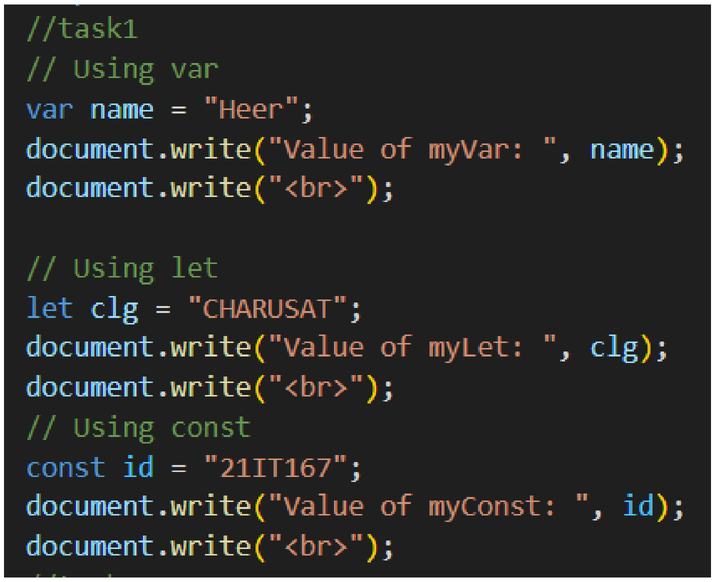
Practical-1
Task-1

Aim: Variables and Data Types.

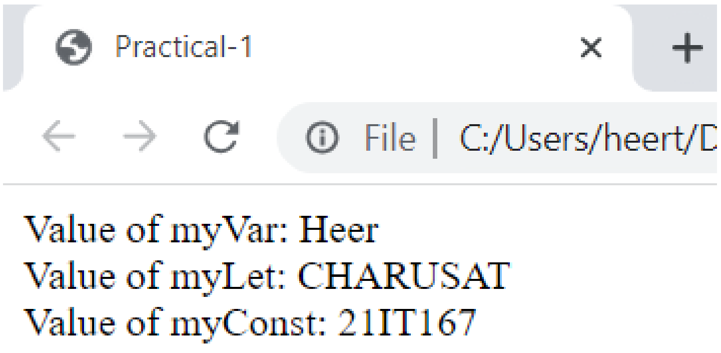
Description: Declare a variable using var, let, and const. Assign different data types to each variable and print their values.

Source Code:



```
//task1
// Using var
var name = "Heer";
document.write("Value of myVar: ", name);
document.write("<br>");

// Using let
let clg = "CHARUSAT";
document.write("Value of myLet: ", clg);
document.write("<br>");
// Using const
const id = "21IT167";
document.write("Value of myConst: ", id);
document.write("<br>");
```

Output:



```
Value of myVar: Heer
Value of myLet: CHARUSAT
Value of myConst: 21IT167
```

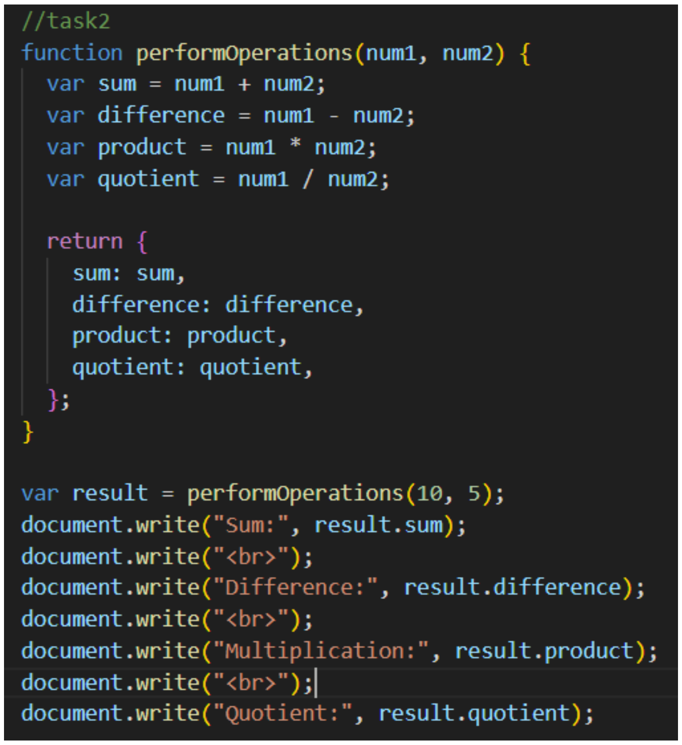
Learning Outcome: From this task, I have learned to declare a variable using var, let, and const.

Task-2

Aim: Operators and Expressions.

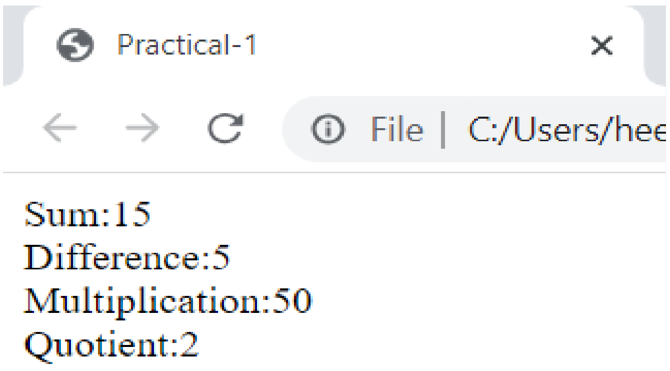
Description: Write a function that takes two numbers as arguments and returns their sum, difference, product, and quotient using arithmetic operators.

Source Code:



```
//task2
function performOperations(num1, num2) {
  var sum = num1 + num2;
  var difference = num1 - num2;
  var product = num1 * num2;
  var quotient = num1 / num2;

  return {
    sum: sum,
    difference: difference,
    product: product,
    quotient: quotient,
  };
}

var result = performOperations(10, 5);
document.write("Sum:", result.sum);
document.write("<br>");
document.write("Difference:", result.difference);
document.write("<br>");
document.write("Multiplication:", result.product);
document.write("<br>");
document.write("Quotient:", result.quotient);
```

Output:



```
Sum:15
Difference:5
Multiplication:50
Quotient:2
```

Learning Outcome: From this task, I have learned the concept of operators and expressions.

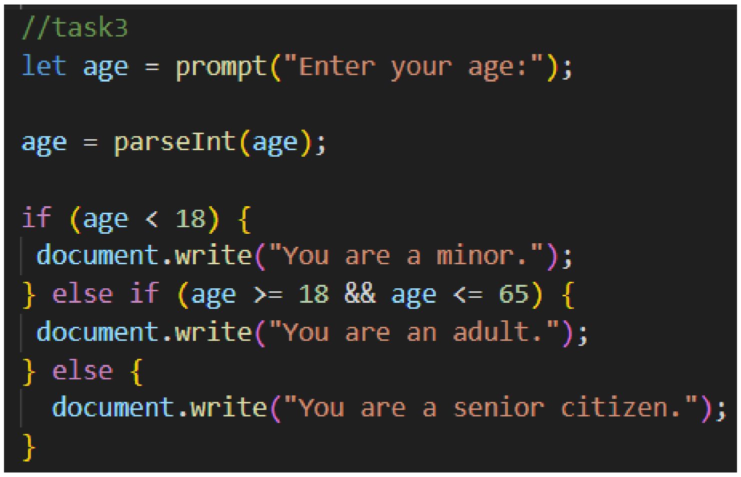
Task-3

Aim: Control Flow.

Description: Write a program that prompts the user to enter their age. Based on their age, display different messages:

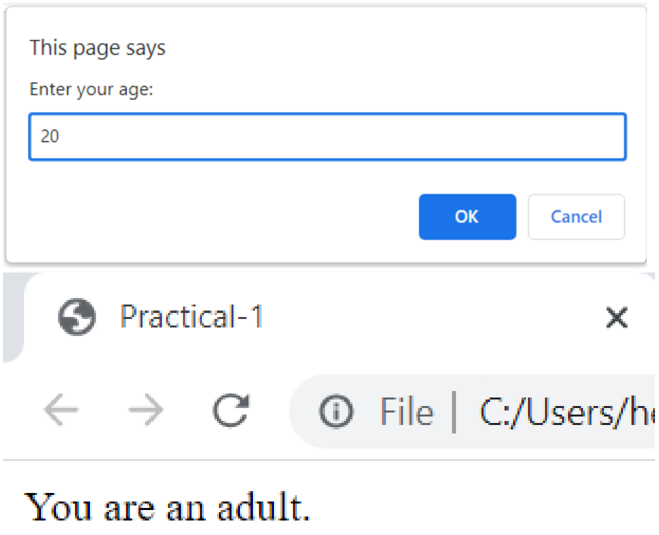
○ If the age is less than 18, display "You are a minor."
○ If the age is between 18 and 65, display "You are an adult."
○ If the age is 65 or older, display "You are a senior citizen."

Source Code:



```
//task3
let age = prompt("Enter your age:");

age = parseInt(age);

if (age < 18) {
 document.write("You are a minor.");
} else if (age >= 18 && age <= 65) {
 document.write("You are an adult.");
} else {
  document.write("You are a senior citizen.");
}
```

Output:



This page says

Enter your age:

20

You are an adult.

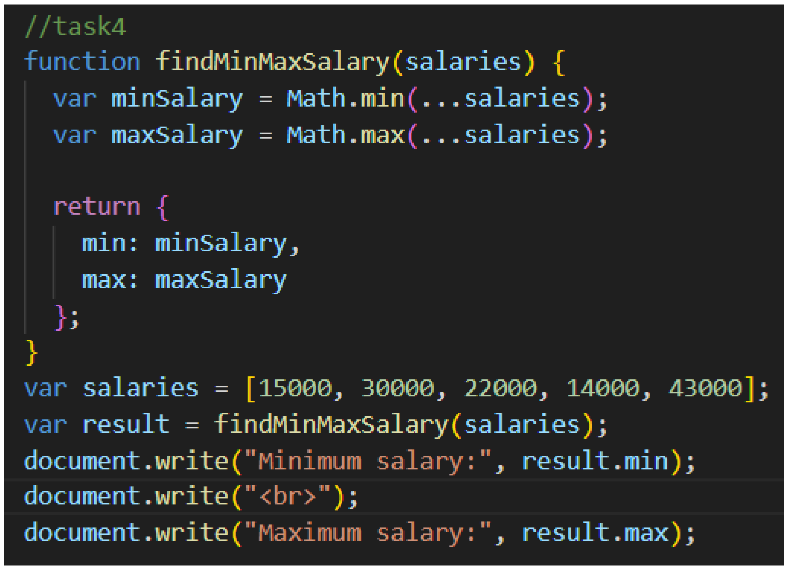
Learning Outcome: From this task, I have learned the concept of control flow.

Task-4

Aim: Functions.

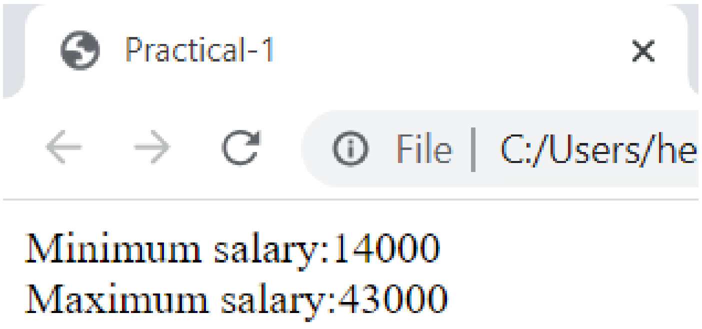
Description: Write a function that takes an array of salary as an argument and returns the min/max salary in the array.

Source Code:



```
//task4
function findMinMaxSalary(salaries) {
  var minSalary = Math.min(...salaries);
  var maxSalary = Math.max(...salaries);

  return {
    min: minSalary,
    max: maxSalary
  };
}
var salaries = [15000, 30000, 22000, 14000, 43000];
var result = findMinMaxSalary(salaries);
document.write("Minimum salary:", result.min);
document.write("<br>");
document.write("Maximum salary:", result.max);
```

Output:



```
Minimum salary:14000
Maximum salary:43000
```

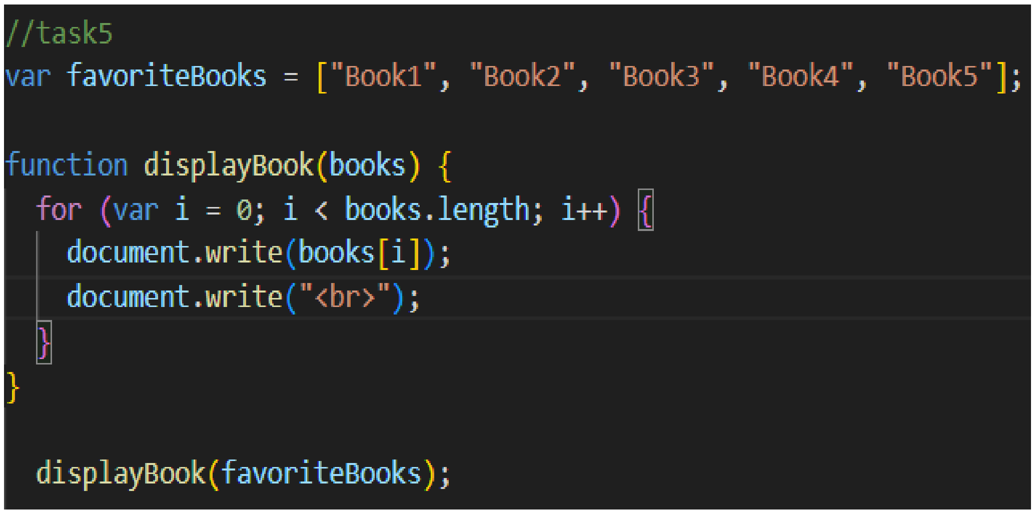
Learning Outcome: From this task, I have learned the concept of functions.

Task-5

Aim: Arrays and Objects.

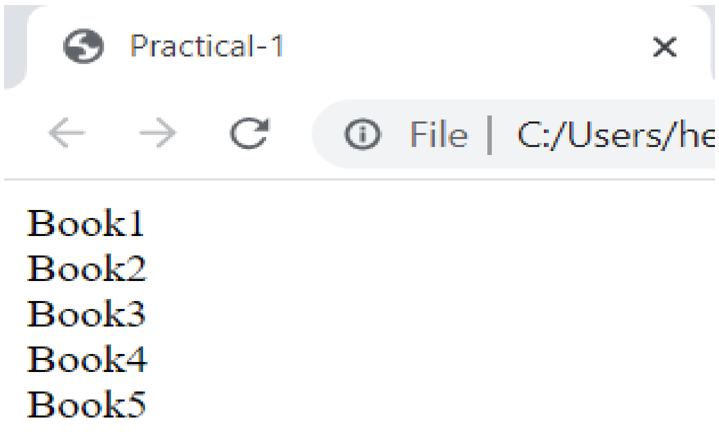
Description: Create an array of your favourite books. Write a function that takes the array as an argument and displays each book title on a separate line.

Source Code:



```
//task5
var favoriteBooks = ["Book1", "Book2", "Book3", "Book4", "Book5"];

function displayBook(books) {
  for (var i = 0; i < books.length; i++) {
    document.write(books[i]);
    document.write("<br>");
  }
}

  displayBook(favoriteBooks);
```

Output:



```
Book1
Book2
Book3
Book4
Book5
```

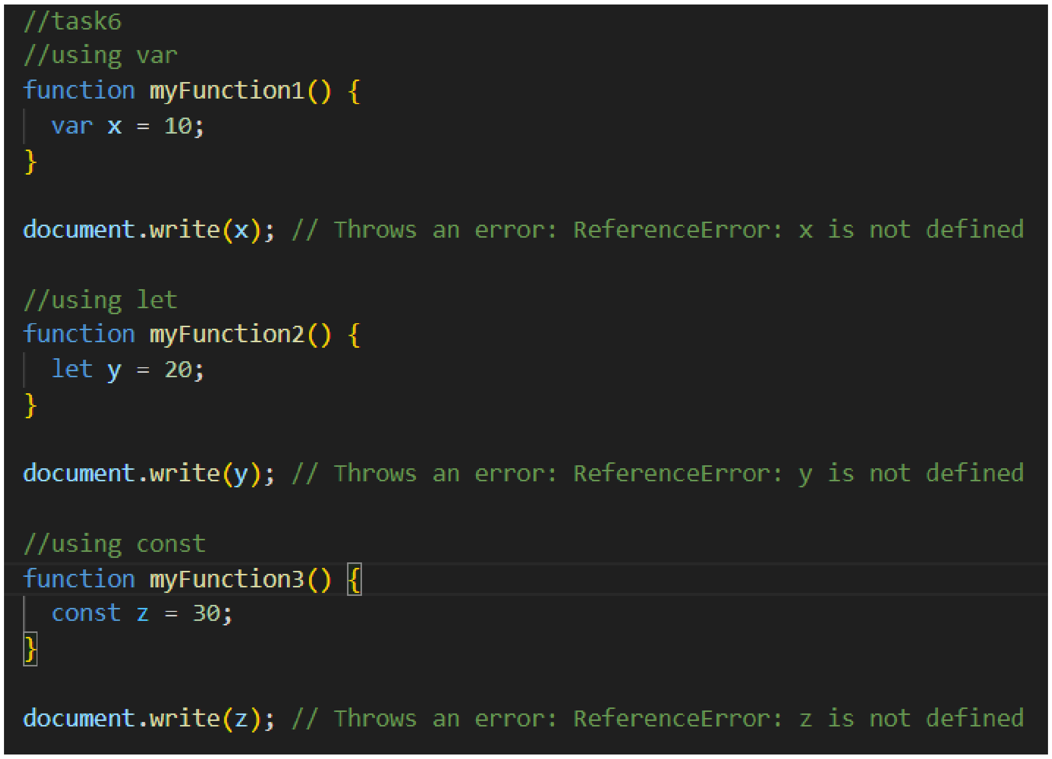
Learning Outcome: From this task, I have learned the use of arrays and object.

Task-6

Aim: Scope and Hoisting.

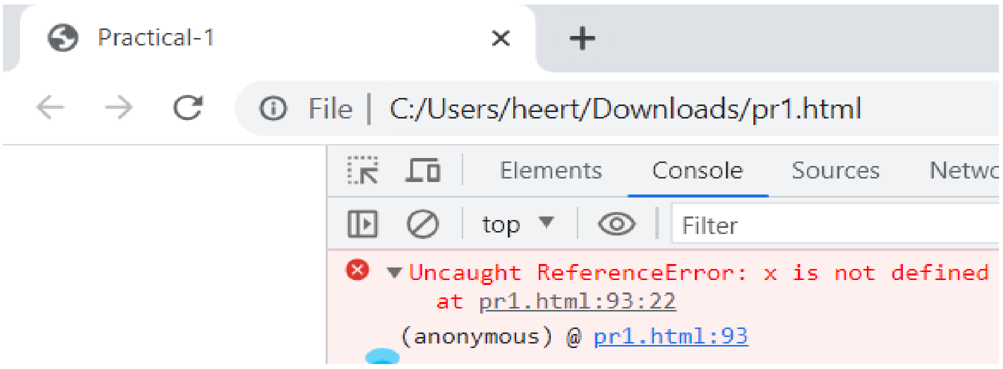
Description: Declare a variable inside a function and try to access it outside the function. Observe the scope behaviour and explain the results. [var vs let vs const]

Source Code:



```
//task6
//using var
function myFunction1() {
  var x = 10;
}

document.write(x); // Throws an error: ReferenceError: x is not defined

//using let
function myFunction2() {
  let y = 20;
}

document.write(y); // Throws an error: ReferenceError: y is not defined

//using const
function myFunction3() {
  const z = 30;
}

document.write(z); // Throws an error: ReferenceError: z is not defined
```

Output:



```
Uncaught ReferenceError: x is not defined
    at pr1.html:93:22
  (anonymous) @ pr1.html:93
```

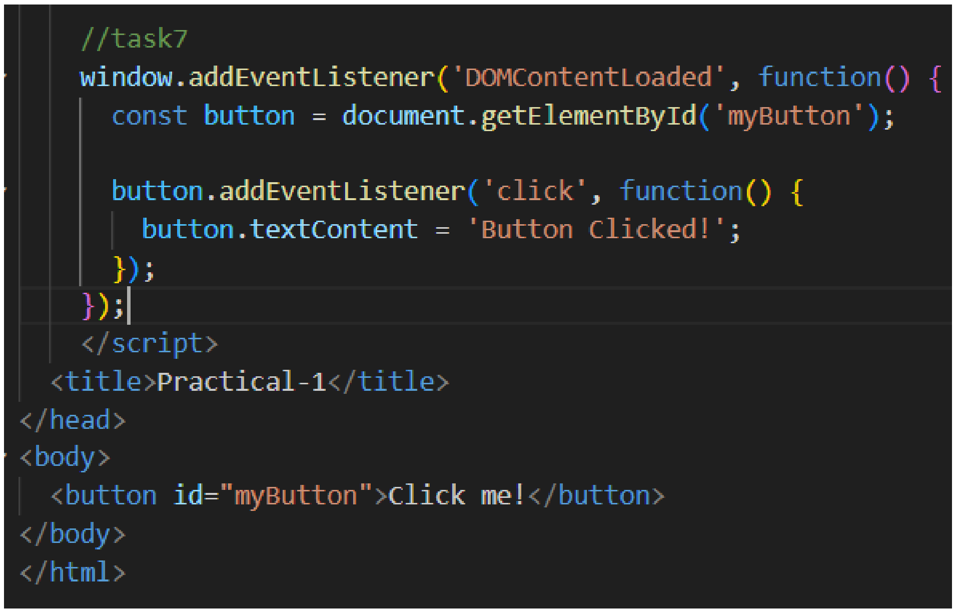
Learning Outcome: From this task, I have learned the concept of scope hoisting.

Task-7

Aim: DOM Manipulation.

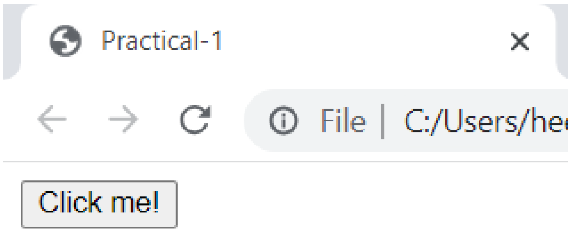
Description: Create an HTML page with a button. Write JavaScript code that adds an event listener to the button and changes its text when clicked.\

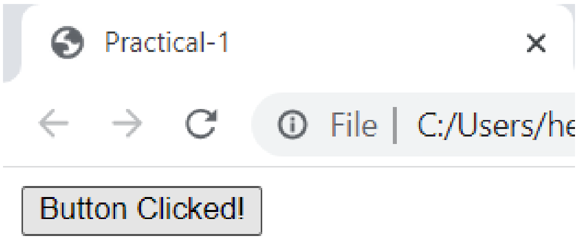
Source Code:



```
    //task7
    window.addEventListener('DOMContentLoaded', function() {
      const button = document.getElementById('myButton');

      button.addEventListener('click', function() {
        button.textContent = 'Button Clicked!';
      });
    });
    </script>
  <title>Practical-1</title>
</head>
<body>
  <button id="myButton">Click me!</button>
</body>
</html>
```

Output:





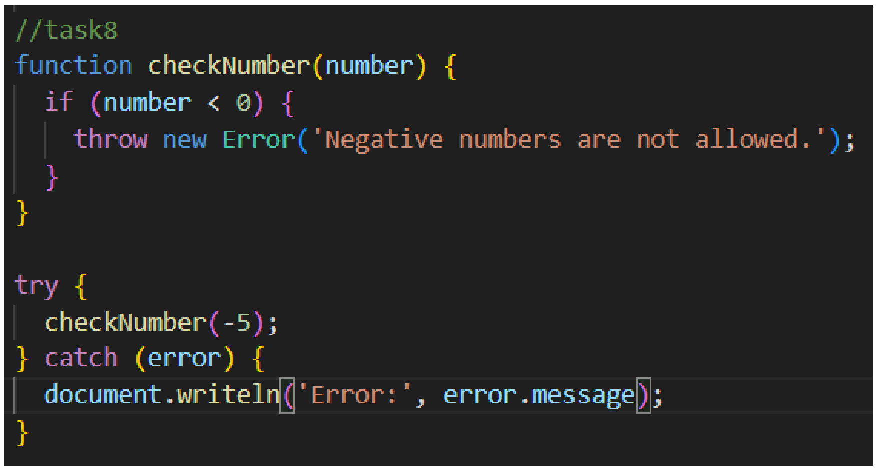
Learning Outcome: From this task, I have learned how to add event listener to the button.

Task-8

Aim: Error Handling.

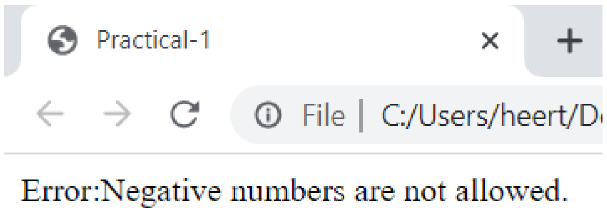
Description: Write a function that takes a number as an argument and throws an error if the number is negative. Handle the error and display a custom error message.

Source Code:



```
//task8
function checkNumber(number) {
  if (number < 0) {
    throw new Error('Negative numbers are not allowed.');
  }
}

try {
  checkNumber(-5);
} catch (error) {
  document.writeln('Error:', error.message);
}
```

Output:



```
Error:Negative numbers are not allowed.
```

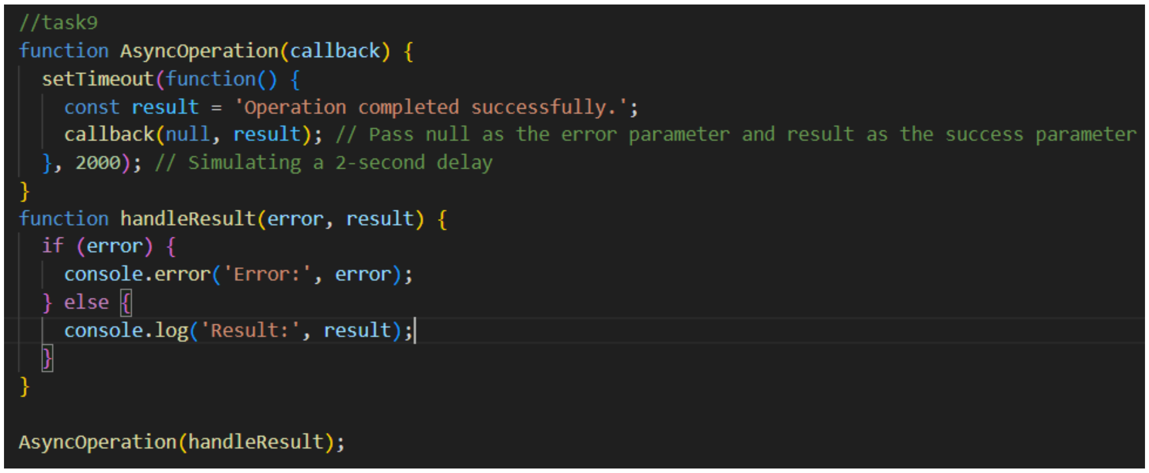
Learning Outcome: From this task, I have learned to handle any given error and display a custom error message.

Task-9

Aim: Asynchronous JavaScript.

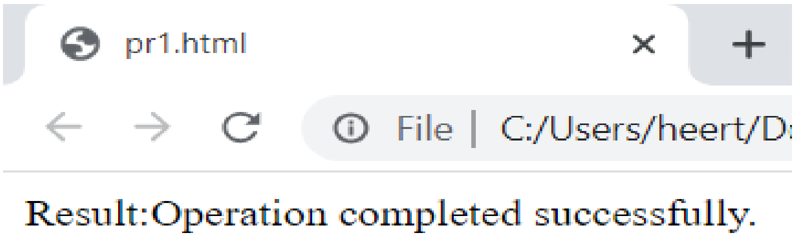
Description: Write a function that uses setTimeout to simulate an asynchronous operation. Use a callback function to handle the result.

Source Code:



```
//task9
function AsyncOperation(callback) {
  setTimeout(function() {
    const result = 'Operation completed successfully.';
    callback(null, result); // Pass null as the error parameter and result as the success parameter
  }, 2000); // Simulating a 2-second delay
}
function handleResult(error, result) {
  if (error) {
    console.error('Error:', error);
  } else {
    console.log('Result:', result);
  }
}

AsyncOperation(handleResult);
```

Output:



Result:Operation completed successfully.

Learning Outcome: From this task, I have learned to simulate an asynchronous operation and used a callback function to handle the result.

Practical - 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.

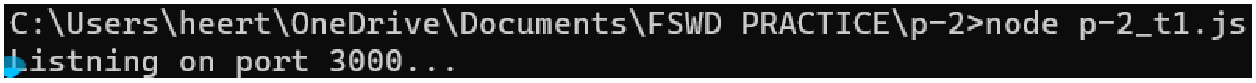
Description:

Setting up a basic HTTP server involves creating a Node.js application that listens for incoming HTTP requests and responds with a simple message. In this task, we will use Node.js and the built-in http module to create an HTTP server.

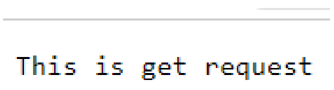
Source Code:

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

const httpserver = http.createServer(function(req,res){

   if(req.method == 'GET')

   {

      res.end("This is get request");

   }

});

httpserver.listen(3000,()=>{

   console.log("Listning on port 3000...");

})
```

Output:



```
C:\Users\heert\OneDrive\Documents\FSWD PRACTICE\p-2>node p-2_t1.js
Listning on port 3000...
```



```
This is get request
```

Learning Outcome: Understanding the basics of setting up an HTTP server using Node.js.

Familiarity with the http module in Node.js for handling HTTP requests and responses.
Ability to create a server object and listen for incoming HTTP requests.

Task-2

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

Description:

Created a Node.js application that handles different types of HTTP requests and provides appropriate responses based on the request method, content type, and desired status code.

Source Code:

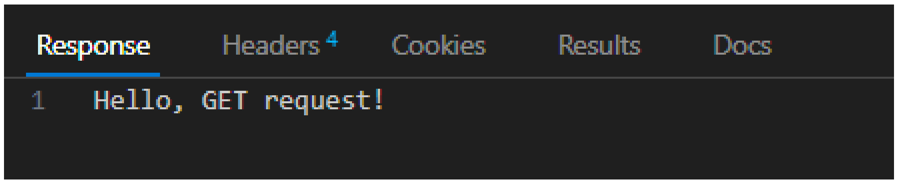
HTTP Methods:

```
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');
  }
});
```

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

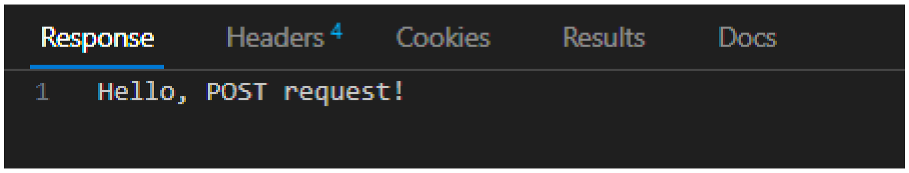
Output:

GET:



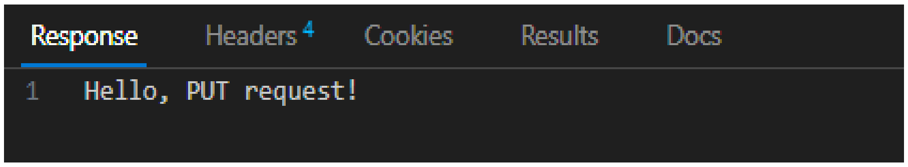
```
Hello, GET request!
```

POST:



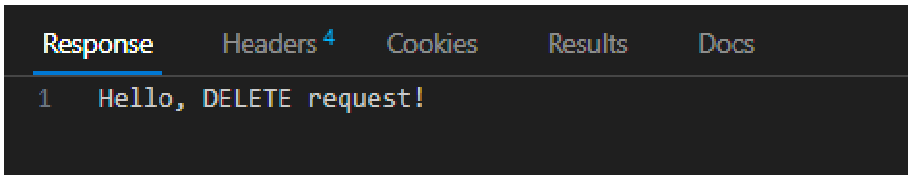
```
Hello, POST request!
```

PUT:



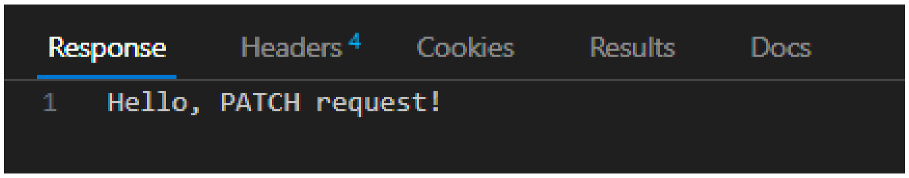
```
Hello, PUT request!
```

DELETE:



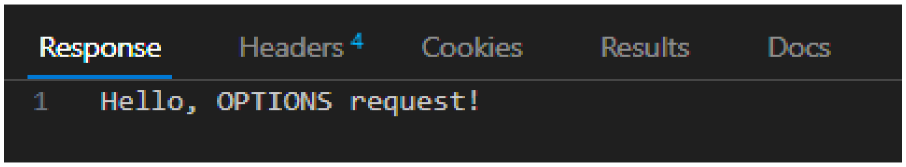
```
Hello, DELETE request!
```

PATCH:



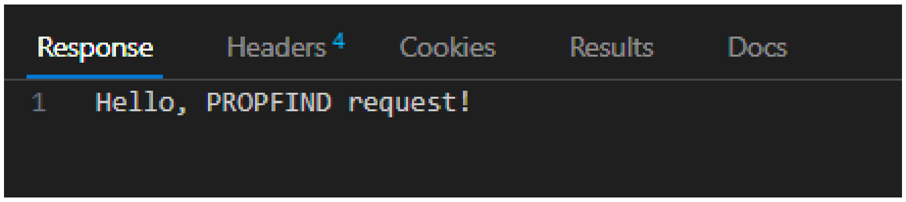
```
Hello, PATCH request!
```

OPTIONS:



```
Hello, OPTIONS request!
```

PROPFIND:



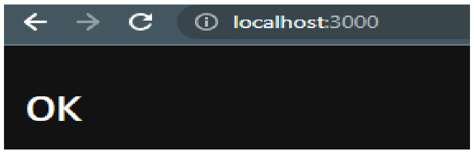
```
Hello, PROPFIND request!
```

Source Code :

Status Code:

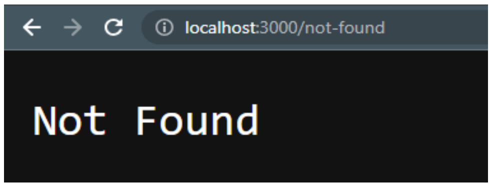
```
if (req.url === '/') {
    res.statusCode = 200;
    res.end('OK');
  } else if (req.url === '/not-found') {
    res.statusCode = 404;
    res.end('Not Found');
  } else if (req.url === '/server-error') {
    res.statusCode = 500;
    res.end('Internal Server Error');
  } else {
    res.statusCode = 400;
    res.end('Bad Request');
  }
});
server.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

Output:


localhost:3000

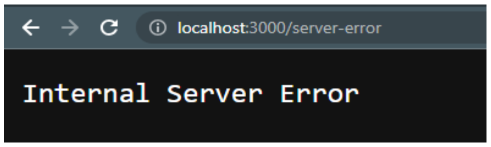
```
OK
```

404


localhost:3000/not-found

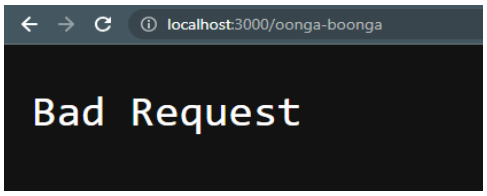
```
Not Found
```

500


localhost:3000/server-error

```
Internal Server Error
```

400


localhost:3000/oonga-boonga

```
Bad Request
```

Learning Outcome: Learned to implement logic to handle different combinations of HTTP methods, content types, and status codes in our application.

Task-3

Aim: Test it using browser, CLI and REST Client.

Description:

In this task, you will test your HTTP server using three different methods: using a web browser, the command-line interface (CLI), and a REST client. Each method provides a unique way to send HTTP requests and receive responses from your server.

Source Code:

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

http.createServer((req,res)=>{
   const readStream = fs.createReadStream('./static/index.htm')
// Assume we have a static folder having 3 static files 1)example.json 2)example.png
3)index.htm
   res.writeHead(200,{'content-type':'text/html'})
   readStream.pipe(res)
}).listen(3000);
```

Output:

Learning Outcome: By testing your server using different tools, you will ensure that your server functions as expected in different environments and provides the desired responses.

Task-4

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

Description:

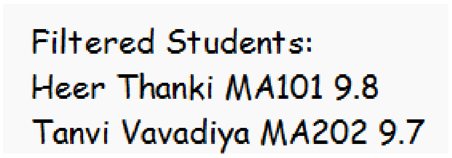
In this task, you will read a file called "student-data.txt" and search for students whose names contain the substring 'MA' and have a CGPA (Cumulative Grade Point Average) greater than 7.

You will write a Node.js program to read the file, parse its contents, and filter the student data based on the specified conditions. This task will involve reading files, string manipulation, and data filtering.

Source Code:

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

fs.readFile('student-data.txt', 'utf8', (err, data) => {
  if (err) {
    console.error('Error reading file:', err);
    return;
  }

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

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

Output:



```
Filtered Students:
Heer Thanki MA101 9.8
Tanvi Vavadiya MA202 9.7
```

Learning Outcome: After this task, I learned to read and process data from a file.

Task-5

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

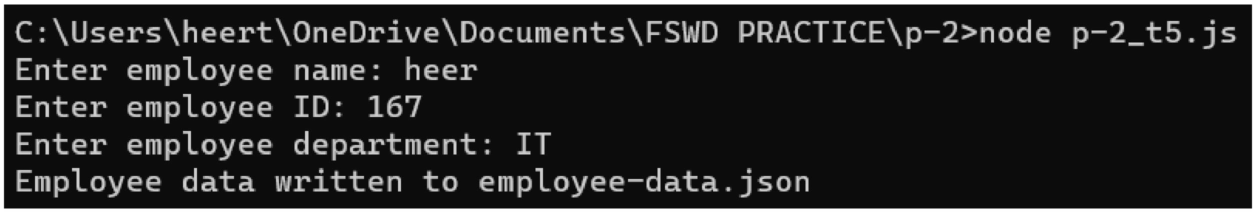
Description:

In this task, you will create a Node.js program that prompts the user to enter employee information and then writes the data to a file called 'employee-data.json'. You will use the built-in `fs` module in Node.js to perform file operations and the `readline` module to capture user input.

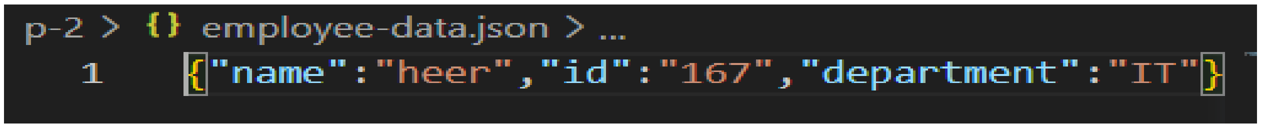
Source Code:

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

const fs = require('fs');

// Create readline interface

const rl = readline.createInterface({

  input: process.stdin,

  output: process.stdout

});

// Prompt user for employee information

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

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

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

      // Create employee object

      const employee = {

        name: name,

        id: id,

        department: department

      };
      // Convert employee object to JSON string
      const employeeData = JSON.stringify(employee);
      // Write data to file
      fs.writeFile('employee-data.json', employeeData, (err) => {
        if (err) throw err;
        console.log('Employee data written to employee-data.json');
        rl.close();
      });
    });
  });
});
```

Output:



```
C:\Users\heert\OneDrive\Documents\FSWD PRACTICE\p-2>node p-2_t5.js
Enter employee name: heer
Enter employee ID: 167
Enter employee department: IT
Employee data written to employee-data.json
```



```
p-2 > {} employee-data.json > ...
{"name":"heer","id":"167","department":"IT"}
```

Learning Outcome:

By completing this task, gained practical experience in handling user input, writing data to a file, and working with JSON.

Task-6

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

Description:

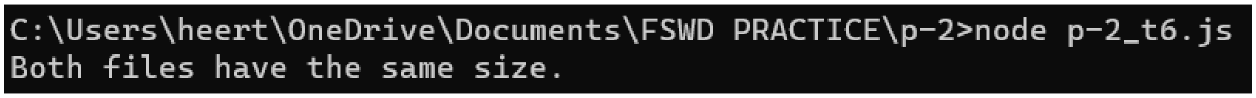
In this task, you will compare two files and determine which file is larger in terms of size.

Source Code:

```
const fs = require('fs');
// Function to compare two files
function compareFiles(file1Path, file2Path) {
  try {
    // Read the contents of both files
    const file1Content = fs.readFileSync(file1Path, 'utf8');
    const file2Content = fs.readFileSync(file2Path, 'utf8');
    // Get the size (in bytes) of both files
    const file1Size = fs.statSync(file1Path).size;
    const file2Size = fs.statSync(file2Path).size;
    // Determine the larger file
    let largerFile;
    if (file1Size > file2Size) {
      largerFile = file1Path;
    } else if (file2Size > file1Size) {
      largerFile = file2Path;
    } else {
      console.log('Both files have the same size.');
      return;
    }
    console.log(`The larger file is: ${largerFile}`);

    // Split file contents into lines
    const file1Lines = file1Content.split('\n');
    const file2Lines = file2Content.split('\n');

    // Compare lines between the two files
    const differentLines = [];
    file1Lines.forEach((line, index) => {
      if (line !== file2Lines[index]) {
        differentLines.push(index + 1);
      }
    });

    if (differentLines.length > 0) {
      console.log('Lines that are different between the files:');
      differentLines.forEach((line) => {
        console.log(`Line ${line}`);
      });
    } else {
      console.log('Both files have the same content.');
    }
```

```
  } catch (err) {
    console.error('Error comparing files:', err);
  }
}
// Compare files
compareFiles('./file1.txt', './file2.txt');
```

Output:



```
C:\Users\heert\OneDrive\Documents\FSWD PRACTICE\p-2>node p-2_t6.js
Both files have the same size.
```

Learning Outcome: After this task, I gained experience in comparing and analyzing the differences between two files.

Task-7

Aim: Create File Backup and Restore Utility.

Description:

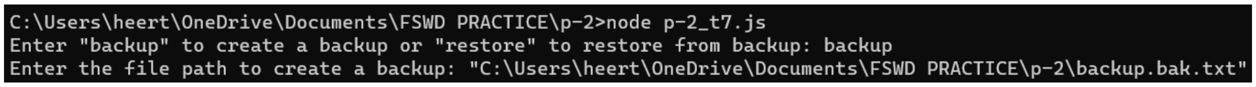
In this task, you will create a file backup and restore utility using a Node.js application. The utility will allow users to create backups of files and restore them when needed. When creating a backup, the utility will copy the contents of the original file to a backup file with a specified name or timestamp. When restoring a backup, the utility will replace the original file with the contents of the backup file, effectively restoring the file to a previous state.

Source Code:

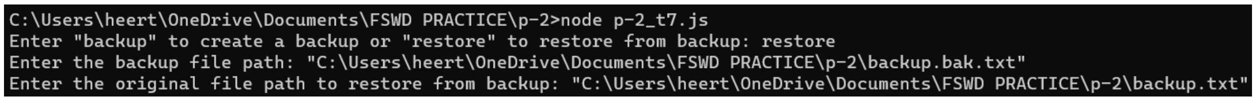
```
const fs = require('fs');
const readline = require('readline');
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});
// Function to create a backup of a file
function createBackup(filePath) {
  const backupFilePath = `${filePath}.bak`;
  fs.copyFile(filePath, 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 restoreFromBackup(backupFilePath, originalFilePath) {
  fs.copyFile(backupFilePath, originalFilePath, (err) => {
    if (err) {
      console.error('Error restoring from backup:', err);
    } else {
      console.log(`File restored: ${originalFilePath}`);
    }
    rl.close();
  });
```

```
}

// Prompt user for choice: backup or restore

rl.question('Enter "backup" to create a backup or "restore" to restore from backup: ', (choice) =>
{

  if (choice === 'backup') {

    rl.question('Enter the file path to create a backup: ', (filePath) => {

      createBackup(filePath);

    });

  } else if (choice === 'restore') {

    rl.question('Enter the backup file path: ', (backupFilePath) => {

      rl.question('Enter the original file path to restore from backup: ', (originalFilePath) => {

        restoreFromBackup(backupFilePath, originalFilePath);

      });

    });

  } else {

    console.log('Invalid choice.');

    rl.close();

  }

});
```

Output:



```
C:\Users\heert\OneDrive\Documents\FSWD PRACTICE\p-2>node p-2_t7.js
Enter "backup" to create a backup or "restore" to restore from backup: backup
Enter the file path to create a backup: "C:\Users\heert\OneDrive\Documents\FSWD PRACTICE\p-2\backup.bak.txt"
```



```
C:\Users\heert\OneDrive\Documents\FSWD PRACTICE\p-2>node p-2_t7.js
Enter "backup" to create a backup or "restore" to restore from backup: restore
Enter the backup file path: "C:\Users\heert\OneDrive\Documents\FSWD PRACTICE\p-2\backup.bak.txt"
Enter the original file path to restore from backup: "C:\Users\heert\OneDrive\Documents\FSWD PRACTICE\p-2\backup.txt"
```

Learning Outcome: Learned to build a file backup and restore utility, which can be useful in various scenarios where data preservation and recovery

Task-8

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

Description:

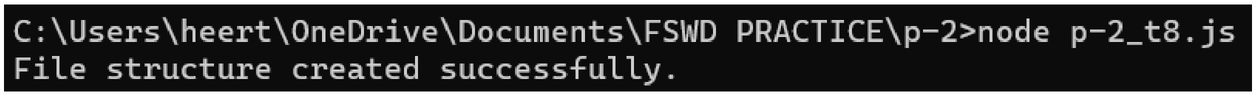
In this task, you will create a file and folder structure based on the information provided in a JSON file. The JSON file will contain a hierarchical structure representing directories and files. You will write a Node.js program to read the JSON file, traverse the structure, and create the corresponding directories and files in the file system.

Source Code:

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

// Function to create directories recursively

function createDirectories(parentPath, directories) {

  directories.forEach((directory) => {

    const dirPath = `${parentPath}/${directory.name}`;

    // Create the directory

    fs.mkdirSync(dirPath);


    // Check if there are files in the current directory

    if (directory.files && directory.files.length > 0) {

      directory.files.forEach((file) => {

        const filePath = `${dirPath}/${file.name}`;

        // Create the file

        fs.writeFileSync(filePath, file.content);

      });
    }
    // Check if there are subdirectories
    if (directory.directories && directory.directories.length > 0) {
      createDirectories(dirPath, directory.directories);
    }
  });
}
// Read the JSON file
fs.readFile('file-structure.json', 'utf8', (err, data) => {
  if (err) {
    console.error('Error reading file structure:', err);
    return;
  }
  try {
    // Parse the JSON data
    const fileStructure = JSON.parse(data);

    // Create the file structure recursively
    createDirectories('.', fileStructure);
    console.log('File structure created successfully.');
  } catch (err) {
    console.error('Error parsing JSON file structure:', err);
  }
```

});

Output:



```
C:\Users\heert\OneDrive\Documents\FSWD PRACTICE\p-2>node p-2_t8.js
File structure created successfully.
```

Learning Outcome: By completing this task, I gained practical experience in creating file and folder structures based on JSON data.

Task-9

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

Description:

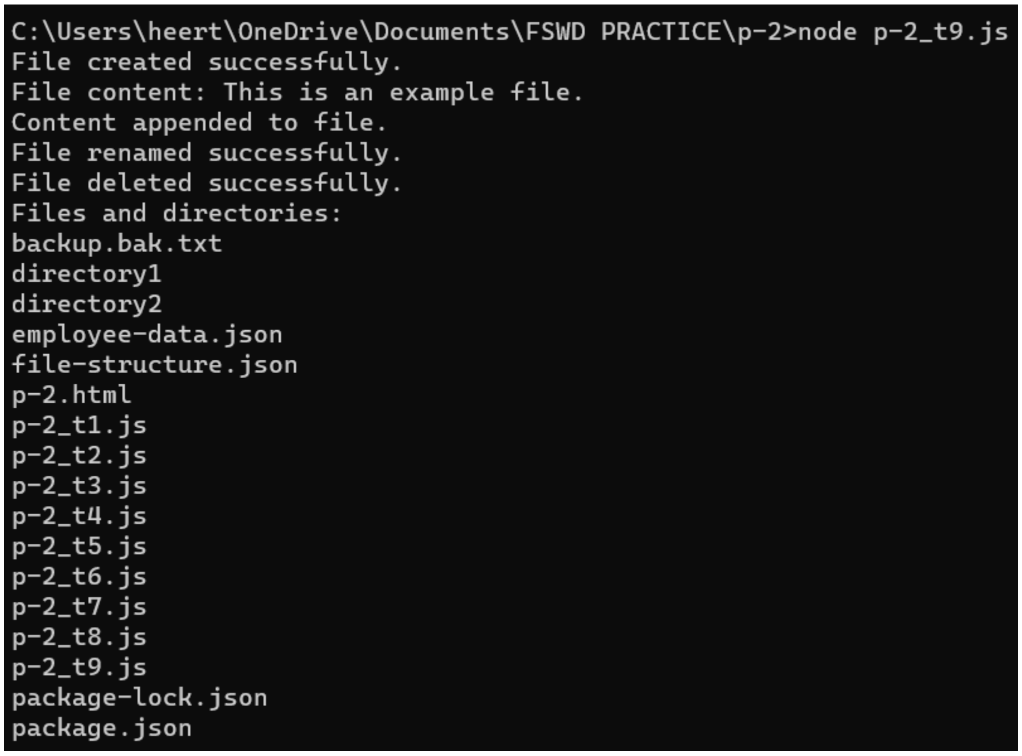
In this task, you will experiment with various file operations in a Node.js application. You will create, read, append, delete, rename files, and list files/directories in a specified directory. This task will involve using the built-in `fs` module in Node.js to perform these file operations.

Source Code:

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

// Create a file

fs.writeFile('example.txt', 'This is an example file.', (err) => {

  if (err) {

    console.error('Error creating file:', err);

    return;

  }

  console.log('File created successfully.');

  // Read the file

  fs.readFile('example.txt', 'utf8', (err, data) => {

    if (err) {

      console.error('Error reading file:', err);
      return;
    }
    console.log('File content:', data);
    // Append to the file
    fs.appendFile('example.txt', '\nThis is appended content.', (err) => {
      if (err) {
        console.error('Error appending to file:', err);
        return;
      }
      console.log('Content appended to file.');
      // Delete the file
      fs.unlink('example.txt', (err) => {
        if (err) {
          console.error('Error deleting file:', err);
          return;
        }
        console.log('File deleted successfully.');
        // Rename a file
        fs.rename('example.txt', 'renamed.txt', (err) => {
          if (err) {
            console.error('Error renaming file:', err);
            return;
          }
```

```
console.log('File renamed successfully.');
// List files and directories
fs.readdir('.', (err, files) => {
  if (err) {
    console.error('Error listing files/directories:', err);
    return;
  }
  console.log('Files and directories:');
  files.forEach((file) => {
    console.log(file);
  }); });});});});});});
```

Output:



```
C:\Users\heert\OneDrive\Documents\FSWD PRACTICE\p-2>node p-2_t9.js
File created successfully.
File content: This is an example file.
Content appended to file.
File renamed successfully.
File deleted successfully.
Files and directories:
backup.bak.txt
directory1
directory2
employee-data.json
file-structure.json
p-2.html
p-2_t1.js
p-2_t2.js
p-2_t3.js
p-2_t4.js
p-2_t5.js
p-2_t6.js
p-2_t7.js
p-2_t8.js
p-2_t9.js
package-lock.json
package.json
```

Learning Outcome: From this practical , I experienced working with files, manipulating their contents, and managing the file system using Node.js.

Practical - 3
Task-1

Aim: OS Information:

- Write a program that uses the os module to display the current user's username, home directory, and operating system platform.
- Create a function that utilizes the os module to display the total system memory, free memory, and the percentage of free memory available.

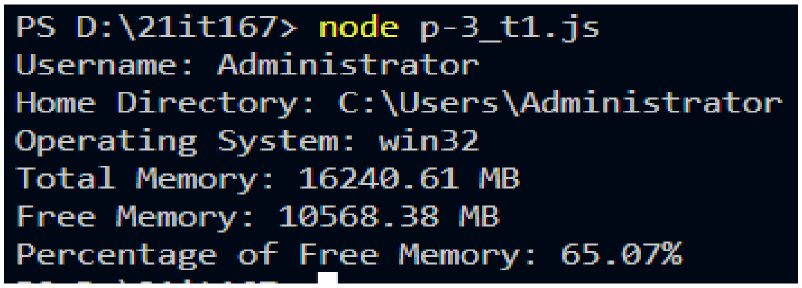
Description:

The OS module provides information about the computer's operating system.

Source Code:

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

// Get current user's information
const currentUser = os.userInfo();
console.log('Username:', currentUser.username);
console.log('Home Directory:', currentUser.homedir);
console.log('Operating System:', os.platform());

// Function to get system memory information
function getMemoryInfo() {
  const totalMemory = os.totalmem();
  const freeMemory = os.freemem();
  const percentageFree = (freeMemory / totalMemory) * 100;

  return {
    totalMemory: `${(totalMemory / (1024 * 1024)).toFixed(2)} MB`,
    freeMemory: `${(freeMemory / (1024 * 1024)).toFixed(2)} MB`,
    percentageFree: `${percentageFree.toFixed(2)}%`,
  };
}

// Get system memory information and display it
const memoryInfo = getMemoryInfo();
console.log('Total Memory:', memoryInfo.totalMemory);
console.log('Free Memory:', memoryInfo.freeMemory);
console.log('Percentage of Free Memory:', memoryInfo.percentageFree);
```

Output:



```
PS D:\21it167> node p-3_t1.js
Username: Administrator
Home Directory: C:\Users\Administrator
Operating System: win32
Total Memory: 16240.61 MB
Free Memory: 10568.38 MB
Percentage of Free Memory: 65.07%
```

Learning Outcome:

CO2: Apply a deep knowledge of MVC(ModelViewController) architecture, making the development process easier and faster using open-source technologies.

Task-2

Aim: Experiment with chalk, upper-case any other External Modules.

Description:

Chalk module in Node.js is the third-party module that is used for styling the format of text and allows us to create our own themes in the node.js project.

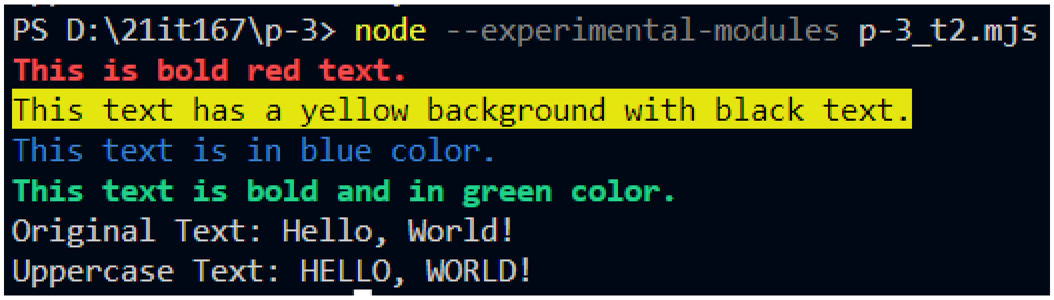
toUpperCase() function converts a string to all uppercase letters in Node. js. It doesn't change anything else about the original string and doesn't take any parameters.

Source Code:

```
import chalk from 'chalk';

import upperCase from 'upper-case';

// Using chalk for colorful terminal output
console.log(chalk.bold.red('This is bold red text.'));
console.log(chalk.bgYellow.black('This text has a yellow background with black text.'));
console.log(chalk.blue('This text is in blue color.'));
console.log(chalk.green.bold('This text is bold and in green color.'));

// Using upper-case module to convert text to uppercase
const originalText = 'Hello, World!';
const upperCaseText = upperCase.upperCase(originalText);
```

console.log(`Original Text: ${originalText}`);
console.log(`Uppercase Text: ${upperCaseText}`);

Output:



```
PS D:\21it167\p-3> node --experimental-modules p-3_t2.mjs
This is bold red text.
This text has a yellow background with black text.
This text is in blue color.
This text is bold and in green color.
Original Text: Hello, World!
Uppercase Text: HELLO, WORLD!
```

Learning Outcome:

CO2: Apply a deep knowledge of MVC(ModelViewController) architecture, making the development process easier and faster using open-source technologies.

Task-3

Aim: Create your own custom module and import/export it to the main module.

Description:

Creating custom modules in Node.js allows you to encapsulate specific functionality into separate files that can be reused across your application or shared with others and to export the functions or properties you want to expose using the exports object, and require the module using require in the files where you want to use it.

Source Code:

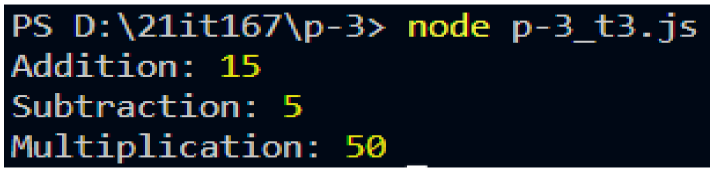
// main.js:

```
const mathOperations = require('./mathOperations');

const resultAdd = mathOperations.add(10, 5);
console.log('Addition:', resultAdd);

const resultSubtract = mathOperations.subtract(10, 5);
console.log('Subtraction:', resultSubtract);

const resultMultiply = mathOperations.multiply(10, 5);
console.log('Multiplication:', resultMultiply);
```

// mathOperations.js:

```
function add(a, b) {
    return a + b;
  }

  function subtract(a, b) {
    return a - b;
  }

  function multiply(a, b) {
    return a * b;
  }

  // Export the functions to make them accessible to other modules
  module.exports = {
    add,
    subtract,
    multiply,
  };
```

Output:



```
PS D:\21it167\p-3> node p-3_t3.js
Addition: 15
Subtraction: 5
Multiplication: 50
```

Learning Outcome:

CO2: Apply a deep knowledge of MVC(ModelViewController) architecture, making the development process easier and faster using open-source technologies.

Practical - 4
Task-1

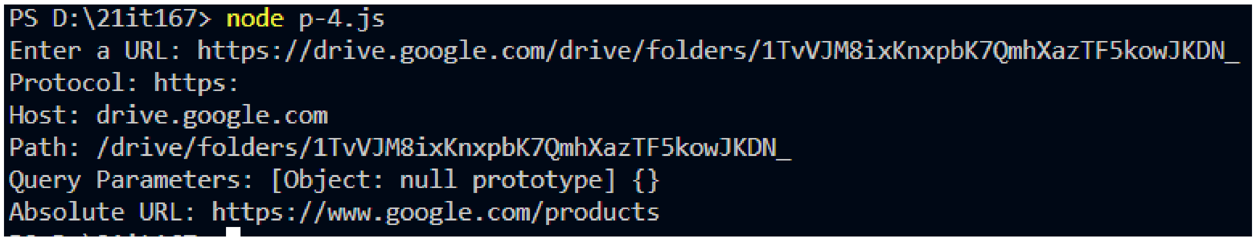
Aim: URL Parsing and Manipulation:
● Write a program that accepts a URL as user input and uses the URL module to parse it.
Display the protocol, host, path, and query parameters separately.
● Implement a function that takes a base URL and a relative path as input, and uses the URL module to resolve and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

Description:
In Node.js, the URL module is a built-in module that provides utilities for parsing and formatting URLs (Uniform Resource Locators). It allows developers to work with URLs in a convenient and efficient manner.

Source Code:

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

const readline = require('readline');

const rl = readline.createInterface({

  input: process.stdin,

  output: process.stdout

});

// Function to parse and display URL components

function parseURL(inputURL) {

  const parsedURL = url.parse(inputURL, true);

  console.log('Protocol:', parsedURL.protocol);

  console.log('Host:', parsedURL.host);

  console.log('Path:', parsedURL.pathname);

  console.log('Query Parameters:', parsedURL.query);
}

// Function to resolve and display absolute URL
function resolveURL(baseURL, relativePath) {
  const absoluteURL = url.resolve(baseURL, relativePath);
  console.log('Absolute URL:', absoluteURL);
}
// Get user input
rl.question('Enter a URL: ', (userURL) => {
  // Parse and display URL components
  parseURL(userURL);
  // Example usage of resolveURL function
  const baseURL = 'https://www.google.com/';
  const relativePath = '/products';
  resolveURL(baseURL, relativePath);
  rl.close();
});
```

Output:



```
PS D:\21it167> node p-4.js
Enter a URL: https://drive.google.com/drive/folders/1TvVJM8ixKnxpbK7QmhXazTF5kowJKDN_
Protocol: https:
Host: drive.google.com
Path: /drive/folders/1TvVJM8ixKnxpbK7QmhXazTF5kowJKDN_
Query Parameters: [Object: null prototype] {}
Absolute URL: https://www.google.com/products
```

Learning Outcome:

CO2: Apply a deep knowledge of MVC(ModelViewController) architecture, making the development process easier and faster using open-source technologies.

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 query string module. The program should
display the extracted key-value pairs as output.

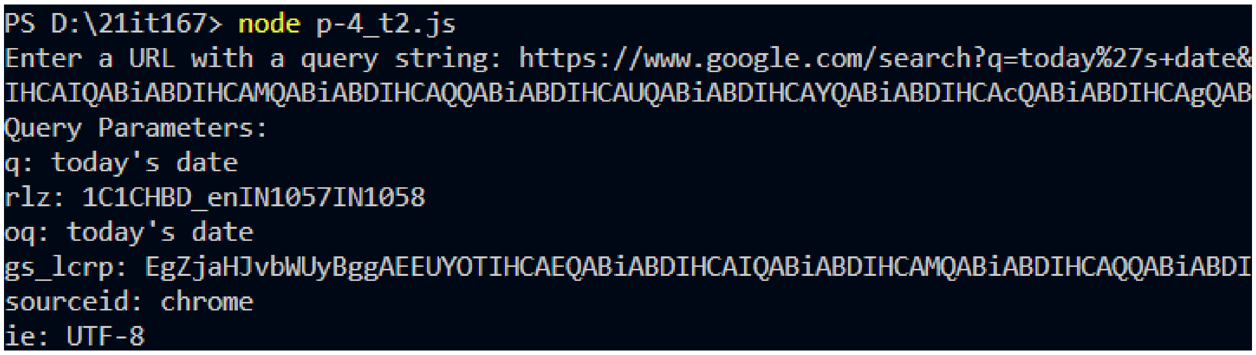
Description:

In Node.js, the term "Query String Operation" typically refers to the manipulation and parsing of query strings. A query string is a part of a URL that follows the question mark (?) and contains key-value pairs separated by ampersands (&).
Node.js provides a built-in module called querystring, which offers various functions to work with query strings.

Source Code:

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

const querystring = require('querystring');

const readline = require('readline');

const rl = readline.createInterface({

  input: process.stdin,

  output: process.stdout

});

// Function to extract and display key-value pairs from a query string

function extractQueryParams(inputURL) {

  const parsedURL = url.parse(inputURL);

  const queryParams = querystring.parse(parsedURL.query);
  console.log('Query Parameters:');
  for (const key in queryParams) {
    console.log(key + ':', queryParams[key]);
  }
}
// Get user input
rl.question('Enter a URL with a query string: ', (userURL) => {
  extractQueryParams(userURL);
  rl.close();
});
```

Output:



```
PS D:\21it167> node p-4_t2.js
Enter a URL with a query string: https://www.google.com/search?q=today%27s+date&
IHCAIQABiABDIHCAMQABiABDIHCAQQABiABDIHCAUQABiABDIHCAYQABiABDIHCAcQABiABDIHCAgQAB
Query Parameters:
q: today's date
rlz: 1C1CHBD_enIN1057IN1058
oq: today's date
gs_lcrp: EgZjaHJvbWUyBggAEEUYOTIHCAEQABiABDIHCAIQABiABDIHCAMQABiABDIHCAQQABiABDI
sourceid: chrome
ie: UTF-8
```

Learning Outcome:

CO2: Apply a deep knowledge of MVC(ModelViewController) architecture, making the development process easier and faster using open-source technologies.

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.

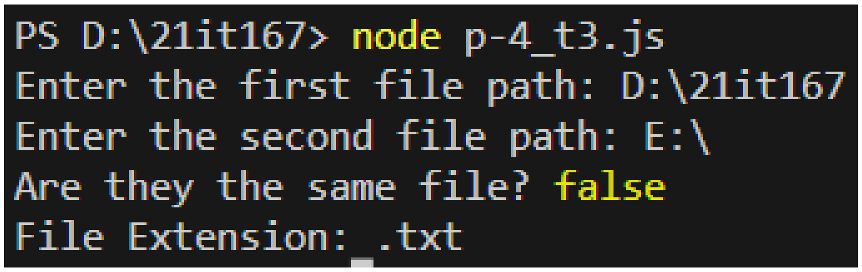
Description:

In Node.js, a "path operation" typically refers to manipulating and interacting with file paths and directories within the file system. The core module path in Node.js provides various methods to work with file paths, making it easier to handle file and directory paths in a platform-independent way.

Source Code:

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

const readline = require('readline');

const fs = require('fs');

const rl = readline.createInterface({

  input: process.stdin,

  output: process.stdout

});

// Function to check if two file paths refer to the same file

function checkSameFile(path1, path2) {

  const absolutePath1 = path.resolve(path1);

  const absolutePath2 = path.resolve(path2);

  if (fs.existsSync(absolutePath1) && fs.existsSync(absolutePath2)) {

    const isSameFile = fs.statSync(absolutePath1).ino === fs.statSync(absolutePath2).ino;

    console.log('Are they the same file?', isSameFile);

  } else {

    console.log('One or both file paths are invalid.');

  }
}
// Function to extract and display the file extension from a file path
function extractFileExtension(filePath) {
  const fileExtension = path.extname(filePath);
  console.log('File Extension:', fileExtension);
}
// Get user input
rl.question('Enter the first file path: ', (path1) => {
  rl.question('Enter the second file path: ', (path2) => {
    // Check if the file paths refer to the same file
    checkSameFile(path1, path2);
    // Example usage of extractFileExtension function
    const exampleFilePath = '/path/to/example/file.txt';
    extractFileExtension(exampleFilePath);
    rl.close();
  });
});
```

Output:



```
PS D:\21it167> node p-4_t3.js
Enter the first file path: D:\21it167
Enter the second file path: E:\
Are they the same file? false
File Extension: .txt
```

Learning Outcome:

CO2: Apply a deep knowledge of MVC(ModelViewController) architecture, making the development process easier and faster using open-source technologies.

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:

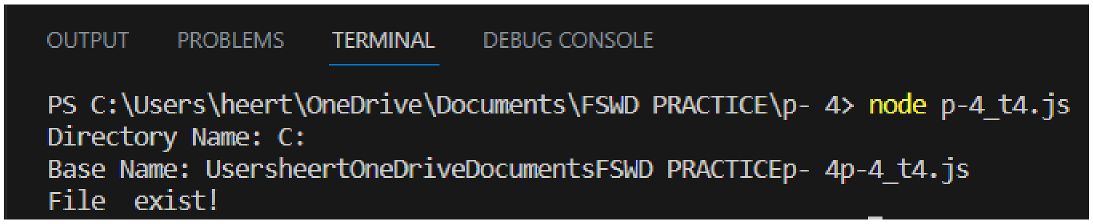
In Node.js, file paths and file operations play a crucial role in handling files and directories within your application. Understanding file paths and how to perform file operations is essential for reading, writing, and manipulating data stored in files.

Source Code:

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

const fs = require('fs');

// Function to extract directory name and base name

function extractFileInfo(filePath) {

  const dirName = path.dirname(filePath);

  const baseName = path.basename(filePath);

  return {dirName, baseName};

}

// Function to check if the file exists

function checkFileExists(filePath) {

  return new Promise((resolve) => {

    fs.stat(filePath, (err, stats) => {
      if (err) {
        resolve(false); // File does not exist
      } else {
        resolve(stats.isFile());
      } });
  });
}
const filePathToProcess = 'C:\Users\heert\OneDrive\Documents\FSWD PRACTICE\p- 4\p-
4_t4.js';
// Extract directory name and base name
const fileInfo = extractFileInfo(filePathToProcess);
console.log('Directory Name:', fileInfo.dirName);
console.log('Base Name:', fileInfo.baseName);
// Check if the file exists
checkFileExists(filePathToProcess)
  .then((fileExists) => {
    if (fileExists) {
      console.log('File  exist!');
    } else {
      console.error('File does not exist.');
    }
  })
  .catch((err) => {
    console.error('Error:', err.message);
```

});

Output:



```
PS C:\Users\heert\OneDrive\Documents\FSWD PRACTICE\p- 4> node p-4_t4.js
Directory Name: C:
Base Name: UsersheertOneDriveDocumentsFSWD PRACTICEp- 4p-4_t4.js
File  exist!
```

Learning Outcome:

CO2: Apply a deep knowledge of MVC(ModelViewController) architecture, making the development process easier and faster using open-source technologies.

Practical – 5&6
Task-1

Aim: Develop an ExpressJS API to perform CRUD operations using In Memory Database & Test it.

Description:

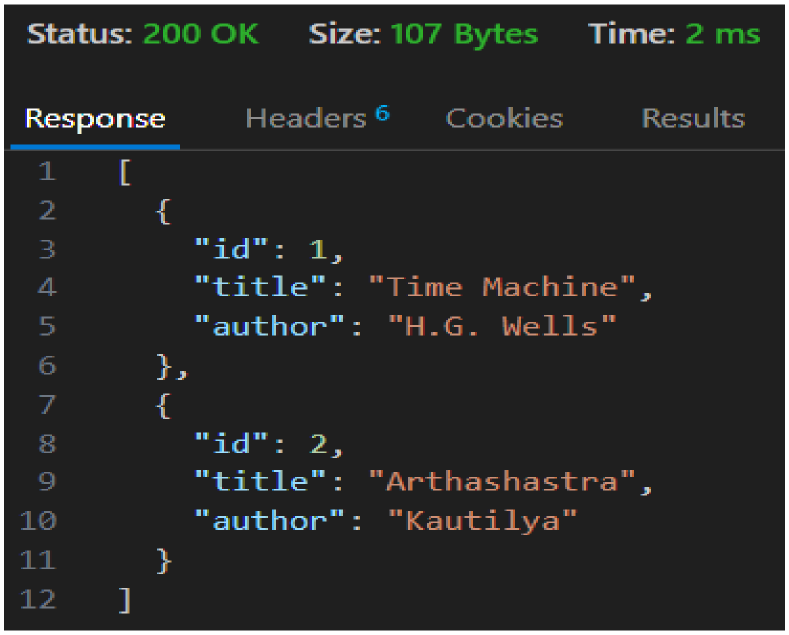
This code sets up a basic Express.js server to handle a CRUD API for managing items. It includes endpoints to retrieve all items or specific items by ID, create new items, update existing items, and delete items. Data is stored in-memory, and the server runs on port 3000, utilizing a body-parser for JSON data handling.

Source Code:

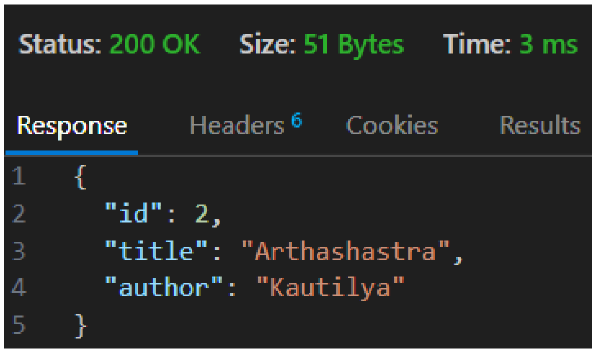
```
const express = require('express');

const bodyParser = require('body-parser');

const app = express();

const port = 3001;

app.use(bodyParser.json());

let books = [];

app.post('/api/books', (req, res) => {

  const {title, author} = req.body;

  if (!title || !author) {

    return res.status(400).json({ error: 'Title and author are required' });

  }

  else{

  const id = books.length + 1;

  const newBook = { id, title, author };

  books.push(newBook);
  return res.status(201).json(newBook);
  }
});
app.get('/api/books', (req, res) => {
  return res.json(books);
});
app.get('/api/books/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const book = books.find((item) => item.id === id);
  if (!book) {
    return res.status(404).json({ error: 'Book not found' });
  }
  return res.json(book);
});
app.put('/api/books/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const {title, author } = req.body;
  const bookIndex = books.findIndex((item) => item.id === id);
  if (bookIndex === -1) {
    return res.status(404).json({ error: 'Book not found' });
  }
  books[bookIndex] = { ...books[bookIndex], title, author };
  return res.json(books[bookIndex]);
```

```
});
app.delete('/api/books/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const bookIndex = books.findIndex((item) => item.id === id);
  if (bookIndex === -1) {
    return res.status(404).json({ error: 'Book not found' });
  }
  books.splice(bookIndex, 1);
  return res.json({ message: 'Book deleted successfully' });
});
app.listen(port, () => {
  console.log(`Server running on http://localhost:${port}`);
});
```

Output:

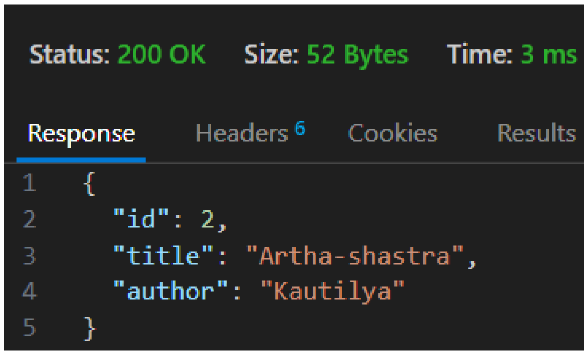


```
[
  {
    "id": 1,
    "title": "Time Machine",
    "author": "H.G. Wells"
  },
  {
    "id": 2,
    "title": "Arthashastra",
    "author": "Kautilya"
  }
]
```


Status: 200 OK Size: 51 Bytes Time: 3 ms

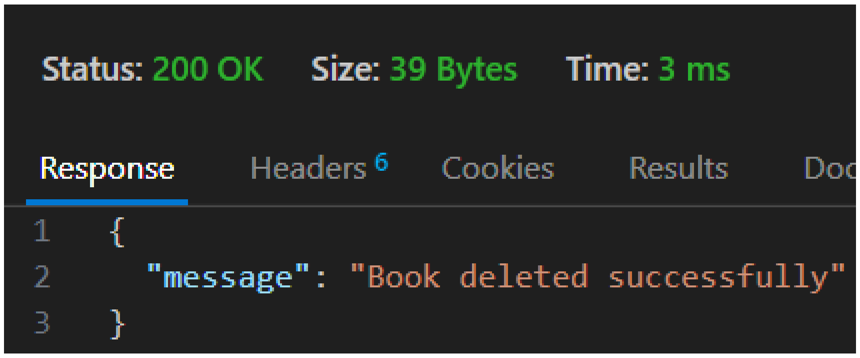
Response Headers 6 Cookies Results

```
{
  "id": 2,
  "title": "Arthashastra",
  "author": "Kautilya"
}
```


Status: 200 OK Size: 52 Bytes Time: 3 ms

Response Headers 6 Cookies Results

```
{
  "id": 2,
  "title": "Artha-shastra",
  "author": "Kautilya"
}
```


Status: 200 OK Size: 39 Bytes Time: 3 ms

Response Headers 6 Cookies Results Doc

```
{
  "message": "Book deleted successfully"
}
```

Task-2

Aim: Validate API-Form/Data at server-side using validator/joi module.

Description:

This code creates an Express.js server for a CRUD API to manage items. It utilizes the body-parser middleware for JSON data and the Joi library for data validation against a predefined schema. Endpoints handle GET, POST, PUT, and DELETE operations, validating data before updates. The server runs on port 3000, managing item data in-memory.

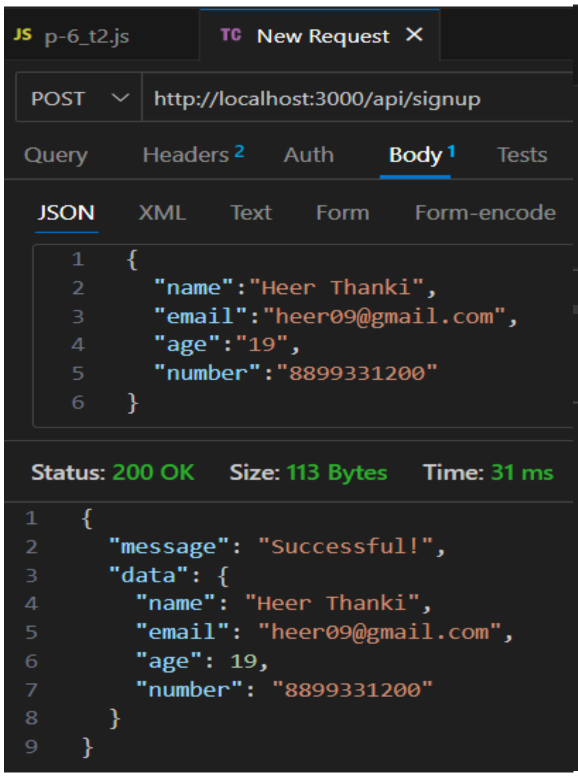
Source Code:

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

const joi = require('Joi');

const app = express();

app.use(express.json());

let users = [];
```

```
const validation = joi.object({
  name: joi.string().required(),
  email: joi.string()
  .email().pattern(/^[a-zA-Z].*\.com$/).required(),
  age: joi.number().integer().min(1).max(120).message('Age must be between 1 to
120').required(),
  number: joi.string()
    .pattern(/^[0-9]{10}$/)
    .message('Phone number must be a 10-digit number.')
    .required(),
});
app.post('/api/signup', (req, res) => {
  const { error, value } = validation.validate(req.body);
  if (error) {
    return res.status(404).json({ error: error.details[0].message });
  }
  users.push(value);
  res.json({ message: 'Successful!', data: value });
});
app.get('/api/users', (req, res) => {
  res.json(users);
});
```

```
const PORT = 3000;

app.listen(PORT, () => {

  console.log(`Server running on port ${PORT}`);

});
```

Output:

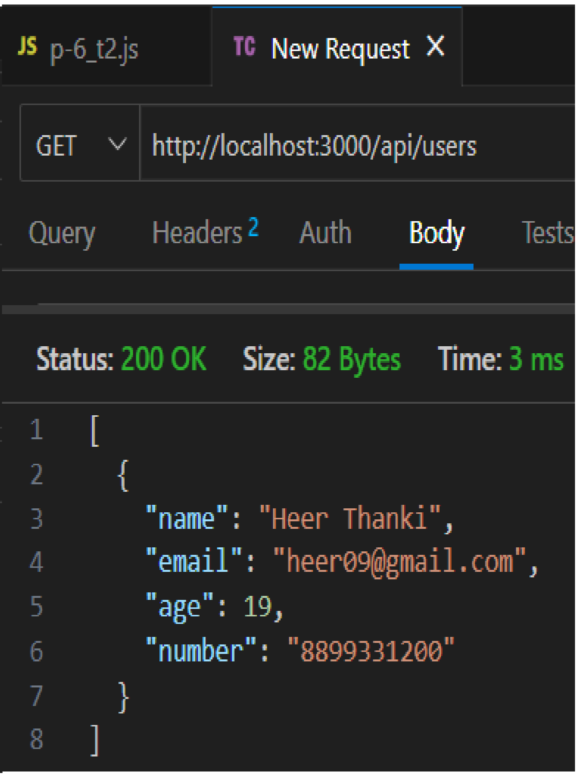


```
POST  http://localhost:3000/api/signup
```

```
{
  "name":"Heer Thanki",
  "email":"heer09@gmail.com",
  "age":"19",
  "number":"8899331200"
}
```

Status: 200 OK   Size: 113 Bytes   Time: 31 ms

```
{
  "message": "Successful!",
  "data": {
    "name": "Heer Thanki",
    "email": "heer09@gmail.com",
    "age": 19,
    "number": "8899331200"
  }
}
```



```
GET  http://localhost:3000/api/users
```

Status: 200 OK   Size: 82 Bytes   Time: 3 ms

```
[
  {
    "name": "Heer Thanki",
    "email": "heer09@gmail.com",
    "age": 19,
    "number": "8899331200"
  }
]
```

Task-3

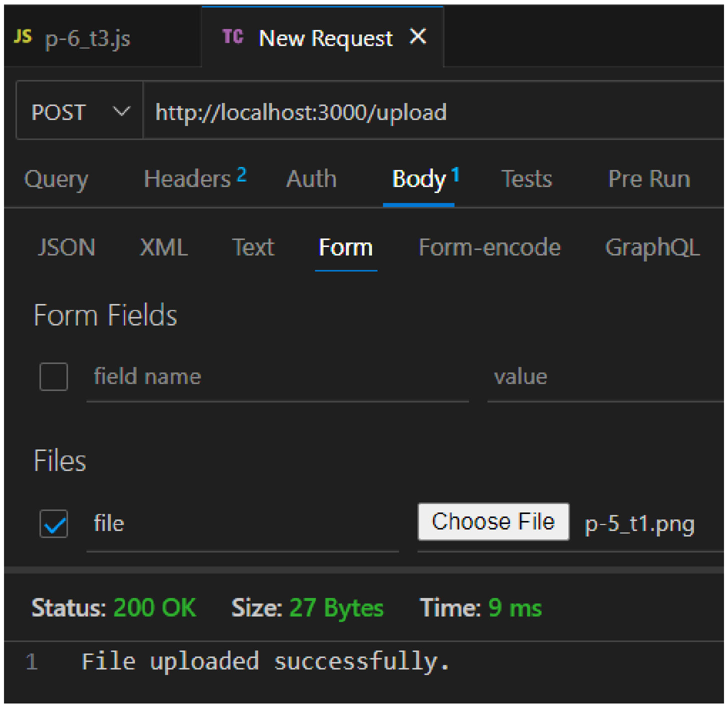
Aim: Create API to upload file & Test it.

Description:
This code defines an Express.js route for handling file uploads. When a POST request is made to '/upload', it checks for an uploaded file, moves it to a designated directory, and responds with success or failure messages. The server starts listening on a specified port.

Source Code:

```
const express = require('express');
const fileUpload = require('express-fileupload');
const path = require('path');
const app = express();
const port = 3000; // Change this to your desired port
app.use(fileUpload());
app.post('/upload', (req, res) => {
  const file = req.files && req.files.file;

  if (!file) {
    return res.status(400).send('No file uploaded.');
  }
  const uploadPath = path.join(__dirname, 'uploads', file.name);
  file.mv(uploadPath, (err) => {
    if (err) {
      console.error(err);
      return res.status(500).send('File upload failed.');
    }
    res.send('File uploaded successfully.');
  });
});
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

Output:



http://localhost:3000/upload

file: p-5_t1.png

Status: 200 OK Size: 27 Bytes Time: 9 ms

```
File uploaded successfully.
```

Task-4

Aim: Create Contact us API to send email to predefined admin using node mailer module.

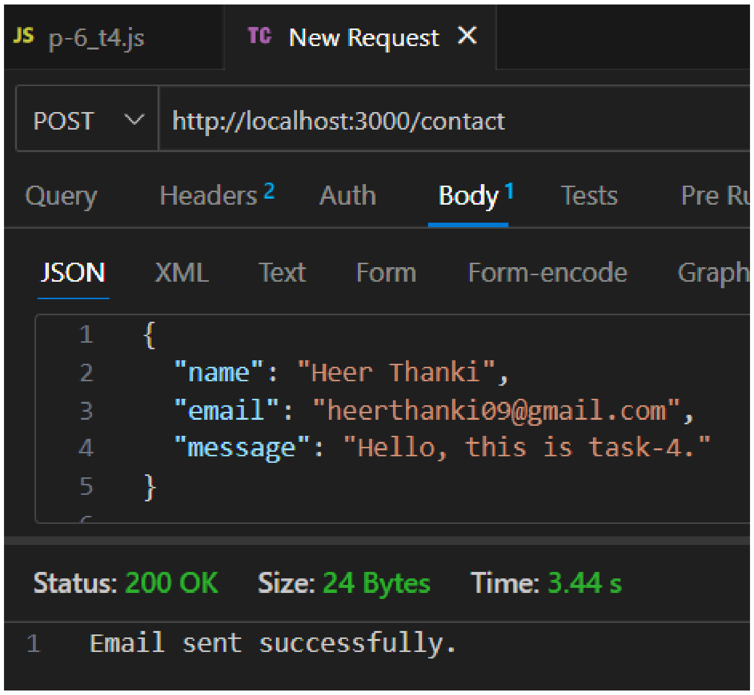
Description:

This code uses Express.js and Node mailer to set up a server for handling a contact form submission. When a POST request is made to '/contact', it validates the received data (name, email, and message), sends an email to an admin using the configured Gmail account, and responds with success or error messages. The server listens on the specified port.

Source Code:

```
const express = require('express');
const nodemailer = require('nodemailer');
const app = express();
const port = 3000
app.use(express.json());
const transporter = nodemailer.createTransport({
  service: 'Gmail',
  auth: {
    user: 'heerthanki09@gmail.com',
    pass: 'zjwfuwbyqjbrqhda'
  }
});
app.post('/contact', (req, res) => {
  const { name, email, message } = req.body;

  if (!name || !email || !message) {
    return res.status(400).send('All fields are required.');
  }

  const mailOptions = {
    from: 'heerthanki09@gmail.com', // Sender's email address
    to: '21it167@charusat.edu.in', // Admin's email address
    subject: 'FSWD-Practical_5&6',
    text: `Name: ${name}\nEmail: ${email}\nMessage: ${message}`
  };

  // Send the email
  transporter.sendMail(mailOptions, (error, info) => {
    if (error) {
      console.error('Error sending email:', error);
      return res.status(500).send('Error sending email.');
    }
    console.log('Email sent:', info.response);
    res.send('Email sent successfully.');
  });
});

// Start the server
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

Output:



```
POST http://localhost:3000/contact
```

```
{
  "name": "Heer Thanki",
  "email": "heerthanki09@gmail.com",
  "message": "Hello, this is task-4."
}
```

Status: 200 OK Size: 24 Bytes Time: 3.44 s

```
Email sent successfully.
```

Task-5

Aim: Create a program that uses the OS module to display all the environment variables on the system.

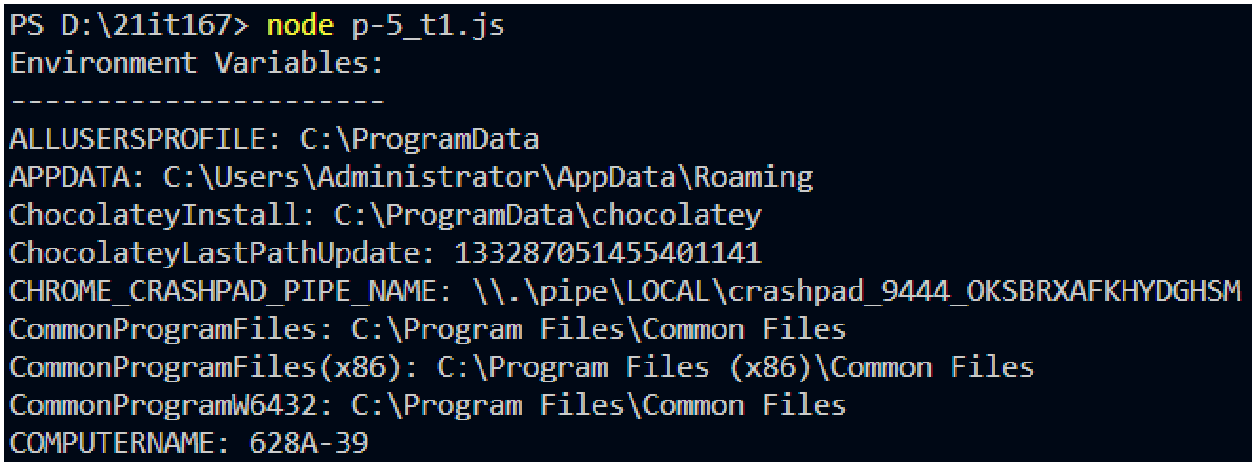
Description:

Create a Node.js program that utilizes the os module to access and display all the environment variables available on the system. The os module provides various methods to interact with the operating system, including retrieving environment variables.

Source Code:

```
function displayEnvironmentVariables() {
   console.log("Environment Variables:");
   console.log("----------------------");
   for (let key in process.env) {
     console.log(`${key}: ${process.env[key]} `);
   }
 }
 displayEnvironmentVariables();
```

Output:



```
PS D:\21it167> node p-5_t1.js
Environment Variables:
----------------------
ALLUSERSPROFILE: C:\ProgramData
APPDATA: C:\Users\Administrator\AppData\Roaming
ChocolateyInstall: C:\ProgramData\chocolatey
ChocolateyLastPathUpdate: 133287051455401141
CHROME_CRASHPAD_PIPE_NAME: \\.\pipe\LOCAL\crashpad_9444_OKSBRXAFKHYDGHSM
CommonProgramFiles: C:\Program Files\Common Files
CommonProgramFiles(x86): C:\Program Files (x86)\Common Files
CommonProgramW6432: C:\Program Files\Common Files
COMPUTERNAME: 628A-39
```

Task-6

Aim: Implement a function that takes an environment variable name as input and uses the process.env object to display its corresponding value.

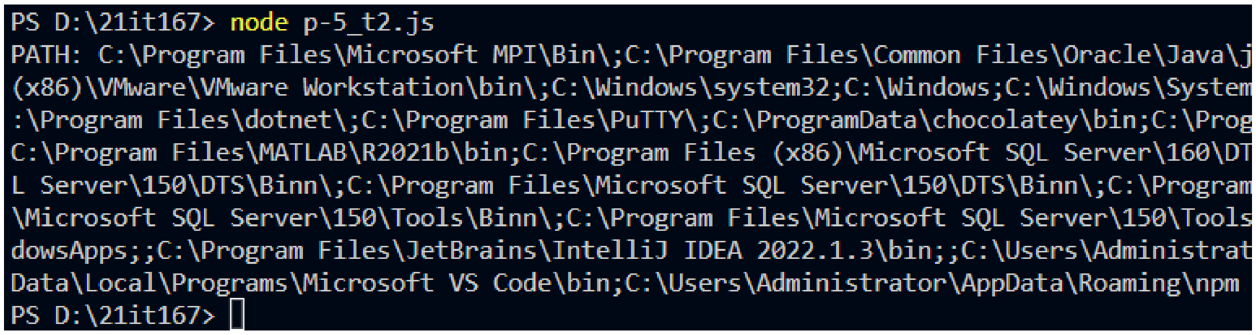
Description:

The aim is to create a function that takes the name of an environment variable as input and then uses the `process.env` object in Node.js to retrieve and display the corresponding value of that environment variable. This function will allow you to easily access environment variable values within your Node.js application.

Source Code:

```
function displayEnvironmentVariable(variableName) {
    if (process.env[variableName] !== undefined) {
      console.log(`${variableName}: ${process.env[variableName]}`);
    } else {
      console.log(`Environment variable "${variableName}" not found.`);
    }
  }
  const envVariableName = "PATH";
  displayEnvironmentVariable(envVariableName);
```

Output:



```
PS D:\21it167> node p-5_t2.js
PATH: C:\Program Files\Microsoft MPI\Bin\;C:\Program Files\Common Files\Oracle\Java\j
(x86)\VMware\VMware Workstation\bin\;C:\Windows\system32;C:\Windows;C:\Windows\System
:\Program Files\dotnet\;C:\Program Files\PuTTY\;C:\ProgramData\chocolatey\bin;C:\Prog
C:\Program Files\MATLAB\R2021b\bin;C:\Program Files (x86)\Microsoft SQL Server\160\DT
L Server\150\DTS\Binn\;C:\Program Files\Microsoft SQL Server\150\DTS\Binn\;C:\Program
\Microsoft SQL Server\150\Tools\Binn\;C:\Program Files\Microsoft SQL Server\150\Tools
dowsApps;;C:\Program Files\JetBrains\IntelliJ IDEA 2022.1.3\bin;;C:\Users\Administrat
Data\Local\Programs\Microsoft VS Code\bin;C:\Users\Administrator\AppData\Roaming\npm
PS D:\21it167>
```

Task-7

Aim: Experiments with dotenv external module & set diff. Credentials for testing, UAT and production environments.

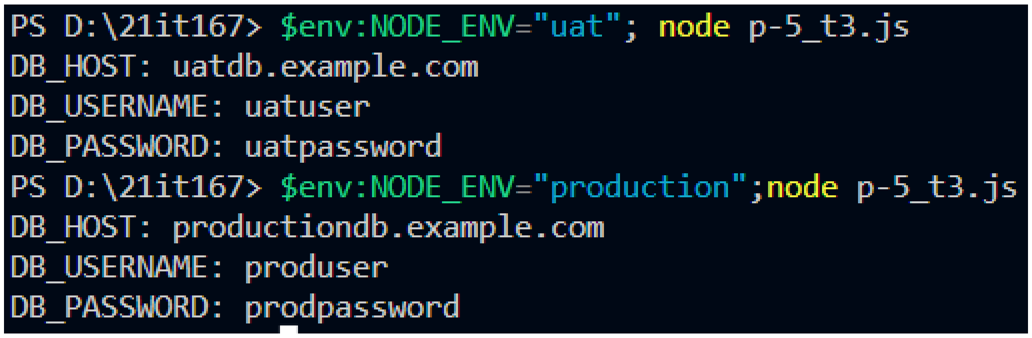
Description:

The aim is to utilize the dotenv external module to manage different sets of credentials for testing, UAT (User Acceptance Testing), and production environments in your application.

Source Code:

```
const dotenv = require('dotenv');
const environment = process.env.NODE_ENV || 'development';
dotenv.config({ path: `.env.${environment}` });
console.log("DB_HOST:", process.env.DB_HOST);
console.log("DB_USERNAME:", process.env.DB_USERNAME);
console.log("DB_PASSWORD:", process.env.DB_PASSWORD);
```

Output:



```
PS D:\21it167> $env:NODE_ENV="uat"; node p-5_t3.js
DB_HOST: uatdb.example.com
DB_USERNAME: uatuser
DB_PASSWORD: uatpassword
PS D:\21it167> $env:NODE_ENV="production";node p-5_t3.js
DB_HOST: productiondb.example.com
DB_USERNAME: produser
DB_PASSWORD: prodpassword
```

Learning Outcome:

CO2: Apply a deep knowledge of MVC(ModelViewController) architecture,making the development process easier and faster using open-source technologies.

Practical – 7&8

Task - 1

Aim:

Create Simple Hello World Component.

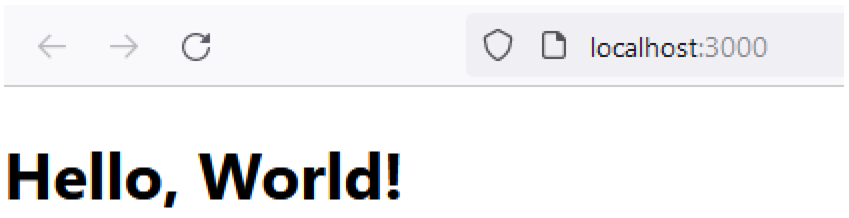
Description:

First we have to import the React library, which is necessary for working with React components. Then we define function in which we return our statement wrapped in div and at last export the HelloWorld component, making it available for use in other parts of your application. The export default syntax allows you to import the component using the import statement in other files.

Source Code:

```
import React from 'react';
function HelloWorld() {
  return (
    <div>
      <h1>Hello, World!</h1>
    </div>
  );
}
export default HelloWorld;
```

Output:



Hello, World!

Aim:

Create Basic/Static Product Page Using nested components & apply styles (exploring JSX Syntax)

Description:

Our goal is to design Product Page by s the power of nested components and applying styles using JSX syntax. You will begin by breaking down the page into reusable components such as ProductHeader, ProductImage, ProductInfo, and ProductFooter. By nesting these components within a main ProductPage component, you'll construct a cohesive layout. Additionally, you will utilize inline styling through JSX to creatively apply design elements such as colors, typography, and spacing.

Source Code:

ProductPage.js

```
import React from 'react';
import Header from './Header';
import Description from './Description';
import Price from './Price';
function ProductPage() {
  return (
    <div className="product-page">
      <Header />
      <Description />
      <Price />
    </div>
  );
}
export default ProductPage;
```

Header.js

```
import React from 'react';

function Header() {
  return (
    <div className="product-header">
      <h1>Mobile Phone</h1>
      <p>
        OnePlus:Oneplus Nord</p>
    </div>
  );
}
export default Header;
```

Description.js

```
import React from 'react';
function Description() {
  return (
    <div className="product-description">
      <p>
        The OnePlus Nord is a premium mid-range smartphone designed to deliver
        flagship-like features at an affordable price. With a sleek and modern
        design, the Nord features a 6.44-inch Fluid AMOLED display with a
        90Hz refresh rate, providing smooth and vibrant visuals.
      </p>
    </div>
  );
}

export default Description;
```

Price.js

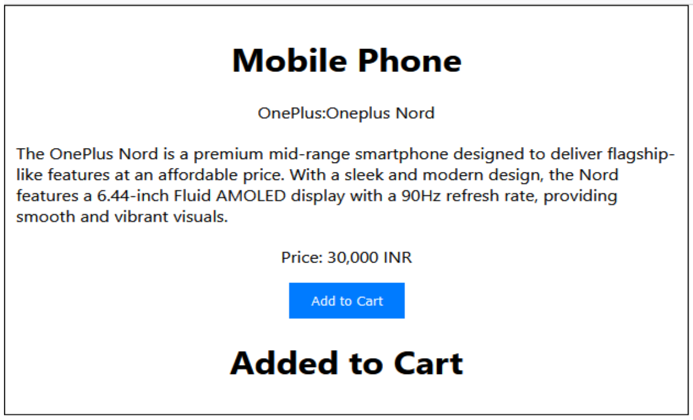
```
import React, { useState } from 'react';
function Price() {
  const [Name, setName] = useState("");
  return (
    <div className="product-price">
      <p>Price: 30,000 INR</p>
      <button onClick={() => {setName("Added to Cart");}}>Add to Cart</button>
      <h1>{Name}</h1>
    </div>
  );
}
export default Price;
```

styles.css

```
.product-page {
    width: 600px;
    margin: 0 ;
    padding: 10px;
    border: 1px solid black;
  }
```

```
.product-header {
  text-align: center;
  margin-bottom: 20px;
}

.product-description {
  margin-bottom: 20px;
}

.product-price {
  text-align: center;
}

button {
  padding: 10px 20px;
  background-color: #007bff;
  color: #fff;
  border: none;
  cursor: pointer;
}
button:hover{
  font-size: large;
  background-color: rgba(0, 0, 255, 0.733);
}
```

Output:



Mobile Phone

OnePlus:Oneplus Nord

The OnePlus Nord is a premium mid-range smartphone designed to deliver flagship-like features at an affordable price. With a sleek and modern design, the Nord features a 6.44-inch Fluid AMOLED display with a 90Hz refresh rate, providing smooth and vibrant visuals.

Price: 30,000 INR

Added to Cart

Aim:
Button Enable/Disable based on checkbox click event.

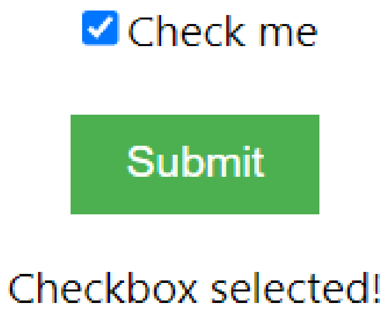
Description:
Our aim is to create a user interface where a button's enable/disable state is controlled by the click event of a checkbox. You will construct a component that includes both a checkbox and a button. Using React's state management, you'll implement the functionality to toggle the button's state based on whether the checkbox is clicked or not. When the checkbox is clicked, the button should become enabled and then text is displayed, and when it's unchecked, the button should become disabled.

Source Code:

```
import React, { useState } from 'react';
import './SubmitButton.css'; // Import the CSS file for styling

const SubmitButton = () => {
  const [isChecked, setIsChecked] = useState(false);
  const [message, setMessage] = useState('');

  const handleCheckboxChange = () => {
    setIsChecked(!isChecked);
  };

  const handleSubmit = () => {
    if (isChecked) {
      setMessage('Checkbox selected!');
    } else {
      setMessage('Checkbox not selected.');
    }
  };

  return (
    <div className="submit-button-container">
      <label htmlFor="checkbox">
        <input
          id="checkbox"
          type="checkbox"
          checked={isChecked}
          onChange={handleCheckboxChange}
        />
        Check me
      </label>
      <br /><br />
      <button
        className={`submit-button ${isChecked ? 'enabled' : 'disabled'}`}
        onClick={handleSubmit}
        disabled={!isChecked}
      >
        Submit
      </button>
      <p className="message">{message}</p> {/* Display the message */}
    </div>
  );
};

export default SubmitButton;

.submit-button-container {
    text-align: center;
    margin-top: 50px;
```

```
}

.submit-button {
  padding: 10px 20px;
  font-size: 16px;
  border: none;
  cursor: pointer;
}

.submit-button.enabled {
  background-color: #4caf50;
  color: white;
}

.submit-button.disabled {
  background-color: #ccc;
  color: #666;
  cursor: not-allowed;
}
```

Output:



```
Checkbox selected!
```

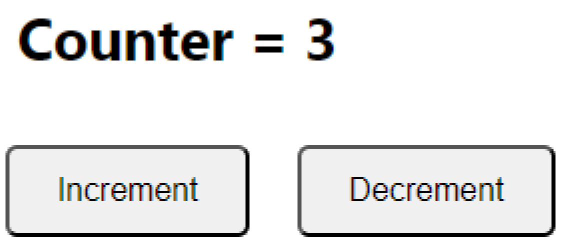
Aim:
Create Simple Counter Page Using useState Hook (share data between components).

Description:
Here our objective is to build a Counter Page utilizing the `useState` hook to manage state within your components. We will begin by creating a Counter component that maintains a numerical count state. Through the `useState` hook, we will manage the state and allow the count to increment or decrement based on user interactions. we will explore the concept of sharing data between components by rendering multiple instances of the Counter component on a single page.

Source Code:
```
import React, { useState } from 'react'; //App.js
import Counter from './Counter';
import CntBtns from './CntBtns';
function App() {
  const [cnt, setCount] = useState(0);
  return (
    <div>
       <Counter cnt={cnt} />
      <CntBtns Increment={() => {setCount(cnt + 1);}}Decrement={() => {
          setCount(cnt - 1);}}/>
    </div>
  );
}
export default App;

import React from 'react'; //Counter.js
import './style.css';
const Counter = ({ cnt }) => {
  return (
    <div>
      <h2>Counter = {cnt}</h2>
    </div>
  );
};
export default Counter;
```

```
import React from 'react'; //CntBtns.js
import './style.css';
const CntBtns = ({ Increment, Decrement }) => {
  return (
    <div>
      <button className="button" onClick={Increment}>Increment</button>
      <button className="button" onClick={Decrement}>Decrement</button>
    </div>
  );
};
export default CntBtns;

.button { //style.css
    margin:10px;
    padding: 10px 20px;
    border-radius: 5px;
    cursor: pointer;
  }
  h2{
    margin-left:15px;
  }
```

Output:


Counter = 3

Aim:
Create Task list Page and conditionally render Tasks based on hard-coded variables & Array.

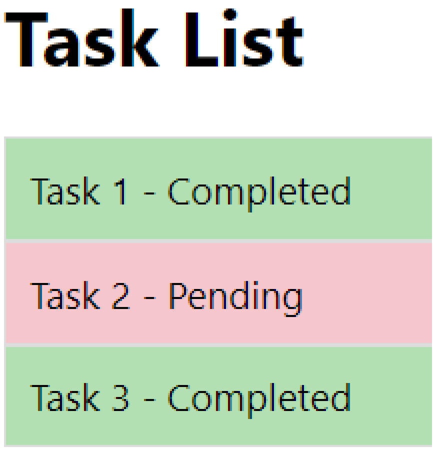
Description:
First our goal is to create a Task List Page where tasks are displayed based on predefined variables and an array. We will start by setting up the basic structure of the page, including a component to showcase the tasks. By employing conditional rendering techniques, we will selectively display tasks using hard-coded variables to simulate different scenarios. Furthermore, we will use an array of tasks, filtering and mapping through it to render task components, showcasing their details.

Source Code:

```
import React from 'react';
import TaskList from './TaskList';
import './App.css';

const App = () => {
  const tasks = [
    { id: 1, title: 'Task 1', completed: true },
    { id: 2, title: 'Task 2', completed: false },
    { id: 3, title: 'Task 3', completed: true },
  ];

  return (
    <div className="app">
      <h1>Task List</h1>
      <TaskList tasks={tasks} />
    </div>
  );
};

export default App;


import React from 'react';
import './Task.css'
```

```
const Task = ({ title, completed }) => {
  return (
    <div className={`task ${completed ? 'completed' : 'pending'}`}>
      <p className="task-title">
        {title} - {completed ? 'Completed' : 'Pending'}
      </p>
    </div>
  );
};

export default Task;

import React from 'react';
import Task from './Task';

const TaskList = ({ tasks }) => {
  return (
    <div className="task-list">
      {tasks.map(task => (
        <Task key={task.id} title={task.title} completed={task.completed} />
      ))}
    </div>
  );
};

export default TaskList;

.task {
    border: 1px solid #ddd;
    padding: 10px;
    text-align: left;
    background-color: #fff;
    display: flex; /* Add flex display */
    align-items: center; /* Center content vertically */
    justify-content: space-between; /* Separate title and status */
}

.completed {
  background-color: #b2e0b2;
}

.pending {
  background-color: #f5c6cb;
}

.task-title {
  margin: 0; /* Remove default margin */
}
```

Output:


Task List

Task 1 - Completed
Task 2 - Pending
Task 3 - Completed

Aim:
Show(render) List of Students (array of student objects) who are having less marks than '35' using Loop and Conditional Rendering: (filter function, map function).

Description:
We first develop a web application to display a list of student records using loop and conditional rendering techniques. Using an array of student objects containing attributes like name, marks, and id, we will employ the `filter` function to isolate students with marks below 35. Leveraging the `map` function, you will iterate

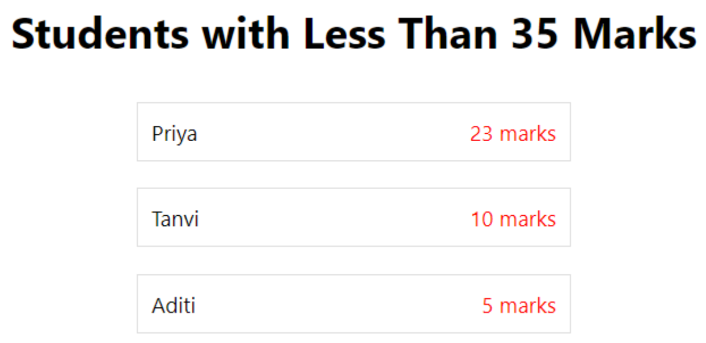
through the filtered array and dynamically render individual student components, showcasing their details.

Source Code:

```
import React from 'react';
import './Student.css';

const students = [
  { id: 1, name: 'Riya', marks: 90 },
  { id: 2, name: 'Sujal', marks: 95 },
  { id: 3, name: 'Het', marks: 89 },
  { id: 4, name: 'Dhruvi', marks: 91 },
  { id: 5, name: 'Bhavesh', marks: 40 },
  { id: 6, name: 'Priya', marks: 23 },
  { id: 7, name: 'Tanvi', marks: 10 },
  { id: 8, name: 'Aditi', marks: 5 },
];

const Student = () => {
  const marks = students.filter(student => student.marks < 35);

  return (
    <div className="app">
      <h1>Students with Less Than 35 Marks</h1>
      <div className="student-list">
        {marks.map(student => (
          <div key={student.id} className="student">
            <p className="student-name">{student.name}</p>
            <p className="student-marks">{student.marks} marks</p>
          </div>
        ))}
      </div>
    </div>
  );
};
```

```
export default Student;

.app {
   text-align: center;
   margin: 20px;
  }

  .student-list {
   display: flex;
   flex-direction: column;
   align-items: center;
   margin-top: 20px;
  }

  .student {
   border: 1px solid #ddd;
   padding: 10px;
   margin: 10px;
   display: flex;
   justify-content: space-between;
   align-items: center;
   width: 300px;
   background-color: #fff;
  }

  .student-name {
   margin: 0;
  }

  .student-marks {
   margin: 0;
   color: red;
  }
```

Output:



Students with Less Than 35 Marks

```
Priya        23 marks
Tanvi        10 marks
Aditi         5 marks
```

Aim:

Create Simple SPA Website using react router package for routing.

Description:

Here we first will utilize the power of React.js along with the React Router package to construct a Single Page Application (SPA) website. By creating multiple React components representing different sections of your website's content, we will implement seamless navigation and routing using React Router.

Source Code:

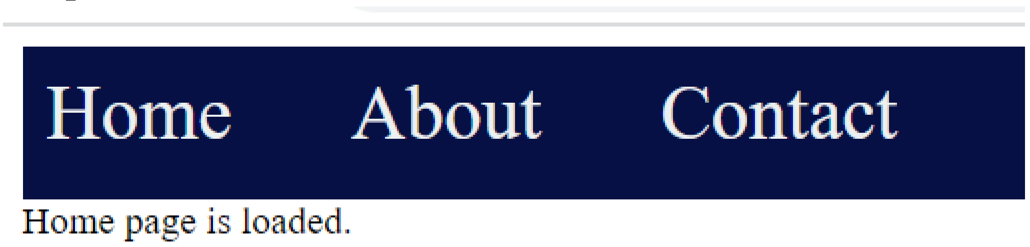
```
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom';
import Home from './components/Home';
import About from './components/About';
import Contact from './components/Contact';
import './App.css';

const App = () => {
  return (
    <Router>
      <div className="app">
        <nav>
        <ul>
            <li>
              <Link to="/">Home</Link>
```

```
          </li>
          <li>
            <Link to="/about">About</Link>
          </li>
          <li>
            <Link to="/contact">Contact</Link>
          </li>
        </ul>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
      </Routes>
    </div>
  </Router>
 );
};

export default App;

.App {

 margin: 0 auto;
 padding: 20px;
}

nav ul {
 list-style: none;
 height: 4rem;
 padding: 0;
 margin: 0;
 display: flex;
 background-color: #071044;
}
```

```
nav li {
  margin-top: 10px;
  font-size:xx-large;
  margin-right: 40px;
  margin-left: 10px;
}

nav a {
  text-decoration: none;
  color: #ecf0f1;
  transition: color 0.5s;
}

nav a:hover {
  color: #b434db;
}

nav a.active {
  color: #3498db;
}

import React from 'react';

const About = () => {
  return <div> About page is loaded.</div>;
};

export default About;

import React from 'react';

const Contact = () => {
  return <div> Contact page is loaded.</div>;
};
```

```
export default Contact;

import React from 'react';

const Home = () => {
  return <div> Home page is loaded.</div>;
};

export default Home;
```

Output:



```
Home page is loaded.
```

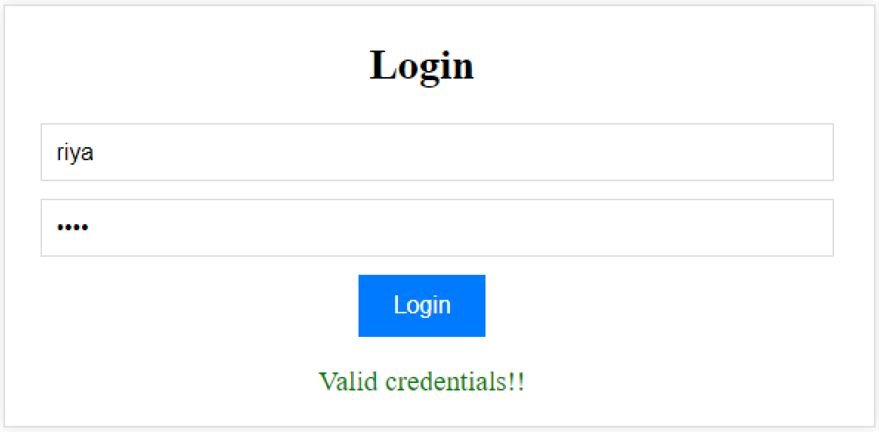
Aim:
Create a Simple Login Page and Validate it.

Description:
First here we create a basic login page using React.js with input validation. You'll start by building a login form component that includes fields for username and password. Using React's state management, event handling, and conditional rendering, you will implement client-side validation to ensure that both fields are filled before submission. By employing CSS for styling and React's component-based architecture, you will create a visually appealing and interactive user experience. This exercise provides a hands-on introduction to building dynamic web interfaces using React and emphasizes the importance of user input validation in enhancing the functionality and security of web applications.

Source Code:

```
import React, { useState } from 'react';
import './App.css';

const App = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [errorMessage, setErrorMessage] = useState('');

  const handleLogin = () => {
    // Simulate validation
    if (username === 'riya' && password === 'riya') {
      setErrorMessage('Valid credentials!!');
      setTimeout(() => {
        setErrorMessage('');
      }, 2000);
    } else {
      setErrorMessage('Invalid credentials!!');
      setTimeout(() => {
        setErrorMessage('');
      }, 2000);
    }
  };

  return (
    <div className="login-page">
      <div className="login-form">
        <h2>Login</h2>
        <input
          type="text"
          placeholder="Username"
          value={username}
          onChange={e => setUsername(e.target.value)}
        />
        <input
          type="password"
          placeholder="Password"
          value={password}
          onChange={e => setPassword(e.target.value)}
        />
        <button onClick={handleLogin}>Login</button>
        <p className={` ${errorMessage.includes('Valid') ? 'success' : 'error-
message'}`}>
          {errorMessage}
        </p>
      </div>
    </div>
  );
};

export default App;

.login-page {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}
.success {
  color: green;
}

.login-form {
  border: 1px solid #ddd;
  padding-right: 40px;
  padding-left: 20px;
  text-align: center;
  background-color: #fff;
  box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
}

.login-form h2 {
  margin-bottom: 20px;
}

.login-form input {
  width: 100%;
  margin-bottom: 10px;
  padding: 8px;
  border: 1px solid #ddd;
}

.login-form button {
  background-color: #007bff;
  color: white;
  border: none;
  padding: 10px 20px;
  cursor: pointer;
}

.login-form button:hover {
  background-color: #0056b3;
}

.error-message {
  color: red;
  font-size: 14px;
  margin-top: 10px;
}
```

Output:



Valid credentials!!

Learning Outcome:

Completing these React practical exercises provides a well-rounded learning experience. You'll cover key aspects like form validation, SPA navigation with React Router, dynamic rendering using map and filter, useState for component interactivity, and styling through JSX. By mastering these techniques, you'll gain a strong foundation in React, enabling you to build interactive and visually appealing web applications.

Practical - 9

Task - 1

Aim:

Create and Design E-Commerce Website - UI (Front-End Only) with Shopping Cart Facility:

- Create Product Listing Page with at least Five Products (Product image, Price, Discount, Add to Cart Button, Product Details)
- Create a Product Detail Page and Display appropriate Details About Product.
- Create Shopping Cart Page which will display
  - List of Product in Cart
  - Product Quantity modification
  - Product Remove from Cart Button
  - Discount
  - Total Price

Guideline:

- Store product data in a JSON file or a JavaScript array.
- Implement required Routing & Middlewares
- Handle state management for cart items using React's useState & useEffect
- You can use third-party CSS Framework/modules of your choice for layout and design

Description:

Source Code:

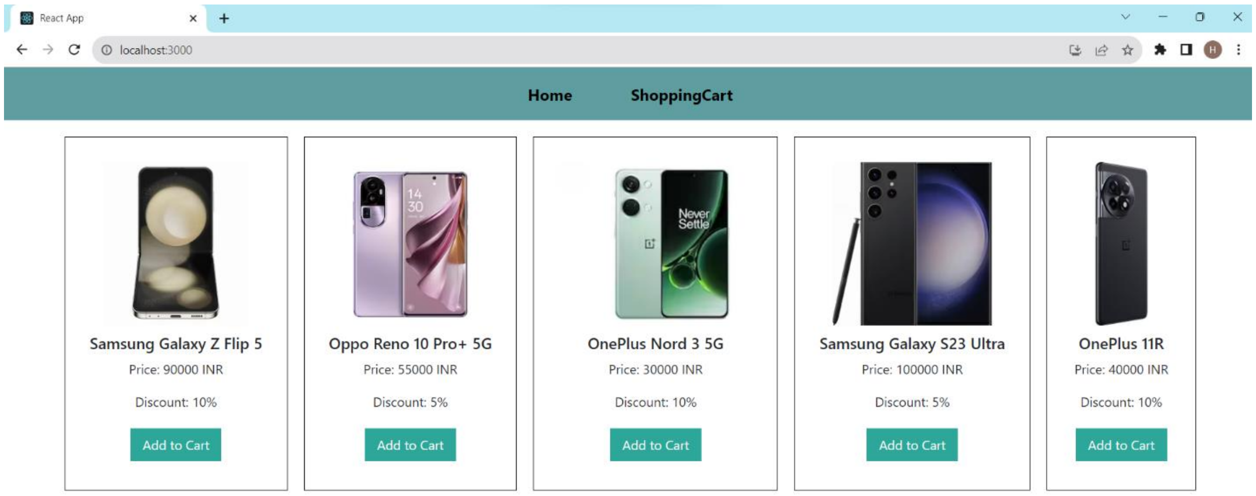
```
import React, { useState } from 'react'; //App.js
import {BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom';
import ProductList from './components/ProductList';
import ProductDetail from './components/ProductDetail';
import ShoppingCart from './components/ShoppingCart';
import './App.css';

const App = () => {
  const [cartItems, setCartItems] = useState([]);
const addToCart = product => {
  const existingItem = cartItems.find(item => item.id === product.id);

  if (existingItem) {
    setCartItems(prevItems =>
      prevItems.map(item =>
        item.id === product.id ? { ...item, quantity: item.quantity + 1 } : item
      )
    );
  } else {
    setCartItems(prevItems => [...prevItems, { ...product, quantity: 1 }]);
  }
};

const updateQuantity = (productId, newQuantity) => {
  setCartItems(prevItems =>
    prevItems.map(item =>
      item.id === productId ? { ...item, quantity: newQuantity } : item
    )
  );
};

const removeFromCart = productId => {
  setCartItems(prevItems => prevItems.filter(item => item.id !== productId));
};

return (
  <Router>
    <nav className='navbar'>
    <Link className="nav-link" to="/">Home</Link>
    <Link className="nav-link" to="/cart">
      ShoppingCart
    </Link>
    </nav>
```

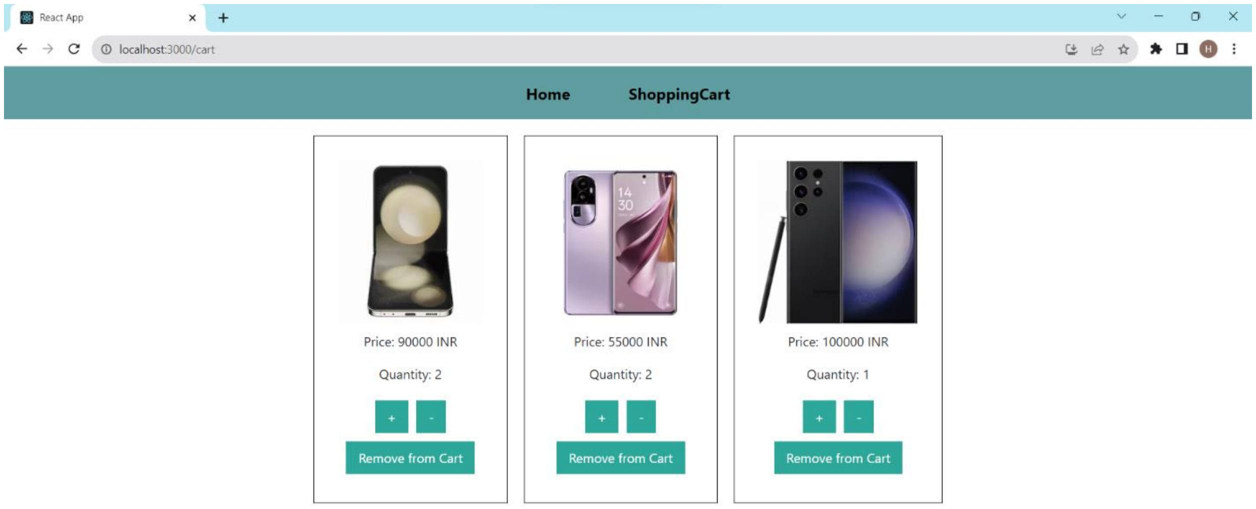
```
      <Routes>
        <Route path="/" element={<ProductList addToCart={addToCart} />} />
        <Route path="/product/:productId" element={<ProductDetail />} />
        <Route
          path="/cart"
          element={
            <ShoppingCart
              cartItems={cartItems}
              updateQuantity={updateQuantity}
              removeFromCart={removeFromCart}
            />
          }
        />
      </Routes>
    </Router>
  );
};
export default App;
.App {
  text-align: center;
  font-family: Arial, sans-serif;
}

.product-list {
  display: flex;
  flex-wrap: wrap;
  justify-content: center;
  gap: 20px;
  padding: 20px;
}

.product {
  border: 1px solid black;
  padding: 30px;
  text-align: center;
  max-width: 300px;
  transition: transform 0.5s;
}

.product img {
  width: 230px;
  height: 300px;
}

.product h3 {
  margin-top: 10px;
  font-size: 1.2rem;
}

.product p {
  margin-top: 5px;
  font-size: 1rem;
}

.product button {
  background-color:rgb(44, 167, 153);
  height:40px;
  color: white;
  border: black;
  padding: 8px 15px;
  margin: 5px 5px ;
  cursor: pointer;
  transition: background-color 0.5s;
}

.product button:hover {
  background-color: #61dafb;
}

/* Add this to your App.css */
```

```
.navbar {
  display: flex;
  justify-content: center;
  align-items: center;
  background-color: #61dafb;
  padding: 10px 20px;
}

.nav-link {
  color: rgb(0, 0, 0);
  font-weight: bold;
  text-decoration: none;
  margin-left: 20px;
  margin-right: 20px;
  font-size: larger;
  transition:  0.2s;
}
//data/products.js
const products = [
    {
    id: 1,
    name: 'Samsung Galaxy Z Flip 5',
    image: 'product1.jpeg',
    price: 90000.0,
    discount: 10,
    details: 'Product Details...',
  },
  {
    id: 2,
    name: 'Oppo Reno 10 Pro+ 5G',
    image: 'product2.jpeg',
    price: 55000.0,
    discount: 5,
    details: 'Product Details...',
  },
  {
    id: 3,
    name: 'OnePlus Nord 3 5G',
    image: 'product3.jpeg',
    price: 30000.0,
    discount: 10,
    details: 'Product Details...',
  },
  {
    id: 4,
    name: 'Samsung Galaxy S23 Ultra',
    image: 'product4.jpeg',
    price: 100000.0,
    discount: 5,
    details: 'Product Details...',
  },
  {
    id: 5,
    name: 'OnePlus 11R',
    image: 'product5.jpeg',
    price: 40000.0,
    discount: 10,
    details: 'Product Details...',
  },
  ];
  export default products;
//ProductList.js
import React from 'react';
import products from '../data/products'; // Import the product data
const ProductList = ({ addToCart }) => {
  return (
    <div className="product-list">
      {products.map(product => (
        <div key={product.id} className="product">
          <img src={product.image} width="90px" alt={product.name} />
```

```
          <h3>{product.name}</h3>
          <p>Price: {product.price} INR</p>
          <p>Discount: {product.discount}%</p>
          <button onClick={() => addToCart(product)}>Add to Cart</button>

        </div>
      ))}
    </div>
  );
};
export default ProductList;
import React from 'react';
//cartItem.js
const CartItem = ({ item, updateQuantity, removeFromCart }) => {
  return (
    <div className="product">
      <img src={item.image} alt={item.title} />
      <h3>{item.title}</h3>
      <p>Price: {item.price} INR</p>
      <p>Quantity: {item.quantity}</p>
      <button
        className="cart-link"
        onClick={() => updateQuantity(item.id, item.quantity + 1)}
      >
        +
      </button>
      <button
        className="cart-link"
        onClick={() => updateQuantity(item.id, item.quantity - 1)}
        disabled={item.quantity <= 1}
      >
        -
      </button><br></br>
      <button
        className="cart-link"
        onClick={() => removeFromCart(item.id)}
      >
        Remove from Cart
      </button>
    </div>
  );
};
export default CartItem;
//ShoppingCart.js
import React from 'react';
import CartItem from './CartItem';
const ShoppingCart = ({ cartItems, updateQuantity, removeFromCart }) => {
  return (
    <div className="product-list">
      {cartItems.map(item => (
        <CartItem
          key={item.id}
          item={item}
          updateQuantity={updateQuantity}
          removeFromCart={removeFromCart}
        />
      ))}
    </div>
  );
};
export default ShoppingCart;
//ProductDetail.js
import React from 'react';
const ProductDetail = ({ product }) => {
  return (
    <div className="product-detail">
      <img src={product.image} width="30px" height="30px" alt={product.name}
/>
      <h2>{product.name}</h2>
      <p>Price: {product.price} INR</p>
      <p>Discount: {product.discount}%</p>
      <p>{product.details}</p>
    </div>
  );
};
export default ProductDetail;
```

Output:





Learning Outcome:

By finishing this hands-on exercise, we acquired the skills to utilize react-router-dom for routing and various other applications. Additionally, we gained insight into the functioning of single-page applications.

Practical - 10

Task - 1

Aim:

Experiments with React Hooks: -

- useState
- useEffect
- useRef
- useContext
- useMemo

Description:

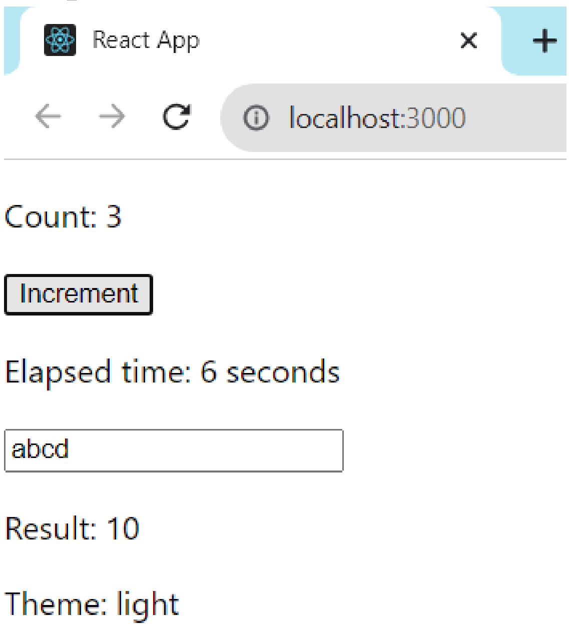
These experiments help developers understand how to manage state, handle side effects, access and manipulate DOM elements, share data through context, and optimize calculations in React applications using Hooks.

Source Code:

```
import React, { useState, useEffect, useRef, useContext, useMemo } from 'react';

const ThemeContext = React.createContext();

function ExpensiveCalculation({ value }) {
  const result = useMemo(() => {
    // Perform some expensive calculation
    return value * 2;
  }, [value]);

  return <p>Result: {result}</p>;
}

function Timer() {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setSeconds(seconds + 1);
    }, 1000);

    return () => clearInterval(interval);
  }, [seconds]);

  return <p>Elapsed time: {seconds} seconds</p>;
}

function FocusInput() {
  const inputRef = useRef(null);

  useEffect(() => {
    inputRef.current.focus();
  }, []);

  return <input ref={inputRef} />;
}

function Toolbar() {
  const theme = useContext(ThemeContext);
  return <div>Theme: {theme}</div>;
}

function App() {
  return (
    <ThemeContext.Provider value="light">
      <div>
        <Counter />
        <Timer />
        <FocusInput />
        <ExpensiveCalculation value={5} />
        <Toolbar />
      </div>
    </ThemeContext.Provider>
  );
```

```
}

function Counter() {
  const [count, setCount] = useState(0);

  const increment = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
}

export default App;
```

Output:



```
Count: 3

Increment

Elapsed time: 6 seconds

abcd

Result: 10

Theme: light
```

Task – 2

Aim:

Create a Custom React Hook to Fetch PUBLIC API data and display it on the webpage.

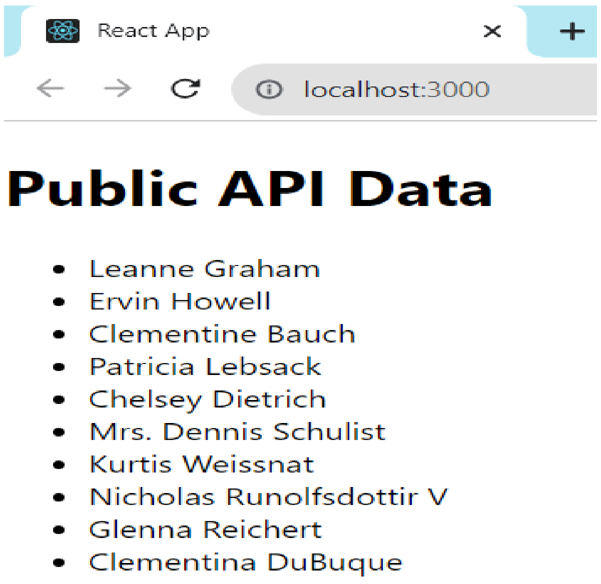
Description:

Makes a special tool in React that gets information from a public website and shows it on a webpage.

Source Code:

```
import React, { useState, useEffect } from 'react';

// Custom hook for fetching data from a public API
function usePublicApiData(apiUrl) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    // Fetch data from the API when the component mounts
    fetch(apiUrl)
      .then((response) => {
        if (!response.ok) {
          throw new Error('Network response was not ok');
        }
        return response.json();
      })
      .then((data) => {
        setData(data);
        setLoading(false);
      })
      .catch((error) => {
        setError(error);
        setLoading(false);
      });
  }, [apiUrl]);

  return { data, loading, error };
}

function MyComponent() {
  const apiUrl = 'https://jsonplaceholder.typicode.com/users'; // Example API URL
  const { data, loading, error } = usePublicApiData(apiUrl);

  if (loading) {
    return <div>Loading...</div>;
  }

  if (error) {
    return <div>Error: {error.message}</div>;
  }
  if (!data) {
    return null; // Handle the case where data is still null
  }
  return (
    <div>
      <h1>Public API Data</h1>
      <ul>
        {data.map((user) => (
          <li key={user.id}>{user.name}</li>
        ))}
      </ul>
    </div>
  );
}

function App() {
  return (
    <div className="App">
      <MyComponent />
    </div>
  );
}

export default App;
```

Output:



```
Public API Data

  • Leanne Graham
  • Ervin Howell
  • Clementine Bauch
  • Patricia Lebsack
  • Chelsey Dietrich
  • Mrs. Dennis Schulist
  • Kurtis Weissnat
  • Nicholas Runolfsdottir V
  • Glenna Reichert
  • Clementina DuBuque
```

Learning Outcome:

By experimenting with React Hooks we have learned to effectively manage state, work with side effects, manipulate web page elements, share data between components, and optimize our React applications.

Additionally, creating a custom React Hook to fetch data from public APIs and display it on a webpage will teach us how to integrate external data sources into our applications, enhancing our ability to build dynamic and data-driven web apps.

Practical - 11

Aim:

Full-Stack to-do application.

Description:

The aim of a full-stack to-do application is to provide users with a comprehensive tool for managing their tasks and to-do lists. This application encompasses both a user-friendly front-end interface for creating, editing, and organizing tasks, and a back-end server that stores and handles task data. The ultimate objective is to enhance users' productivity and organization by facilitating efficient task management.

Source Code:

Client/App.js:

```
import React, { useState } from 'react';
import axios from 'axios';

function App() {
  const [todos, setTodos] = useState([]);
  const [todo, setTodo] = useState('');

  const addTodo = () => {
    if (todo.trim() !== '') {
      axios
        .post('/api/todos', { todo })
        .then(() => {
          setTodos([...todos, todo]);
          setTodo('');
        })
        .catch((error) => {
          console.error('Error adding todo:', error);
        });
    }
  };

  return (
    <div className="App">
      <h1>Todo App</h1>
      <div>
```

```
        <input
          type="text"
          value={todo}
          onChange={(e) => setTodo(e.target.value)}
          placeholder="Enter a new todo"
        />
        <button onClick={addTodo}>Add</button>
      </div>
      <ul>
        {todos.map((t, index) => (
          <li key={index}>{t}</li>
        ))}
      </ul>
    </div>
  );
}

export default App;
```

<u>server/server.js:</u>

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');

const app = express();
const port = process.env.PORT || 5000;

app.use(cors());
app.use(bodyParser.json());

app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});

const todos = [];

app.get('/api/todos', (req, res) => {
  res.json(todos);
});
```

```
app.post('/api/todos', (req, res) => {
  const newTodo = req.body.todo;
  todos.push(newTodo);
  res.json({ message: 'Todo added successfully' });
});
```

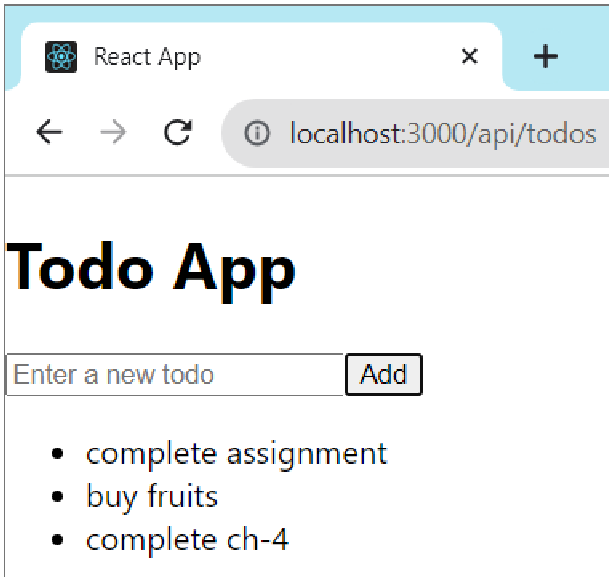
Login.ejs:

```
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <h1>Login</h1>
    <form action="/login" method="POST">
        <label for="username">Username:</label>
        <input type="text" name="username" required><br>
        <label for="password">Password:</label>
        <input type="password" name="password" required><br>
        <button type="submit">Login</button>
    </form>
</body>
</html>
```

Style.css

```
  .todo-list {
    margin: 0;
    padding: 0;
    list-style: none;
  }

  .todo-item {
    display: flex;
    align-items: center;
    margin-bottom: 10px;
    padding: 10px;
    background-color: #f5f5f5;
    border-radius: 5px;
  }
```

Output:



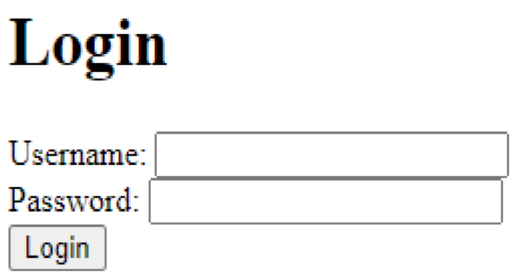
React App

localhost:3000/api/todos

Todo App

Enter a new todo | Add

- complete assignment
- buy fruits
- complete ch-4



Login

Username:

Password:

Login

Learning Outcome:

This project has taught us how to create interactive and responsive web applications, enhance our JavaScript skills, and use RESTful API design and usage. We also figured out how to handle important things like keeping track of what's happening in our app (state management) and letting users log in (user authentication).

Practical - 12

Aim:

Building a User Authentication System with Sessions and Cookies without Database (hard-coded username and password)
Create Front End in REACT and Back End using Node and Expressjs.
use Cookies to remember background color of Front-End UI.

Description: - Cookie-parser is like a middleware for your website. It helps your website understand and use cookies, which are like little pieces of information that websites use to remember things about you. This makes it easier to do things like keeping you logged in or remembering your preferences when you use a website built with Express.js.

server.js

```
// server/server.js
const express = require('express');
const cookieParser = require('cookie-parser');
const bodyParser = require('body-parser');

const app = express();
const port = 5000;

// Middleware
app.use(express.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.use(cookieParser());

// Hard-coded users (in-memory)
const users = [
  { username: 'Heer', password: 'heer1234' }
];

// Authentication Middleware
const requireAuth = (req, res, next) => {
  const username = req.cookies.user;

  if (!username) {
    return res.status(401).json({ message: 'Authentication required' });
  }
  // You can perform more validation here if needed

  next();
};

// Login Route
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  const user = users.find((user) => user.username === username && user.password ===
password);

  if (!user) {
    return res.status(401).json({ message: 'Invalid credentials' });
  }

  // Set a session cookie to remember the user's login
  res.cookie('user', username, { httpOnly: true });
  res.json({ message: 'Login successful' });
});

// Logout Route
app.get('/logout', (req, res) => {
  // Clear the session cookie
  res.clearCookie('user');
  res.json({ message: 'Logout successful' });
});

// Protected Route - Requires Authentication
app.get('/dashboard', requireAuth, (req, res) => {
  res.json({ message: 'Welcome to the dashboard!' });
});

app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

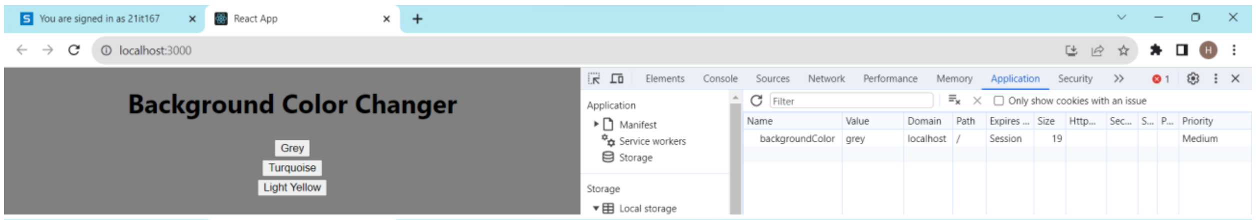
App.js

```
// client/src/App.js
```

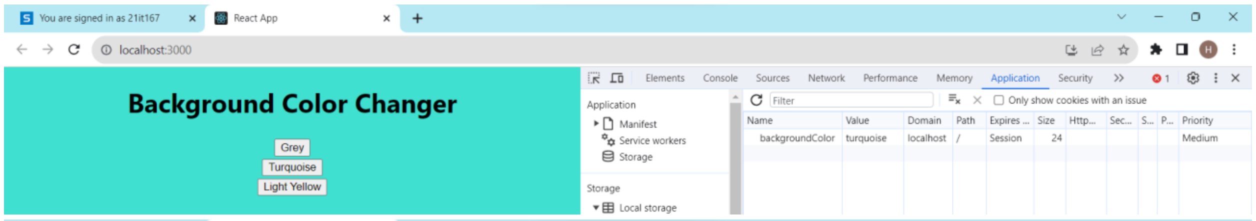
```
import React, { useState, useEffect } from 'react';
import './App.css';

function App() {
  const [backgroundColor, setBackgroundColor] = useState('');

  useEffect(() => {
    // Retrieve the backgroundColor cookie when the component mounts
    const savedBackgroundColor = getCookie('backgroundColor');
    if (savedBackgroundColor) {
      setBackgroundColor(savedBackgroundColor);
    }
  }, []);

  // Function to set the backgroundColor cookie on button click
  const setCookieAndBackgroundColor = (color) => {
    document.body.style.backgroundColor = color;
    document.cookie = `backgroundColor=${color}`;
    setBackgroundColor(color);
  };

  // Function to retrieve a cookie by name
  const getCookie = (name) => {
    const cookies = document.cookie.split(';');
    for (const cookie of cookies) {
      const [cookieName, cookieValue] = cookie.trim().split('=');
      if (cookieName === name) {
        return cookieValue;
      }
    }
    return null;
  };

  return (
    <div className="App">
      <h1>Background Color Changer</h1>
      <button onClick={() => setCookieAndBackgroundColor('lightblue')}>Light Blue</button>
      <button onClick={() => setCookieAndBackgroundColor('lightgreen')}>Light
Green</button>
```

```
      <button onClick={() => setCookieAndBackgroundColor('lightpink')}>Light Pink</button>
    </div>
  );
}

export default App;
```

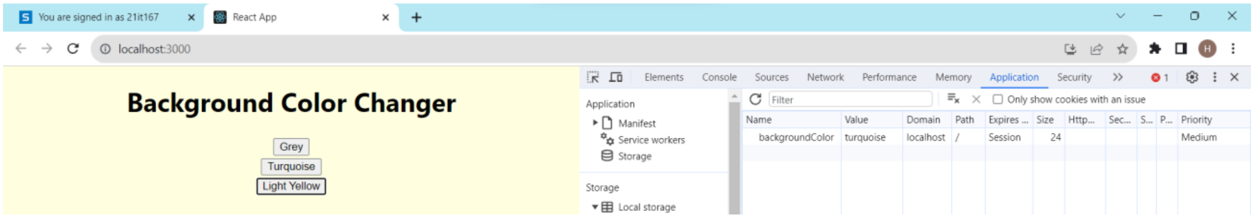
Output:



Name	Value	Domain	Path	Expires ...	Size	Http...	Sec...	S...	P...	Priority
backgroundColor	grey	localhost	/	Session	19					Medium



Name	Value	Domain	Path	Expires ...	Size	Http...	Sec...	S...	P...	Priority
backgroundColor	turquoise	localhost	/	Session	24					Medium



Name	Value	Domain	Path	Expires ...	Size	Http...	Sec...	S...	P...	Priority
backgroundColor	turquoise	localhost	/	Session	24					Medium

Learning Outcome:-

After completing this practical, I will be able to develop a User Authentication System with Sessions and Cookies, utilizing hard-coded username and password. I will also gain proficiency in building the Front-End using React and the Back-End with Node and Express.js. Furthermore, I will learn how to employ Cookies to remember the background color of the Front-End UI.

Practical-13

Aim: MERN STACK Mini Project

Output:

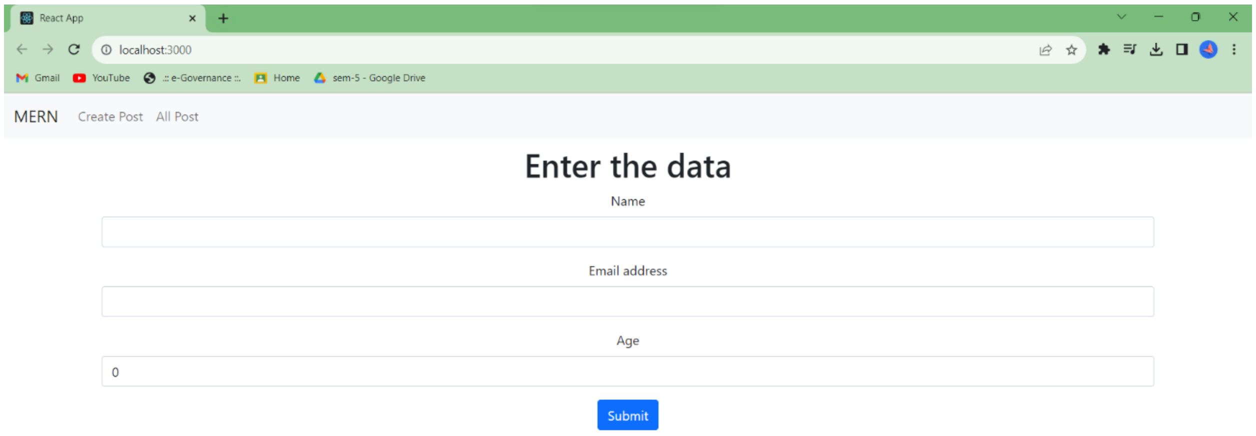
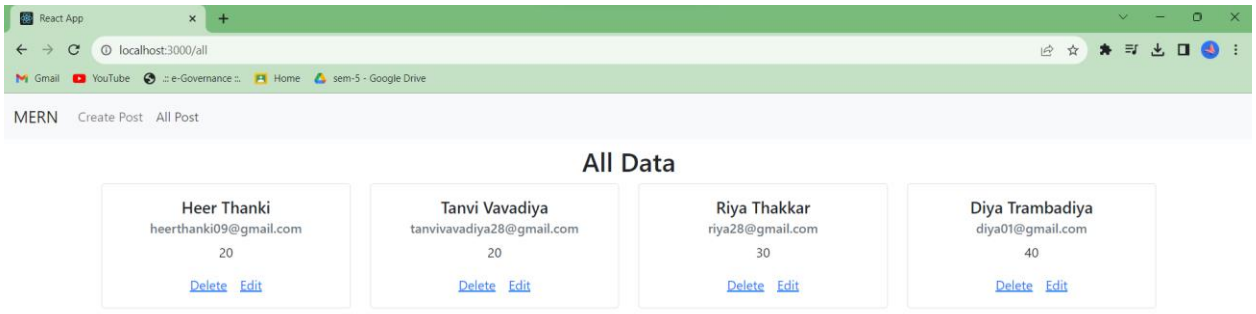


Fig-1: Home page of website, where user can add the data.



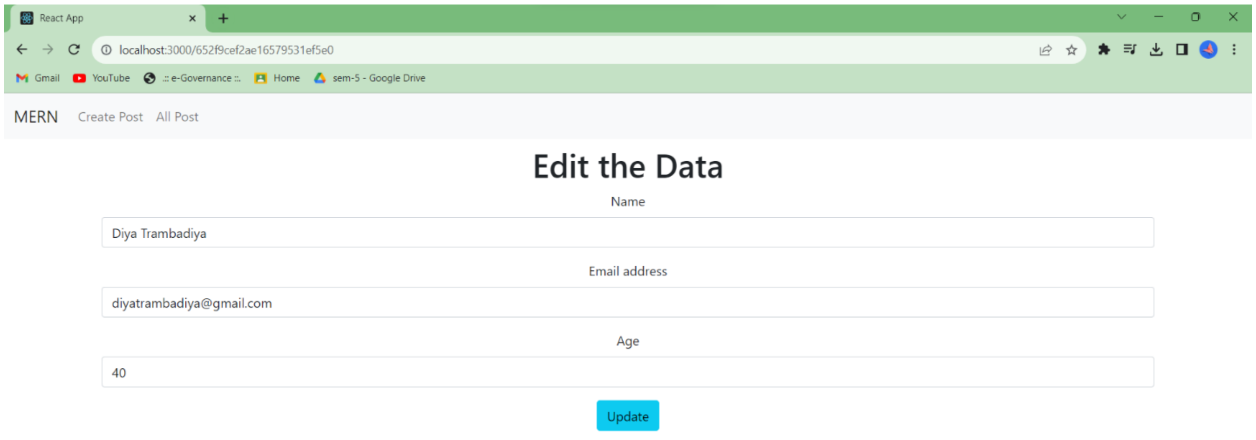
Heer Thanki
heerthanki09@gmail.com
20
Delete Edit

Tanvi Vavadiya
tanvivavadiya28@gmail.com
20
Delete Edit

Riya Thakkar
riya28@gmail.com
30
Delete Edit

Diya Trambadiya
diya01@gmail.com
40
Delete Edit

Fig-2: Second page of website, where all user data can be seen.



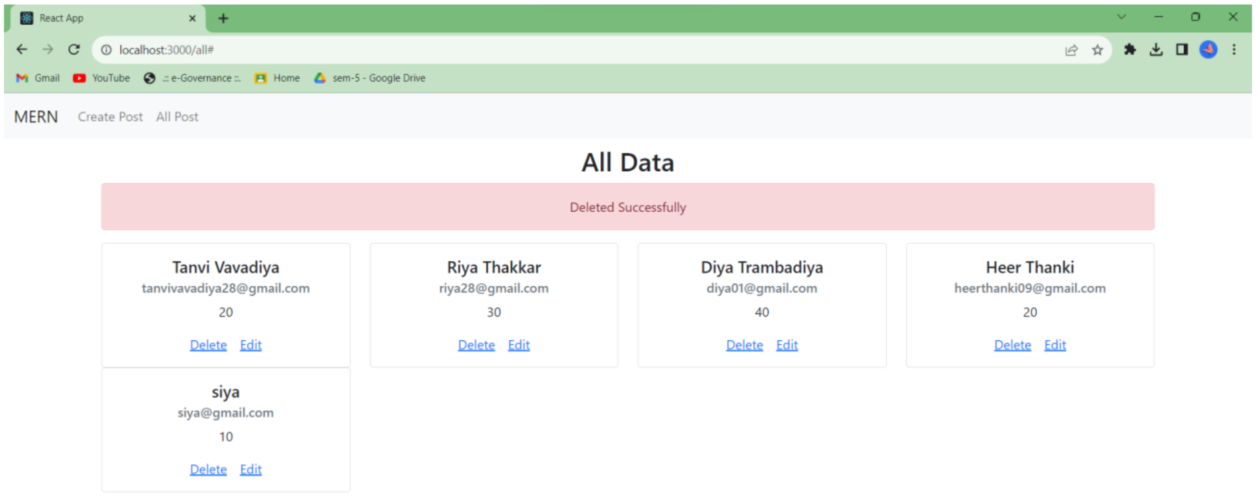
Edit the Data

Name: Diya Trambadiya

Email address: diyatrambadiya@gmail.com

Age: 40

Fig-3: Third page of website, where user can edit the data.

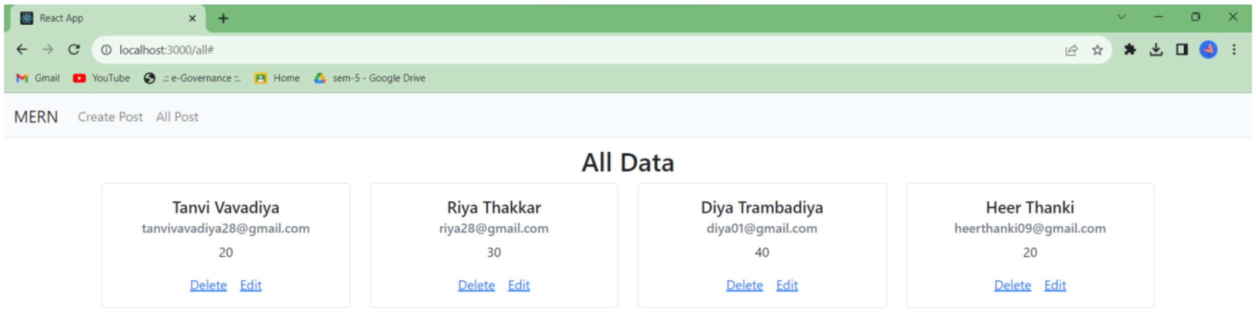


All Data

Deleted Successfully

Tanvi Vavadiya tanvivavadiya28@gmail.com 20 Delete Edit	Riya Thakkar riya28@gmail.com 30 Delete Edit	Diya Trambadiya diya01@gmail.com 40 Delete Edit	Heer Thanki heerthanki09@gmail.com 20 Delete Edit
siya siya@gmail.com 10 Delete Edit			

Fig-4: Here, deleted operation is performed on single data.



All Data

Tanvi Vavadiya tanvivavadiya28@gmail.com 20 Delete Edit	Riya Thakkar riya28@gmail.com 30 Delete Edit	Diya Trambadiya diya01@gmail.com 40 Delete Edit	Heer Thanki heerthanki09@gmail.com 20 Delete Edit

Fig-5: Output after delete operation is performed on single data.