const queryParameters = querystring.parse(parsedURL.query);
  return queryParameters;
}

rl.question('Enter the URL with a query string: ', (inputURL) => {
  const extractedParams = extractQueryParameters(inputURL);
  console.log('Extracted key-value pairs:');
  for (const key in extractedParams) {
    console.log(`  ${key}: ${extractedParams[key]}`);
  }
  rl.close();
});
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_2>node Task_2.js
Enter the URL with a query string: http://www.google.com/default?name=rushi&id=21it145
Extracted key-value pairs:
  name: rushi
  id: 21it145

D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_2>
```

Task-3

Aim: Path Operations:

- Create a program that accepts two file paths as input and uses the path module to determine if they refer to the same file.
- Implement a function that accepts a file path as input and uses the path module to extract the file extension. Display the extracted extension to the user.

Description:

- In Node.js, the `path` module provides utilities for working with file and directory paths. It is built-in and does not require any installation.
- Here are some of the common operations provided by the `path` module in Node.js:

1. **Joining Paths:**
   The `path.join()` function is used to join multiple path segments into a single normalized path.

2. **Normalizing Paths:**
   The `path.normalize()` function is used to normalize a given path by resolving '..' and '.' segments.

3. **Getting the Directory Name:**
   The `path.dirname()` function is used to get the directory name from a file path.

4. **Getting the File Extension:**
   The `path.extname()` function is used to get the file extension from a file path.

5. **Getting the Base Name:**
   The `path.basename()` function is used to get the base name (last portion) of a file path.

6. **Getting the File Name without Extension:**
   The `path.parse()` function is used to parse a file path and return an object containing its different components.

These are some of the useful operations provided by the `path` module in Node.js, making it easier to work with file and directory paths in a platform-independent manner.

Source Code & Output:

Task 3.1js

```js
const path = require('path');
function arePathsSame(file1Path, file2Path) {
 try {
   const file1Stats = path.resolve(file1Path);
   const file2Stats = path.resolve(file2Path);
   return file1Stats === file2Stats;
 } catch (error) {
   return false;
 }
}

const file1Path = 'D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_3';
const file2Path = 'D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_3';
const areSame =  arePathsSame(file1Path, file2Path);
console.log(`Paths refer to the same file: ${areSame}`);
```

```
D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_3>node Task_3.js
Paths refer to the same file: true

D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_3>
```

```js
const path = require('path');
function arePathsSame(file1Path, file2Path) {
 try {
   const file1Stats = path.resolve(file1Path);
   const file2Stats = path.resolve(file2Path);
   return file1Stats === file2Stats;
 } catch (error) {
   return false;
 }
}

const file1Path = 'D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4';
const file2Path = 'D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_3';
const areSame =  arePathsSame(file1Path, file2Path);
console.log(`Paths refer to the same file: ${areSame}`);
```

```
D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_3>node Task_3.js
Paths refer to the same file: false

D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_3>
```

Task 3.2js

```js
const path = require('path');
const readline = require('readline');
const fs = require('fs').promises;

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

function extractFileExtension(filePath) {
  return path.extname(filePath);
}

rl.question('Enter the file path: ',  (file1Path) => {
    const extension = extractFileExtension(file1Path);
    console.log(`File extension: ${extension}`);
    rl.close();
  });
```

```
D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_3>node Task_3(II).js
Enter the file path: D:\Clg 3nd Year\Node JS\package.json
File extension: .json

D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_3>
```

Task-4

Aim: File Paths and Operations:

- Implement a program that accepts a file path as input and uses the path module to extract the directory name and base name. Display the extracted values separately.
- Write a function that uses the fs module to check if a given file path exists. Display a success message if the file exists, or an error message if it does not.

Description:

- File path and operations in Node.js involve working with file and directory paths, reading and writing files, checking file existence, and performing various file-related operations.
- Below are some of the common file path and operations in Node.js:

1. **File Path Operations:**
   - `path.join()`: Joins multiple path segments into a single normalized path.
   - `path.normalize()`: Normalizes a given path by resolving '..' and '.' segments.
   - `path.dirname()`: Gets the directory name from a file path.
   - `path.basename()`: Gets the base name (last portion) of a file path.
   - `path.extname()`: Gets the file extension from a file path.
   - `path.parse()`: Parses a file path and returns an object containing different components.
2. **Checking File Existence:**
   - `fs.access()`: Checks if a file or directory exists and has the specified permissions.
3. **Reading and Writing Files:**
   - `fs.readFile()`: Reads the content of a file asynchronously.
   - `fs.writeFile()`: Writes data to file asynchronously, overwriting the existing file content.
   - `fs.appendFile()`: Appends data to file asynchronously, preserving existing file content.
4. **Creating and Removing Directories:**
   - `fs.mkdir()`: Creates a directory asynchronously.
   - `fs.rmdir()`: Removes a directory asynchronously.
   - `fs.readdir()`: Reads the contents of a directory asynchronously.
5. **File Stats:**
   - `fs.stat()`: Gets the file stats asynchronously, including file size, creation time, etc.
6. **Renaming and Deleting Files:**
   - `fs.rename()`: Renames a file or moves it to another location asynchronously.
   - `fs.unlink()`: Deletes a file asynchronously.

These are some of the basic file path and operations available in Node.js through the built-in `path` and `fs` modules.

Source Code & Output:

Task 4.1js

```
Task_4_1.js
Task_4 > Task_4_1.js > ...
const path = require('path');
const readline = require('readline');

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});
function pathmod(path) {
  const d = path.dirname(path);
  const b = path.basename(path);
  console.log('Directory Name:', d);
  console.log('Base Name:', b);
}

rl.question('Enter file path: ', (path) => {
pathmod(filePath);
  rl.close();
});
```

Task 4.2js

```
Task_4_1.js   Task_4_2.js
Task_4 > Task_4_2.js > exists
const path = require('path');
const fs = require('fs').promises;
Complexity is 3 Everything is cool!
async function exists(file) {
  try {
   await fs.access(file);
    console.log(`File "${file}" exists.`);
  } catch (error) {
    console.error(`Error: File "${file}" does not exist.`);
  }
}
const filepath = path.join(__dirname, 'randomname.js');
 exists(filepath);

PROBLEMS 3   OUTPUT   DEBUG CONSOLE   TERMINAL

D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_4>node Task_4_2.js
Error: File "D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_4\randomname.js" does not exist.

D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_4>
```

```
Task_4_1.js   Task_4_2.js
Task_4 > Task_4_2.js > filepath
const path = require('path');
const fs = require('fs').promises;
Complexity is 3 Everything is cool!
async function exists(file) {
  try {
   await fs.access(file);
    console.log(`File "${file}" exists.`);
  } catch (error) {
    console.error(`Error: File "${file}" does not exist.`);
  }
}
const filepath = path.join(__dirname, 'Task_4_1.js');
 exists(filepath);

PROBLEMS 2   OUTPUT   DEBUG CONSOLE   TERMINAL

D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_4>node Task_4_2.js
File "D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_4\Task_4_1.js" exists.

D:\Clg 3nd Year\5th SEMESTER BTech IT 21IT145\FSWD\Week_4\Task_4>
```

Learning Outcome:

From this practical, I learnt about various concepts of NodeJS which helped me to understand the basics of NodeJS and also practically performed it.

- Demonstrate the use of JavaScript to fulfill the essentials of front-end development To back-end development.
- Apply a deep knowledge of MVC(ModelViewController) architecture,making the development process easier and faster using open-source technologies.