**EXPERIMENT 1**

**DEMONSTRATION OF CONST, LET, STRING TEMPLATES, CALLBACKS, ARROW FUNCTIONS, CLASS, CLASS-PROPERTIES, METHODS USING JAVA SCRIPT**

**Aim:**

To demonstrate the usage of const, let, string templates, callbacks, arrow functions, classes, class properties, and methods in JavaScript.

**Theory:**

1. **Const and Let:**
   * const: Used to declare constants whose values cannot be reassigned. It ensures read-only reference.
   * let: Used for variables that can be reassigned. It is block-scoped and does not allow redeclaration in the same scope.
2. **String Templates:**
   * Introduced in ES6, string templates use backticks (`) and ${} for embedding variables and expressions within strings.
3. **Arrow Functions:**
   * Concise syntax for writing functions. They inherit the this value from the enclosing context.
4. **Callbacks:**
   * Functions passed as arguments to other functions. They are invoked after an operation completes.
5. **Classes, Class Properties, and Methods:**
   * ES6 introduced classes as syntactic sugar over JavaScript's prototype-based inheritance.
   * constructor() initializes object properties.
   * Methods are functions defined within the class.

**Programs:**

**1. Const and Let**

// Using const and let

const PI = 3.14;

let radius = 5;

let area = PI \* radius \* radius;

console.log("Area of circle:", area);

// Trying to change const value will give an error

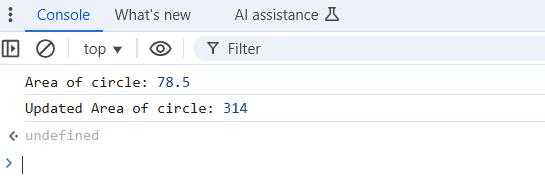
// PI = 3.14159;  // Uncommenting this line will cause an error

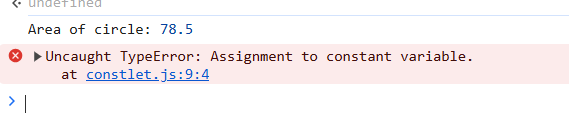
radius = 10;

area = PI \* radius \* radius;

console.log("Updated Area of circle:", area);

**Output:**





**2. String Templates**

// Using string templates

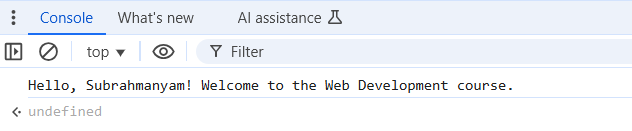
let name = "Subrahmanyam";

let course = "Web Development";

let message = `Hello, ${name}! Welcome to the ${course} course.`;

console.log(message);

**Output:**



**3. Arrow Functions**

// Using arrow function

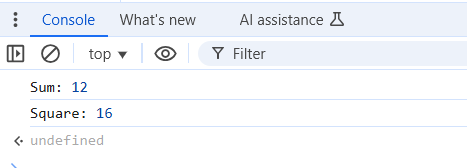
const add = (a, b) => a + b;

const square = (x) => x \* x;

console.log("Sum:", add(5, 7));

console.log("Square:", square(4));

**Output:**



**4. Callbacks**

// Using callbacks

function greet(name, callback) {

    console.log("Hello, " + name);

    callback();

}

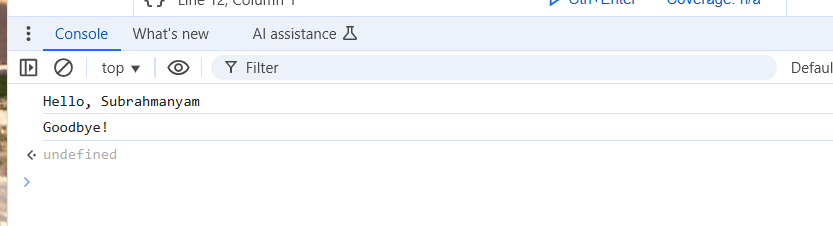
function bye() {

    console.log("Goodbye!");

}

greet("Subrahmanyam", bye);

**Output:**



**5. Class, Class Properties, and Methods**

// Using Class, Properties, and Methods

class Person {

    constructor(name, age) {

        this.name = name;

        this.age = age;

    }

    // Method

    greet() {

        console.log(`Hi, my name is ${this.name} and I am ${this.age} years old.`);

    }

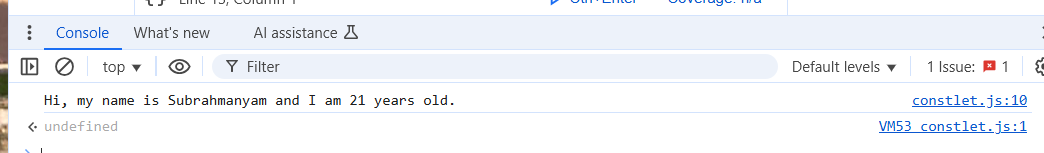
}

// Creating an object

const person1 = new Person("Subrahmanyam", 21);

person1.greet();

**Output:**



**EXPERIMENT 2**

**USE OF GLOBAL OBJECT, VARIABLES, STANDARD INPUT AND STANDARD OUTPUT IN NODE JS TO BUILD APPLICATION**

**Aim:**

To understand the use of the global object, variables, standard input, and standard output in Node.js to build an application.

**Theory:**

1. **Global Object (global)**
   * In Node.js, global is the top-level object, similar to window in browsers.
   * It provides access to built-in modules, functions, and variables that can be used anywhere in the application.
2. **Variables in Node.js**
   * Variables in Node.js can be declared using var, let, and const.
   * let and const are block-scoped, while var is function-scoped.
3. **Standard Input (process.stdin)**
   * Node.js allows reading user input via process.stdin.
   * The readable stream listens for user input from the terminal.
4. **Standard Output (process.stdout)**
   * Node.js uses process.stdout to print messages to the console.
   * console.log() is the most common method for outputting data.

**Programs:**

**1. Demonstrating the Global Object**

// Accessing the global object

console.log("Current Directory:", \_\_dirname);

console.log("Current File:", \_\_filename);

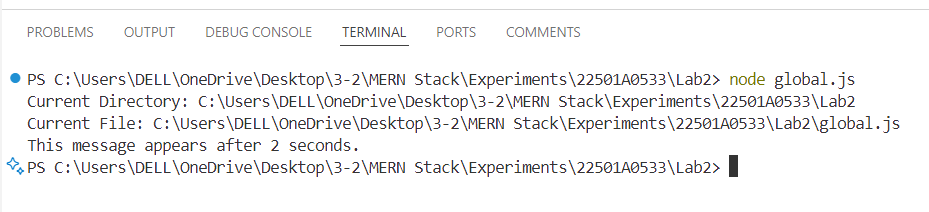
// Using setTimeout (part of the global object)

global.setTimeout(() => {

    console.log("This message appears after 2 seconds.");

}, 2000);

**Output:**



**2. Using Variables in Node.js**

let name = "Subrahmanyam";

const course = "Node.js Application";

console.log(`Name: ${name}`);

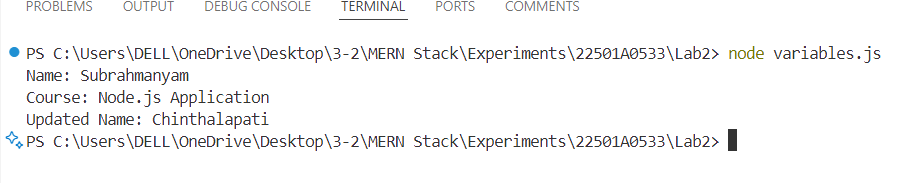
console.log(`Course: ${course}`);

// Changing the value of let variable

name = "Chinthalapati";

console.log(`Updated Name: ${name}`);

**Output:**



**3. Standard Input (Reading User Input)**

const readline = require('readline');

const rl = readline.createInterface({

    input: process.stdin,

    output: process.stdout

});

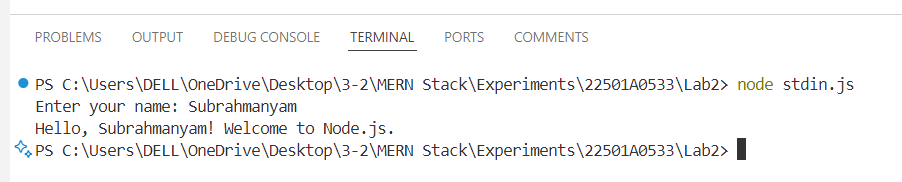
rl.question("Enter your name: ", (name) => {

    console.log(`Hello, ${name}! Welcome to Node.js.`);

    rl.close();

});

**Output:**

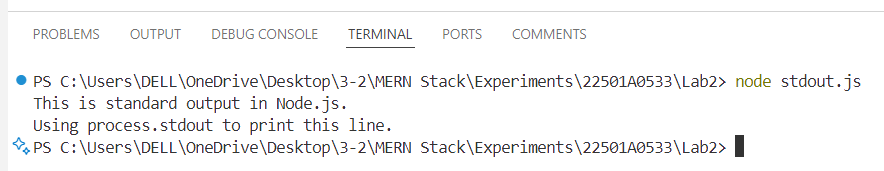


**4. Standard Output (Printing to Console)**

console.log("This is standard output in Node.js.");

process.stdout.write("Using process.stdout to print this line.\n");

**Output:**



**Calculator Application using Node.js**

const readline = require('readline');

// Create a readline interface for user input

const rl = readline.createInterface({

    input: process.stdin,

    output: process.stdout

});

// Function to display the menu and handle user choice

function showMenu() {

    console.log("\n======================================");

    console.log("    Welcome to the Node.js Calculator ");

    console.log("======================================");

    console.log("1. Age Calculator");

    console.log("2. BMI Calculator");

    console.log("3. Simple Interest Calculator");

    console.log("4. Tax Calculator");

    console.log("5. Exit");

    rl.question("\nEnter your choice (1-5): ", (choice) => {

        if (choice === "1") {

            // Age Calculator

            rl.question("\nEnter your birth year: ", (birthYear) => {

                const currentYear = new Date().getFullYear();

                const age = currentYear - parseInt(birthYear);

                console.log(`\nYou are ${age} years old.`);

                showMenu(); // Show menu again

            });

        } else if (choice === "2") {

            // BMI Calculator

            rl.question("\nEnter your weight in kg: ", (weight) => {

                rl.question("Enter your height in meters: ", (height) => {

                    const bmi = parseFloat(weight) / (parseFloat(height) \* parseFloat(height));

                    console.log(`\nYour BMI is ${bmi.toFixed(2)}.`);

                    if (bmi < 18.5) console.log("You are underweight.");

                    else if (bmi >= 18.5 && bmi < 24.9) console.log("You have a normal weight.");

                    else console.log("You are overweight.");

                    showMenu(); // Show menu again

                });

            });

        } else if (choice === "3") {

            // Simple Interest Calculator

            rl.question("\nEnter principal amount: ", (principal) => {

                rl.question("Enter rate of interest per year: ", (rate) => {

                    rl.question("Enter time in years: ", (time) => {

                        const interest = (parseFloat(principal) \* parseFloat(rate) \* parseFloat(time)) / 100;

                        console.log(`\nSimple Interest: ₹${interest.toFixed(2)}`);

                        showMenu(); // Show menu again

                    });

                });

            });

        } else if (choice === "4") {

            // Tax Calculator

            rl.question("\nEnter your annual salary in ₹: ", (salary) => {

                salary = parseFloat(salary);

                let tax = 0;

                if (salary <= 250000) {

                    tax = 0;

                } else if (salary > 250000 && salary <= 500000) {

                    tax = (salary - 250000) \* 0.05;

                } else if (salary > 500000 && salary <= 1000000) {

                    tax = (250000 \* 0.05) + ((salary - 500000) \* 0.2);

                } else {

                    tax = (250000 \* 0.05) + (500000 \* 0.2) + ((salary - 1000000) \* 0.3);

                }

                console.log(`\nYour income tax amount: ₹${tax.toFixed(2)}`);

                showMenu(); // Show menu again

            });

        } else if (choice === "5") {

            console.log("\nExiting application...");

            rl.close(); // Exit the application

        } else {

            console.log("\nInvalid choice! Please enter a number between 1 and 5.");

            showMenu(); // Show menu again

        }

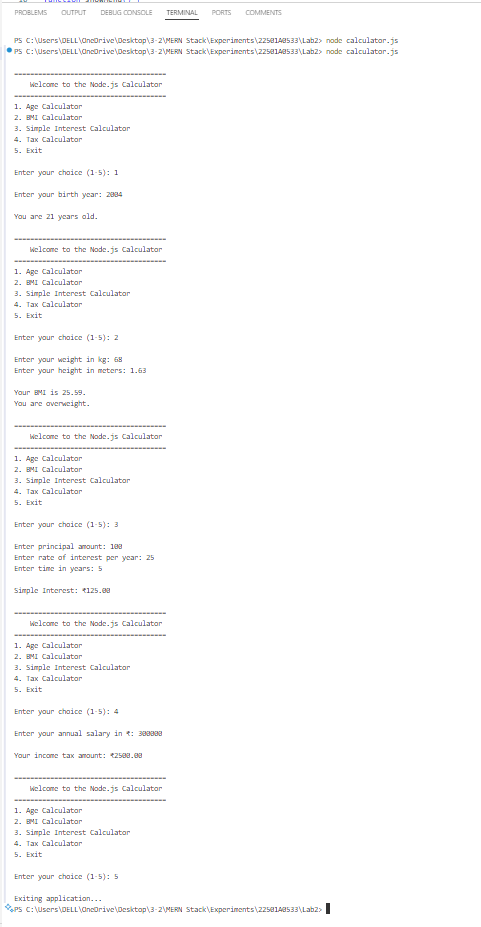
    });

}

// Start the application

showMenu();

**Output:**



**WRITE A PROGRAM TO DEMONSTRATE THE PUBLISHING AND INSTALLATION OF NPM PACKAGE INTO NPM REGISTRY**

**Aim:**

To write a node package program and demonstrate publishing and installing of the NPM packge into NPM registry.

**Description:**

**Installing Node.js**

* Download Node.js from the official website ([nodejs.org](https://nodejs.org/)) and install it.
* Choose the LTS (Long-Term Support) version for stability.

**Verifying npm Installation**

npm -v

* If installed correctly, this command will return the npm version.

**What is npm?**

* Node Package Manager (npm) is a tool for managing JavaScript packages.
* It allows installation, updating, and removal of dependencies for a Node.js project.

**Initializing a Node.js Project (npm init)**

npm init

* This command creates a package.json file that stores project metadata and dependencies.
* Use npm init -y to generate the file with default values.

**package.json**

* It contains information about the project, including dependencies, scripts, and metadata.
* Essential fields include name, version, dependencies, and scripts.

**Node Package Registry**

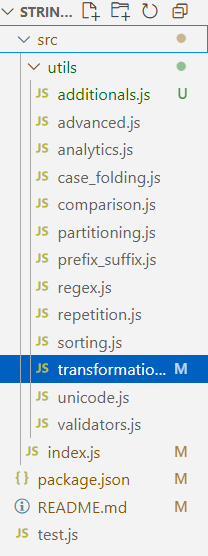
* A repository of open-source Node.js packages.
* Developers can publish and share reusable modules.

**Node Package Manager Commands**

* **npm pack**: Creates a .tgz file for distribution.
* **npm publish**: Publishes the package to the npm registry.
* **npm unpublish**: Removes a published package (only within 72 hours).
* **npm login**: Authenticates a user for publishing packages.

**Program 1:**

**Create an publish an NPM module.**

****

Structure of files

**index.js:**

const validators = require('./utils/validators');

const transformations = require('./utils/transformations');

const analytics = require('./utils/analytics');

const unicode = require('./utils/unicode');

const regex = require('./utils/regex');

const repetition = require('./utils/repetition');

const sorting = require('./utils/sorting');

const partitioning = require('./utils/partitioning');

const prefixSuffix = require('./utils/prefix\_suffix');

const caseFolding = require('./utils/case\_folding');

const advanced = require('./utils/advanced');

const comparison = require('./utils/comparison');

const additionals = require('./utils/additionals');

module.exports = {

  ...validators,

  ...transformations,

  ...analytics,

  ...unicode,

  ...regex,

  ...repetition,

  ...sorting,

  ...partitioning,

  ...prefixSuffix,

  ...caseFolding,

  ...advanced,

  ...comparison,

  ...additionals,

};

**validators.js**

// Validators for strings

module.exports = {

    isEmail: (str) => /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(str),

    isURL: (str) =>

      /^(https?:\/\/)?([\w\d-]+\.){1,}\w{2,}(\/[\w\d-\_.]\*)\*\/?$/.test(str),

    isPalindrome: (str) => {

      const cleaned = str.replace(/[^a-zA-Z0-9]/g, '').toLowerCase();

      return cleaned === cleaned.split('').reverse().join('');

    },

    isAlphanumeric: (str) => /^[a-zA-Z0-9]+$/.test(str),

    isDigit: (str) => /^\d+$/.test(str),

    isSpace: (str) => /^\s+$/.test(str),

    passwordStrength: (password) => {

      if (password.length < 8) return 'Weak';

      if (!/[A-Z]/.test(password)) return 'Medium';

      if (!/[0-9]/.test(password)) return 'Medium';

      if (!/[!@#$%^&\*]/.test(password)) return 'Medium';

      return 'Strong';

    },

  };

**transformations.js**

// String transformations

module.exports = {

    toUpperCase: (str) => str.toUpperCase(),

    toLowerCase: (str) => str.toLowerCase(),

    reverse: (str) => str.split('').reverse().join(''),

    concatenate: (...args) => args.join(''),

    replace: (str, search, replace) => str.split(search).join(replace),

    trim: (str) => str.trim(),

    padStart: (str, length, char = ' ') => str.padStart(length, char),

    padEnd: (str, length, char = ' ') => str.padEnd(length, char),

    caseConversion: (str, targetCase) => {

      if (targetCase === 'camel') {

        return str

          .toLowerCase()

          .replace(/[^a-zA-Z0-9]+(.)/g, (\_, char) => char.toUpperCase());

      }

      if (targetCase === 'snake') return str.replace(/\s+/g, '\_').toLowerCase();

      if (targetCase === 'kebab') return str.replace(/\s+/g, '-').toLowerCase();

      return str;

    },

  };

  module.exports = {

    ...module.exports,

    // Encoding and Decoding

    encodeBase64: (str) => Buffer.from(str).toString('base64'),

    decodeBase64: (str) => Buffer.from(str, 'base64').toString('utf-8'),

    encodeURI: (str) => encodeURIComponent(str),

    decodeURI: (str) => decodeURIComponent(str),

    // Padding

    leftPad: (str, length, char = ' ') => str.padStart(length, char),

    rightPad: (str, length, char = ' ') => str.padEnd(length, char),

    centerPad: (str, length, char = ' ') => {

      const totalPadding = length - str.length;

      const left = Math.floor(totalPadding / 2);

      const right = totalPadding - left;

      return char.repeat(left) + str + char.repeat(right);

    },

  };

**analytics.js**

// String analytics

module.exports = {

    getLength: (str) => str.length,

    countWords: (str) => str.trim().split(/\s+/).length,

    countCharacters: (str) => str.length,

    characterFrequency: (str) => {

      const frequency = {};

      for (const char of str) {

        frequency[char] = (frequency[char] || 0) + 1;

      }

      return frequency;

    },

    startsWith: (str, search) => str.startsWith(search),

    endsWith: (str, search) => str.endsWith(search),

    substring: (str, start, end) => str.substring(start, end),

    split: (str, delimiter) => str.split(delimiter),

    join: (array, delimiter) => array.join(delimiter),

  };

**unicode.js**

// Unicode operations

module.exports = {

    charAt: (str, index) => str.charAt(index),

    charCodeAt: (str, index) => str.charCodeAt(index),

    fromCharCode: (...codes) => String.fromCharCode(...codes),

    toUnicode: (str) => str.split('').map((char) => `\\u${char.charCodeAt(0).toString(16).padStart(4, '0')}`).join(''),

    fromUnicode: (unicodeStr) =>

      unicodeStr.replace(/\\u([\dA-Fa-f]{4})/g, (\_, hex) => String.fromCharCode(parseInt(hex, 16))),

  };

**regex.js**

// Regex utilities

module.exports = {

    match: (str, regex) => str.match(regex),

    replaceRegex: (str, regex, replacement) => str.replace(regex, replacement),

    testRegex: (str, regex) => regex.test(str),

    extractEmails: (str) => str.match(/[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}/g) || [],

    extractURLs: (str) => str.match(/https?:\/\/[^\s/$.?#].[^\s]\*/g) || [],

  };

**repetition.js**

// String repetition

module.exports = {

    repeatString: (str, times) => str.repeat(times),

    generateCaptcha: (length = 6) => {

      const chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';

      let captcha = '';

      for (let i = 0; i < length; i++) {

        captcha += chars.charAt(Math.floor(Math.random() \* chars.length));

      }

      return captcha;

    },

  };

**sorting.js**

// Sorting utilities

module.exports = {

    sortCharacters: (str, order = 'asc') => {

      const sorted = [...str].sort((a, b) => (order === 'asc' ? a.localeCompare(b) : b.localeCompare(a)));

      return sorted.join('');

    },

    sortWords: (str, order = 'asc') => {

      const words = str.split(/\s+/);

      const sorted = words.sort((a, b) => (order === 'asc' ? a.localeCompare(b) : b.localeCompare(a)));

      return sorted.join(' ');

    },

    uniqueCharacters: (str) => {

      return [...new Set(str)].join('');

    },

  };

**partitioning.js**

// Partitioning utilities

module.exports = {

    partition: (str, delimiter) => {

      const index = str.indexOf(delimiter);

      if (index === -1) return [str, '', ''];

      return [str.slice(0, index), delimiter, str.slice(index + delimiter.length)];

    },

    rpartition: (str, delimiter) => {

      const index = str.lastIndexOf(delimiter);

      if (index === -1) return [str, '', ''];

      return [str.slice(0, index), delimiter, str.slice(index + delimiter.length)];

    },

  };

**prefix\_suffix.js**

// Prefix and Suffix Utilities

module.exports = {

    removePrefix: (str, prefix) => (str.startsWith(prefix) ? str.slice(prefix.length) : str),

    removeSuffix: (str, suffix) => (str.endsWith(suffix) ? str.slice(0, -suffix.length) : str),

  };

**case\_folding.js**

// Case folding and normalization

module.exports = {

    caseFold: (str) => str.toLocaleLowerCase(),

    normalize: (str, form = 'NFC') => str.normalize(form),

  };

**advanced.js**

module.exports = {

    // String Translation

    translate: (str, mapping) => {

      const translated = [...str].map((char) => mapping[char] || char).join('');

      return translated;

    },

    // Expand Tabs

    expandTabs: (str, tabSize = 4) => str.replace(/\t/g, ' '.repeat(tabSize)),

    // Character Iteration

    iterateCharacters: (str) => [...str],

    // Indexing and Slicing

    sliceString: (str, start, end) => str.slice(start, end),

    getIndex: (str, index) => (index >= 0 && index < str.length ? str[index] : undefined),

    // String Partitioning

    splitAt: (str, index) => [str.slice(0, index), str.slice(index)],

    // Multibyte Encoding (UTF-8)

    encodeToBytes: (str) => new TextEncoder().encode(str),

    decodeFromBytes: (bytes) => new TextDecoder().decode(bytes),

  };

**comparision.js**

// String comparison utilities

module.exports = {

    equalsIgnoreCase: (str1, str2) => str1.toLowerCase() === str2.toLowerCase(),

    compareLexicographically: (str1, str2) => str1.localeCompare(str2),

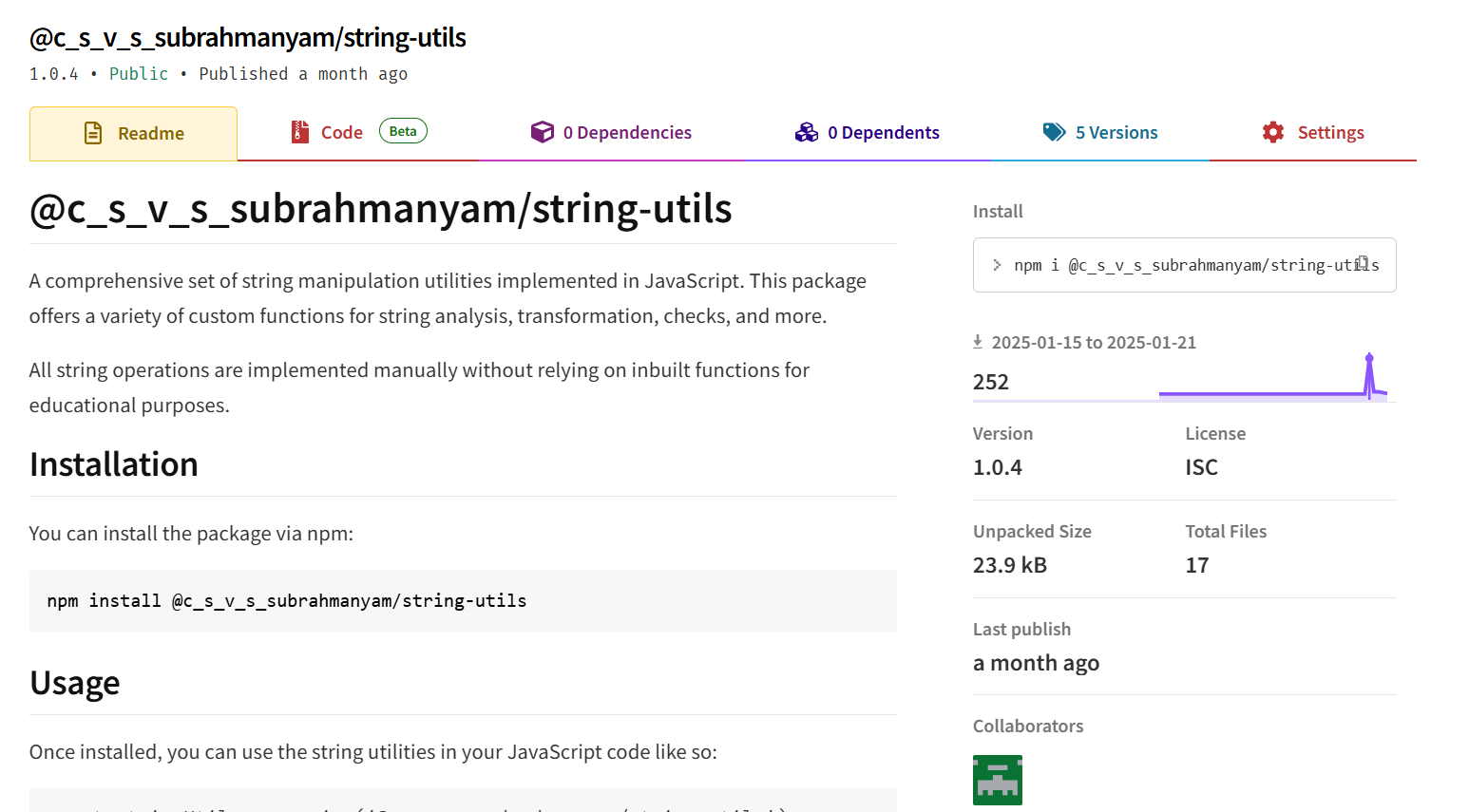
    startsWith: (str, prefix) => str.startsWith(prefix),

    endsWith: (str, suffix) => str.endsWith(suffix),

    containsSubstring: (str, substring) => str.includes(substring),

  };

**Output:**

****

Run the command in terminal to use the npm package

Testing the package and using their functions

**test.js**

// Update 1 - core functionality

const stringUtils = require('./src/index');

console.log(stringUtils.isEmail('test@example.com')); // true

console.log(stringUtils.reverse('hello')); // olleh

console.log(stringUtils.passwordStrength('P@ssw0rd123')); // Strong

console.log(stringUtils.getLength('hello world')); // 11

console.log(stringUtils.startsWith('hello', 'he')); // true

//Update 2

// Test Unicode Operations

console.log(stringUtils.toUnicode('hello')); // \u0068\u0065\u006c\u006c\u006f

console.log(stringUtils.fromUnicode('\\u0068\\u0065\\u006c\\u006c\\u006f')); // hello

// Test Encoding and Decoding

console.log(stringUtils.encodeBase64('hello')); // aGVsbG8=

console.log(stringUtils.decodeBase64('aGVsbG8=')); // hello

// Test Regex Utilities

console.log(stringUtils.extractEmails('Contact us at support@example.com.')); // [ 'support@example.com' ]

console.log(stringUtils.extractURLs('Visit https://example.com now!')); // [ 'https://example.com' ]

// Test Repetition and Captcha

console.log(stringUtils.repeatString('ha', 3)); // hahaha

console.log(stringUtils.generateCaptcha(8)); // Random 8-character captcha

console.log(stringUtils.sortCharacters('dcba')); // abcd

console.log(stringUtils.sortWords('banana apple cherry')); // apple banana cherry

console.log(stringUtils.uniqueCharacters('aabbcc')); // abc

// Test Partitioning

console.log(stringUtils.partition('hello world', ' ')); // ['hello', ' ', 'world']

console.log(stringUtils.rpartition('hello world hello', 'hello')); // ['hello world ', 'hello', '']

// Test Prefix/Suffix Operations

console.log(stringUtils.removePrefix('unhappy', 'un')); // happy

console.log(stringUtils.removeSuffix('baking', 'ing')); // bak

// Test Case Folding

console.log(stringUtils.caseFold('Straße')); // straße

console.log(stringUtils.normalize('\u1E9B\u0323', 'NFC')); // ẛ̣

// Update 3

console.log(stringUtils.translate('hello', { h: 'H', e: '3', l: '1', o: '0' })); // H3110

// Test Expand Tabs

console.log(stringUtils.expandTabs('Hello\tWorld', 8)); // Hello   World

// Test Character Iteration

console.log(stringUtils.iterateCharacters('abc')); // [ 'a', 'b', 'c' ]

// Test Indexing and Slicing

console.log(stringUtils.sliceString('hello', 1, 4)); // ell

console.log(stringUtils.getIndex('hello', 2)); // l

console.log(stringUtils.splitAt('hello', 2)); // [ 'he', 'llo' ]

// Test Multibyte Encoding

const bytes = stringUtils.encodeToBytes('hello');

console.log(bytes); // Uint8Array([...])

console.log(stringUtils.decodeFromBytes(bytes)); // hello

// Test Comparison Utilities

console.log(stringUtils.equalsIgnoreCase('Hello', 'hello')); // true

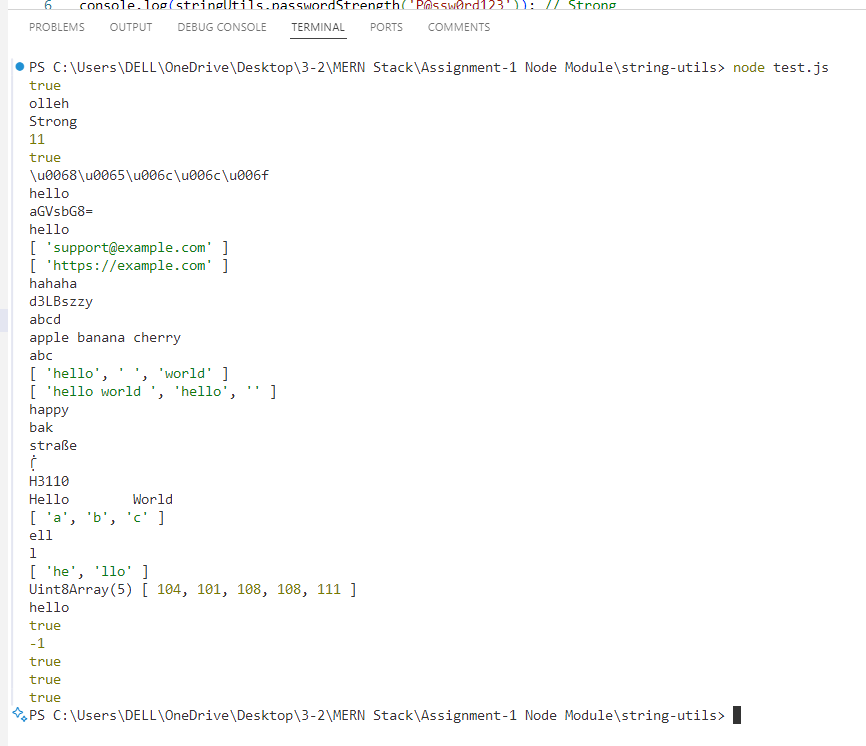
console.log(stringUtils.compareLexicographically('abc', 'abd')); // -1

console.log(stringUtils.startsWith('hello', 'he')); // true

console.log(stringUtils.endsWith('hello', 'lo')); // true

console.log(stringUtils.containsSubstring('hello world', 'world'));

**Output:**

****

**EXPERIMENT 3**

**DEVELOP NODE JS APPLICATION AND IMPLEMENT HTTP SERVICES IN NODE JS (REQUEST AND RESPONSE)**

**Aim:**

To develop a **Node.js application** that implements HTTP services, handling **requests and responses** using the built-in **HTTP module**.

**Theory:**

**Introduction to Node.js and HTTP Services**

Node.js is a **server-side runtime environment** that allows developers to build scalable network applications using JavaScript. It provides an in-built **HTTP module** to create web servers, handle requests, and send responses.

**Key Concepts:**

1. **HTTP Module (http)**
   * The http module in Node.js is used to create web servers and handle client requests.
   * It provides methods like http.createServer() to set up a server.
2. **Request and Response Objects (req, res)**
   * req (Request Object): Contains details about the client’s request (method, URL, headers, etc.).
   * res (Response Object): Used to send responses back to the client.
3. **Creating an HTTP Server**
   * A basic server listens on a specific **port** and responds based on the request URL.
4. **Handling Different HTTP Methods**
   * GET: Used to retrieve data from the server.
   * POST: Used to send data to the server.
5. **Client-Server using Express.js**

* Express.js is a lightweight framework for handling HTTP requests.
* Provides middleware, routing, and request handling capabilities.
* Simplifies the creation of RESTful APIs.

**Program 1:**

**HTTP client and request**

**server.js**

const http = require("http");

const server = http.createServer((req, res) => {

    res.setHeader("Content-Type", "application/json");

    if (req.method === "GET" && req.url === "/") {

        console.log("request to GET")

        res.writeHead(200);

        res.end(JSON.stringify({ message: "GET request received" }));

    }

    else if (req.method === "POST" && req.url === "/") {

        console.log("request to POST")

        let body = "";

        req.on("data", chunk => {

            body += chunk.toString();

        });

        req.on("end", () => {

            res.writeHead(201);

            res.end(JSON.stringify({ message: "POST request received", data: JSON.parse(body) }));

        });

    }

    else if (req.method === "PUT" && req.url === "/") {

        console.log("request to PUT")

        let body = "";

        req.on("data", chunk => {

            body += chunk.toString();

        });

        req.on("end", () => {

            res.writeHead(200);

            res.end(JSON.stringify({ message: "PUT request received", data: JSON.parse(body) }));

        });

    }

    else if (req.method === "DELETE" && req.url === "/") {

        console.log("request to DELETE")

        res.writeHead(200);

        res.end(JSON.stringify({ message: "DELETE request received" }));

    }

    else {

        res.writeHead(404);

        res.end(JSON.stringify({ message: "Route not found" }));

    }

});

const PORT = 3000;

server.listen(PORT, () => {

    console.log(`Server running on http://localhost:${PORT}`);

});

**client.js**

const http = require("http");

function sendRequest(method, data = null) {

    const options = {

        hostname: "localhost",

        port: 3000,

        path: "/",

        method: method,

        headers: {

            "Content-Type": "application/json"

        }

    };

    const req = http.request(options, (res) => {

        let responseData = "";

        res.on("data", (chunk) => {

            responseData += chunk;

        });

        res.on("end", () => {

            console.log(`Response for ${method}:`, responseData);

        });

    });

    req.on("error", (error) => {

        console.error(`Error with ${method} request:`, error.message);

    });

    if (data) {

        req.write(JSON.stringify(data));

    }

    req.end();

}

// Sending requests

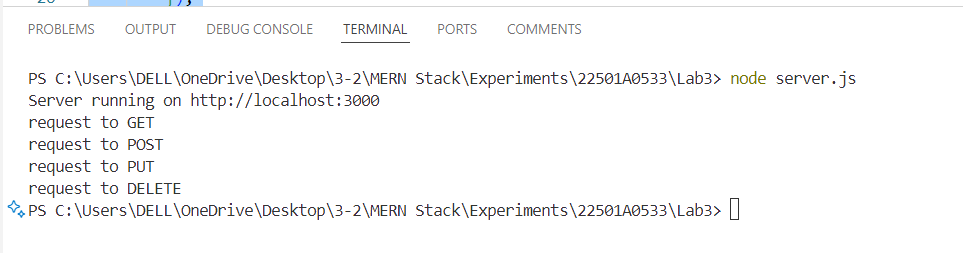
sendRequest("GET");

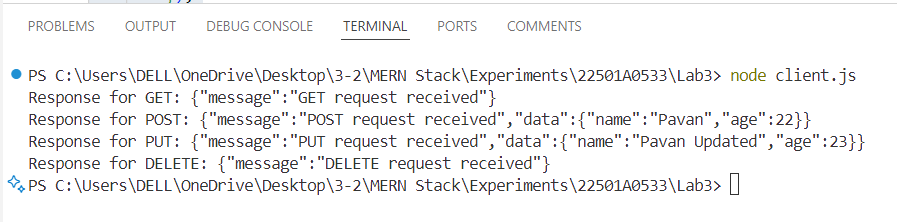
sendRequest("POST", { name: "Pavan", age: 22 });

sendRequest("PUT", { name: "Pavan Updated", age: 23 });

sendRequest("DELETE");

**Output:**

****

****

**Program 2:**

**Demonstrate client server using express.**

**server.js**

const express = require('express');

const app = express();

const port = 3000;

app.get('/',(req,res)=>{

    res.send("used GET method to access the server")

});

app.put('/',(req,res)=>{

    res.send("used PUT method to access the server")

});

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

    res.send("used POST method to access the server")

});

app.listen(port,()=>{

    console.log("Listening to the server " + port)

});

**client.js**

const http = require('http');

// Function for GET request

function getRequest() {

    const options = {

        hostname: 'localhost',

        port: 3000,

        path: '/',

        method: 'GET',

    };

    const req = http.request(options, (res) => {

        let data = '';

        res.setEncoding('utf8');

        res.on('data', (chunk) => {

            data += chunk;

        });

        res.on('end', () => {

            console.log('GET Response from server: used GET method to access the server');

        });

    });

    req.on('error', (err) => {

        console.error('Error during GET request:', err.message);

    });

    req.end();

}

// Function for POST request

function postRequest() {

    const options = {

        hostname: 'localhost',

        port: 3000,

        path: '/',

        method: 'POST',

    };

    const req = http.request(options, (res) => {

        res.on('data', (chunk) => {});

        res.on('end', () => {

            console.log('POST Response from server: used POST method to access the server');

        });

    });

    req.on('error', (err) => {

        console.error('Error during POST request:', err.message);

    });

    req.end();

}

// Function for PUT request

function putRequest() {

    const options = {

        hostname: 'localhost',

        port: 3000,

        path: '/',

        method: 'PUT',

    };

    const req = http.request(options, (res) => {

        res.on('data', (chunk) => {});

        res.on('end', () => {

            console.log('PUT Response from server: used PUT method to access the server');

        });

    });

    req.on('error', (err) => {

        console.error('Error during PUT request:', err.message);

    });

    req.end();

}

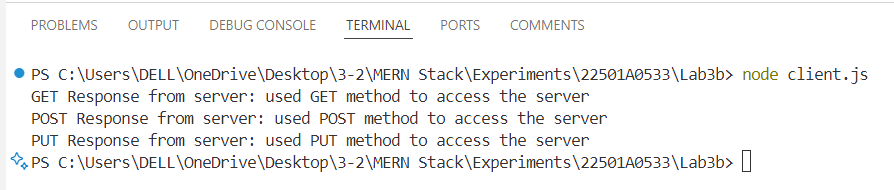
// Making requests

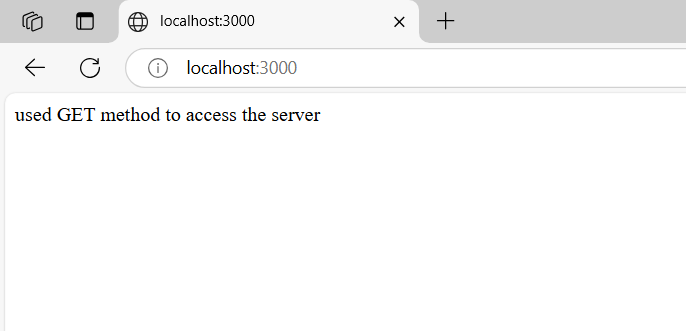
getRequest();

postRequest();

putRequest();

**Output:**

****

****

**DEMONSTRATING EXPRESS ROUTING**

**Aim:**

Write a program to demonstrate Routing in Express

**Theory:**

**Introduction to Express Routing**

Express.js is a lightweight and fast web framework for Node.js that simplifies handling HTTP requests and responses. **Routing** in Express.js allows defining URL paths and associating them with specific handler functions to process client requests efficiently.

**Types of Routes in Express.js**

1. **Basic Routes** – Handle requests to specific URL paths using HTTP methods like GET, POST, PUT, and DELETE.
2. **Parameterized Routes** – Use dynamic parameters in the URL to handle different request variations.
3. **Middleware in Routes** – Process requests before sending a response, useful for authentication or logging.
4. **Route Grouping** – Organizes multiple related routes under a common path for better code structure.

**Importance of Express Routing**

* Organizes the server-side logic for handling different requests.
* Enhances scalability and maintainability of web applications.
* Allows dynamic URL handling with parameters and middleware.

Express routing is essential for developing APIs and full-stack applications, enabling smooth communication between the client and server.

**Program:**

**Express Routing**

// Express Routing

const express = require('express');

// Create an Express application

const app = express();

// Define a route for the root URL

const port = 3000;

app.get('/', (req, res) => {

    res.send('<h1>Welcome to the Express server!</h1>');

});

app.get('/login', (req, res) => {

    res.send('<h2>Welcome to the login page</h2>');

});

app.get('/home', (req, res) => {

    res.send('<h2>Welcome to the home page</h2>');

});

app.get('./contact', (req, res) => {

    res.send('<h2>Welcome to the contact page</h2>');

});

app.get('/about', (req, res) => {

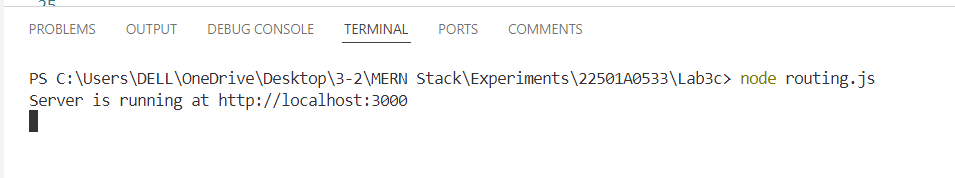
    res.send('<h2>22501A0533. Hi! I am Subrahmanyam.</h2>');

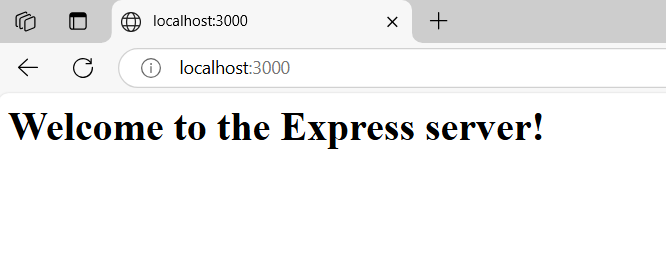
});

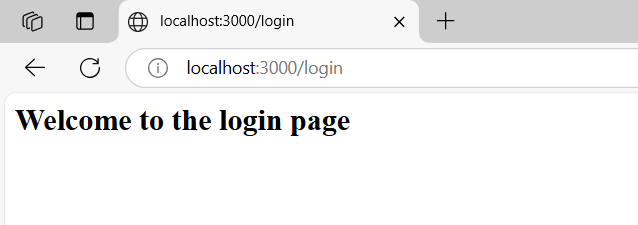
app.listen(port, () => {

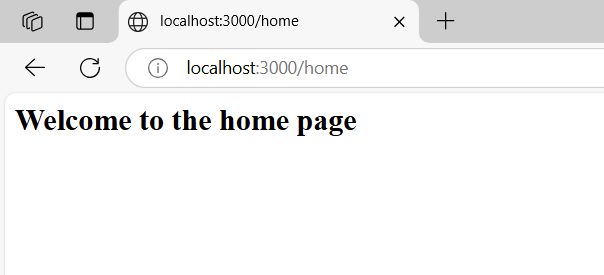
    console.log(`Server is running at http://localhost:${port}`);

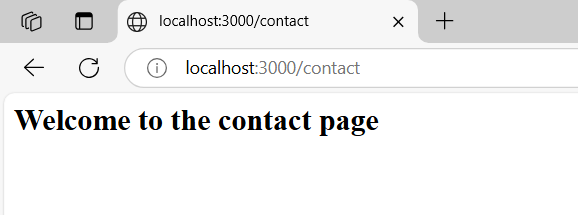
});

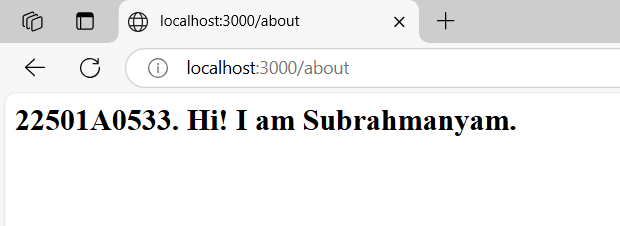
**Output:  
**

****

****

****

****

****

**EXPERIMENT 6**

**READING AND WRITING TO MONGODB DATABASE USING APIS**

**Aim:**

To develop a **Node.js application** using **Express.js** and **MongoDB**, where:

* **Data is written (inserted) into MongoDB** using a POST API.
* **Data is read (fetched) from MongoDB** using a GET API.

**Theory:**

**1️. What is MongoDB?**

MongoDB is a **NoSQL database** that stores data in **JSON-like documents**. It is widely used in modern applications due to its scalability and flexibility.

**2️. What is Express.js?**

Express.js is a lightweight framework for **building web applications and APIs** using Node.js.

**3️. How Does API-Based MongoDB CRUD Work?**

* **Create (POST):** Inserts data into the database.
* **Read (GET):** Fetches data from the database.
* **Update (PUT/PATCH):** Modifies existing data.
* **Delete (DELETE):** Removes data.

**Program:**

const express = require('express');

const mongoose = require('mongoose');

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

const cors = require('cors');

const app = express();

const PORT = 5000;

// Middleware

app.use(cors());

app.use(bodyParser.json());

// Connect to MongoDB

mongoose.connect('mongodb://localhost:27017/ShoppingMall', {

    useNewUrlParser: true,

    useUnifiedTopology: true

}).then(() => console.log("Connected to MongoDB"))

  .catch(err => console.error("MongoDB connection error:", err));

// Define Shop Schema and Model

const shopSchema = new mongoose.Schema({

    name: String,

    category: String,

    location: String,

    items: [{ name: String, price: Number }]

});

const Shop = mongoose.model('Shop', shopSchema);

app.get('/shops', async (req, res) => {

    try {

        const shops = await Shop.find();

        res.json(shops);

    } catch (error) {

        res.status(500).json({ error: "Failed to fetch shops" });

    }

});

app.post('/shops', async (req, res) => {

    try {

        const { name, category, location, items } = req.body;

        const newShop = new Shop({ name, category, location, items });

        await newShop.save();

        res.status(201).json({ message: "Shop added successfully", shop: newShop });

    } catch (error) {

        res.status(500).json({ error: "Failed to add shop" });

    }

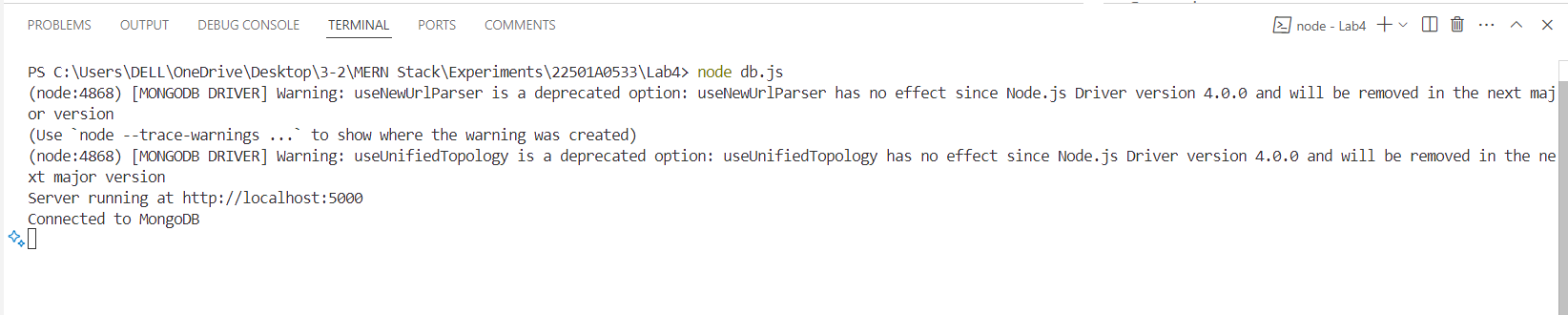
});

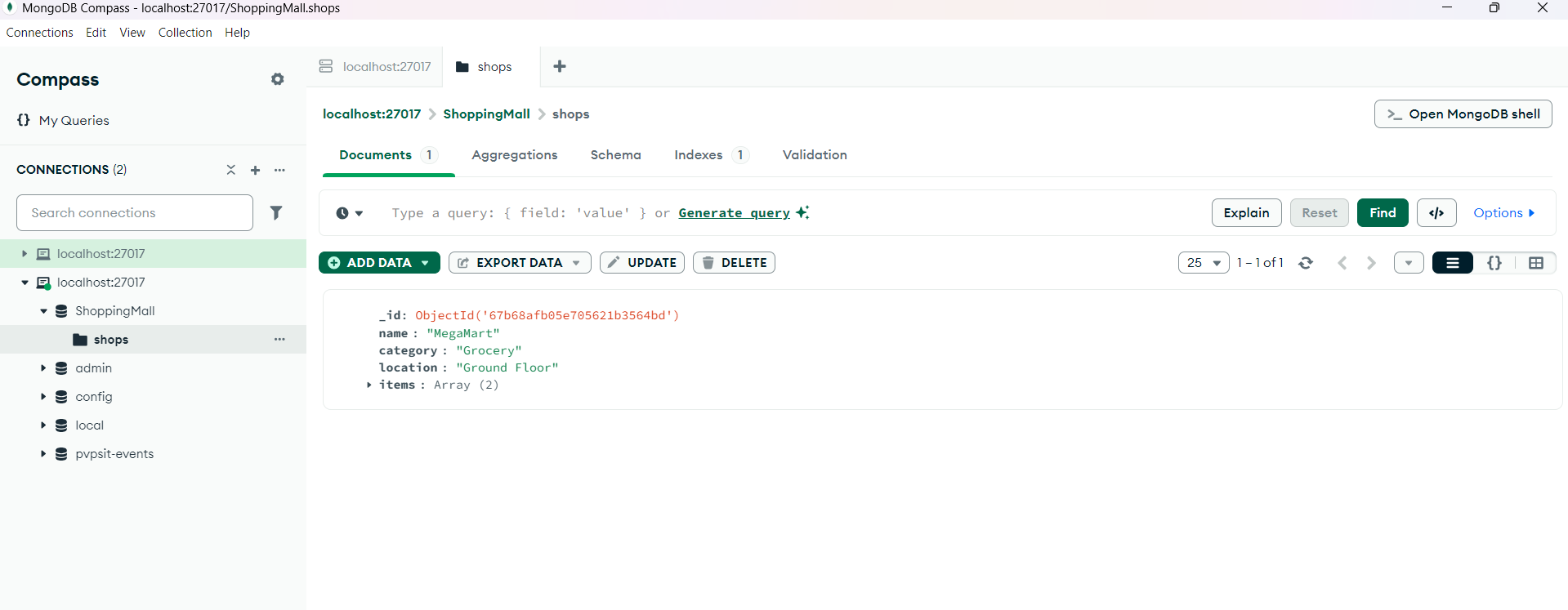
app.listen(PORT, () => {

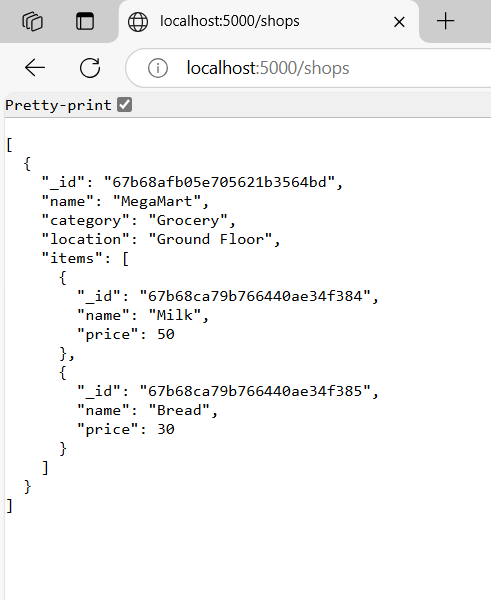
    console.log(`Server running at http://localhost:${PORT}`);

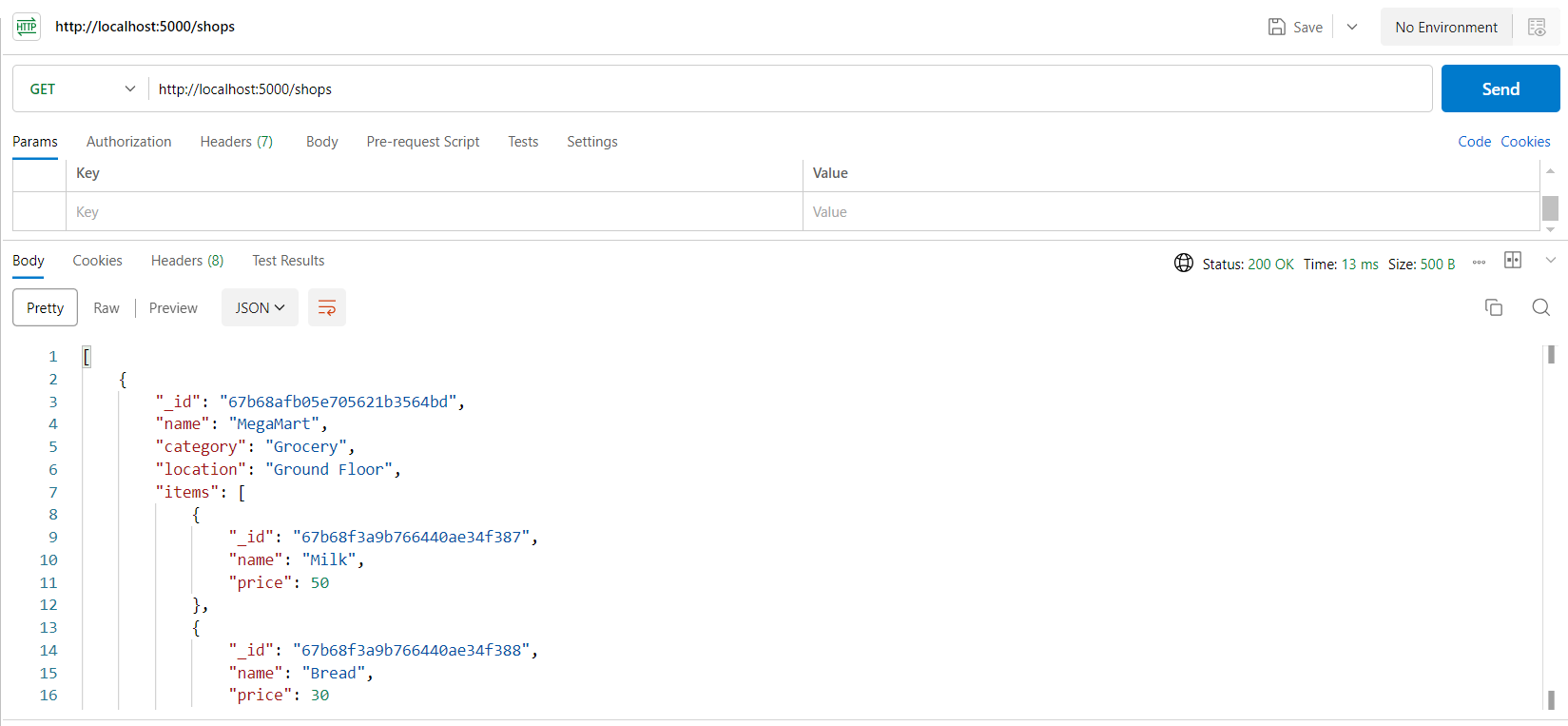
});

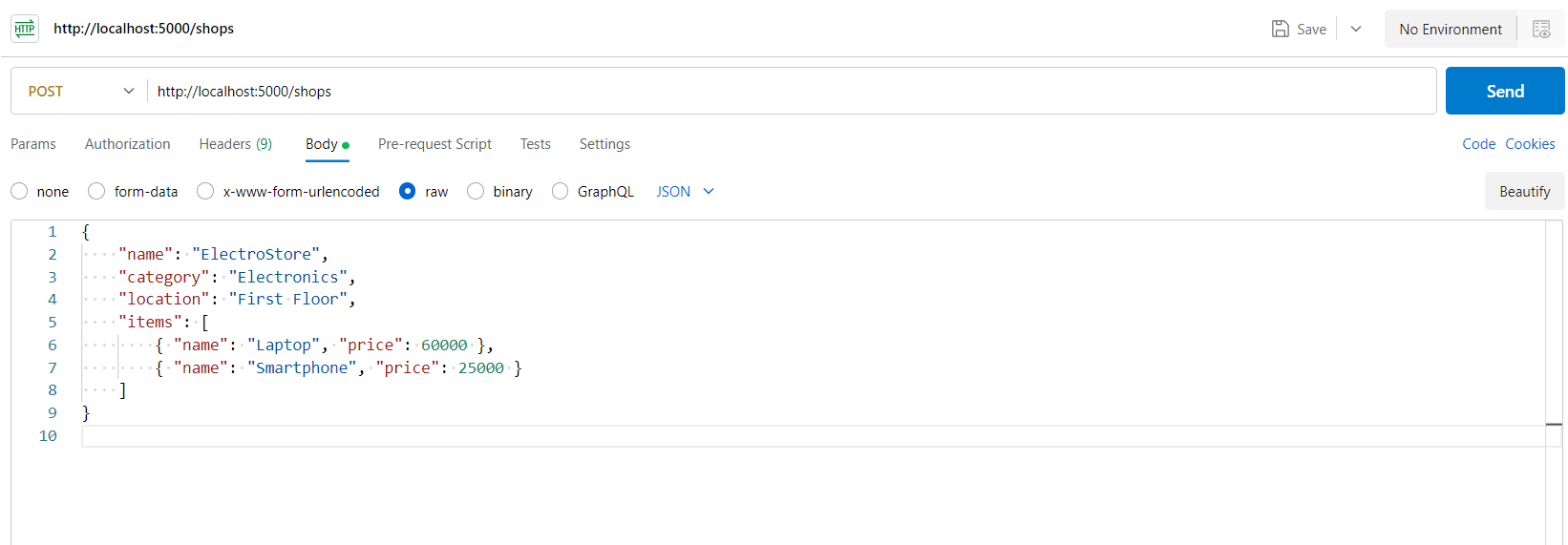
**Output:**

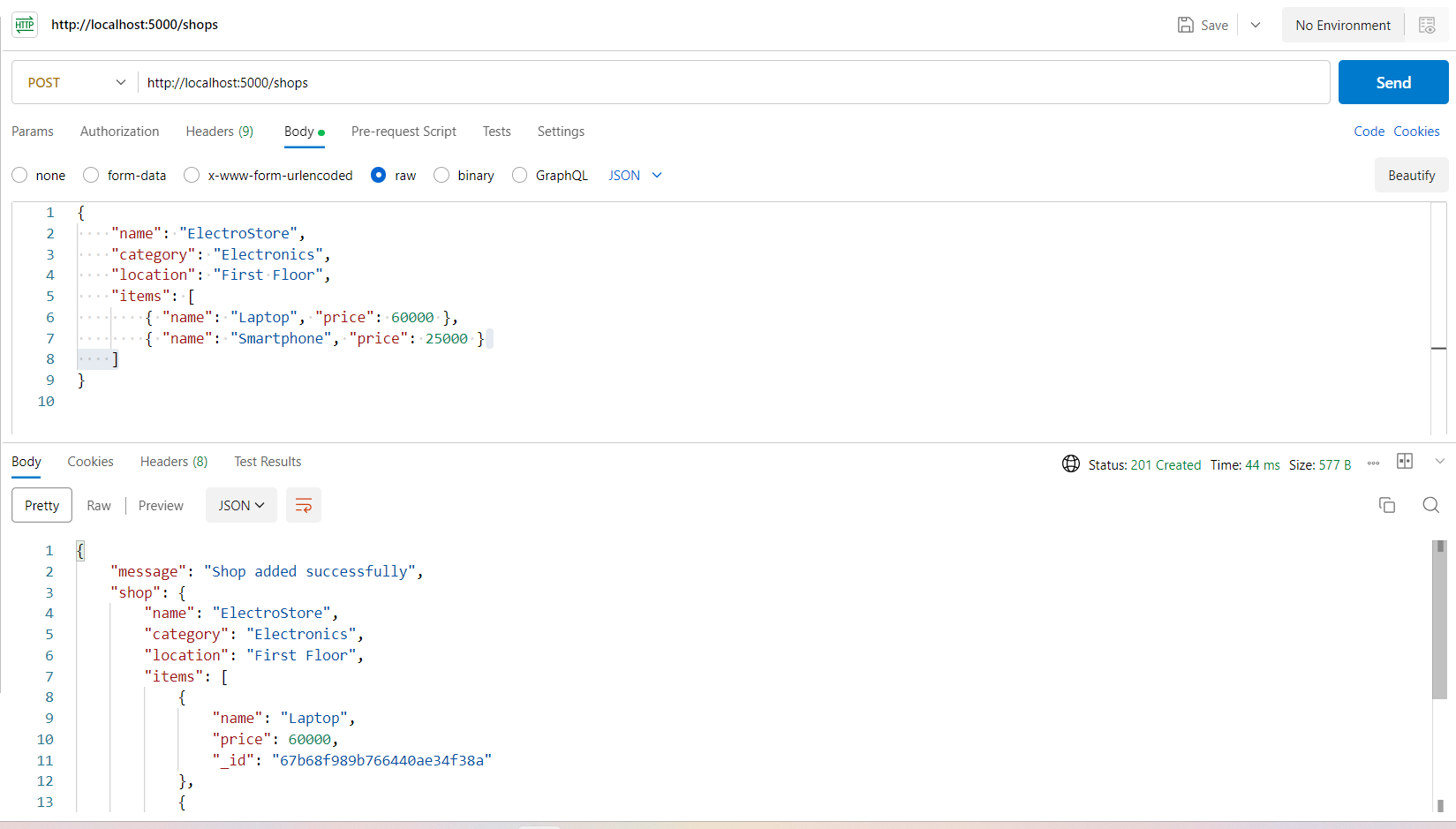
****

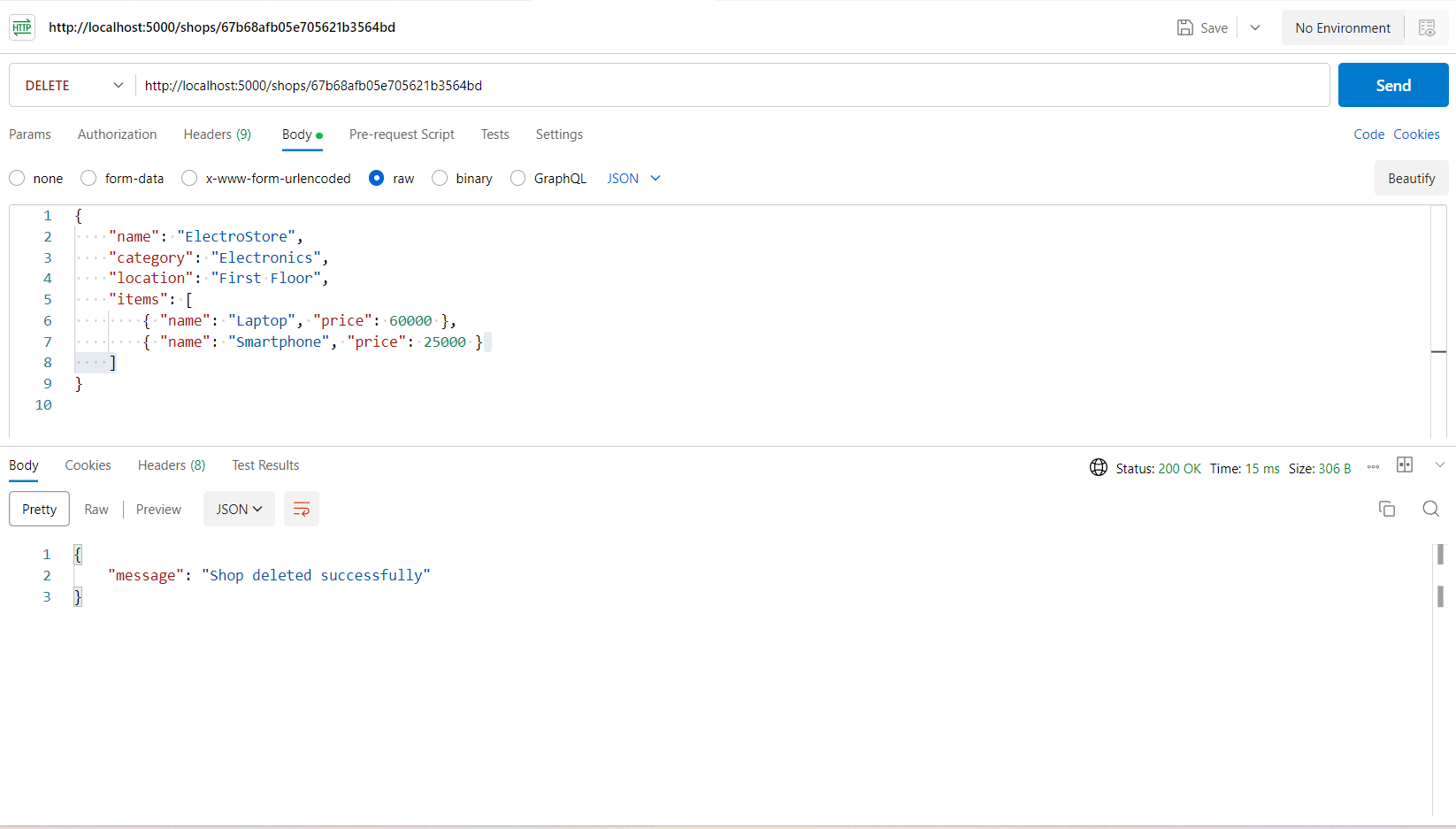
****

****

****

****

****

****

**EXPERIMENT - 7**

**DEVELOPING A SIMPLE CRUD APPLICATION USING THE MERN STACK**

**Aim:**

To develop a **CRUD (Create, Read, Update, Delete) application** using the **MERN (MongoDB, Express.js, React, Node.js) stack**, where users can perform database operations through a web interface.

**Theory:**

**What is the MERN Stack?**

* MongoDB: NoSQL database for storing application data.
* Express.js: Backend framework for handling API requests.
* React.js: Frontend framework for building user interfaces.
* Node.js: JavaScript runtime for executing server-side code.

**Steps to Develop a CRUD Application with MERN**

1. **Setup the Backend (Node.js + Express + MongoDB)**
   * Install Node.js and MongoDB.
   * Create an Express server.
   * Connect to MongoDB using Mongoose.
   * Define a model/schema for the data.
   * Implement API routes for Create (POST), Read (GET), Update (PUT), and Delete (DELETE).
2. **Setup the Frontend (React.js)**
   * Create a React app using create-react-app.
   * Install Axios for API calls.
   * Build components for listing, adding, updating, and deleting data.
   * Use React Router for navigation between pages.
   * Manage state using React hooks (useState, useEffect).
3. **Connect Frontend with Backend**
   * Configure API endpoints in React using Axios.
   * Handle form submissions to create/update data.
   * Display retrieved data in UI components.
4. **Testing the Application**
   * Use Postman to test API requests.
   * Check database updates in MongoDB.
   * Debug and fix any errors in the client-server interaction.
5. **Deploying the Application**
   * Host the backend on platforms like Heroku or Render.
   * Deploy the frontend on Vercel or Netlify.
   * Use MongoDB Atlas for a cloud-based database.

**Program:**

**server.js**

const express = require("express");

const mongoose = require("mongoose");

const cors = require("cors");

const app = express();

const port = 5000;

// Middleware

app.use(cors());

app.use(express.json());

// Connect to MongoDB

mongoose.connect("mongodb://localhost:27017/ShoppingMall", {

  useNewUrlParser: true,

  useUnifiedTopology: true,

});

// Define Schema and Model

const shopSchema = new mongoose.Schema({

  name: String,

  category: String,

  location: String,

  items: [{ name: String, price: Number }],

});

const Shop = mongoose.model("Shop", shopSchema);

// Routes

// Create a new shop (POST)

app.post("/shops", async (req, res) => {

  try {

    const newShop = new Shop(req.body);

    await newShop.save();

    res.status(201).json({ message: "Shop added successfully!" });

  } catch (error) {

    res.status(400).json({ error: error.message });

  }

});

// Get all shops (GET)

app.get("/shops", async (req, res) => {

  try {

    const shops = await Shop.find();

    res.status(200).json(shops);

  } catch (error) {

    res.status(500).json({ error: error.message });

  }

});

// Update a shop (PUT)

app.put("/shops/:id", async (req, res) => {

  try {

    await Shop.findByIdAndUpdate(req.params.id, req.body);

    res.json({ message: "Shop updated successfully!" });

  } catch (error) {

    res.status(400).json({ error: error.message });

  }

});

// Delete a shop (DELETE)

app.delete("/shops/:id", async (req, res) => {

  try {

    await Shop.findByIdAndDelete(req.params.id);

    res.json({ message: "Shop deleted successfully!" });

  } catch (error) {

    res.status(400).json({ error: error.message });

  }

});

// Start Server

app.listen(port, () => {

  console.log(`Server running on http://localhost:${port}`);

});

**index.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Shopping Mall CRUD</title>

    <style>

        body { font-family: Arial, sans-serif; text-align: center; }

        table { width: 80%; margin: auto; border-collapse: collapse; }

        th, td { border: 1px solid black; padding: 10px; }

        input, button { margin: 5px; padding: 8px; }

    </style>

</head>

<body>

    <h2>Shopping Mall Management</h2>

    <!-- Input fields for adding a shop -->

    <input type="text" id="name" placeholder="Shop Name">

    <input type="text" id="category" placeholder="Category">

    <input type="text" id="location" placeholder="Location">

    <button onclick="addShop()">Add Shop</button>

    <h3>List of Shops</h3>

    <!-- Table to display shop data -->

    <table>

        <thead>

            <tr>

                <th>Name</th>

                <th>Category</th>

                <th>Location</th>

                <th>Actions</th>

            </tr>

        </thead>

        <tbody id="shopTable"></tbody>

    </table>

    <script>

        const apiUrl = "http://localhost:5000/shops";

        // Fetch and display all shops

        async function fetchShops() {

            const res = await fetch(apiUrl);

            const data = await res.json();

            const table = document.getElementById("shopTable");

            table.innerHTML = "";

            data.forEach(shop => {

                table.innerHTML += `

                    <tr>

                        <td>${shop.name}</td>

                        <td>${shop.category}</td>

                        <td>${shop.location}</td>

                        <td>

                            <button onclick="editShop('${shop.\_id}')">Edit</button>

                            <button onclick="deleteShop('${shop.\_id}')">Delete</button>

                        </td>

                    </tr>

                `;

            });

        }

        // Add a new shop

        async function addShop() {

            const name = document.getElementById("name").value;

            const category = document.getElementById("category").value;

            const location = document.getElementById("location").value;

            await fetch(apiUrl, {

                method: "POST",

                headers: { "Content-Type": "application/json" },

                body: JSON.stringify({ name, category, location })

            });

            fetchShops();

        }

        // Delete a shop

        async function deleteShop(id) {

            await fetch(`${apiUrl}/${id}`, { method: "DELETE" });

            fetchShops();

        }

        // Edit a shop

        async function editShop(id) {

            const newName = prompt("Enter new shop name:");

            const newCategory = prompt("Enter new category:");

            const newLocation = prompt("Enter new location:");

            await fetch(`${apiUrl}/${id}`, {

                method: "PUT",

                headers: { "Content-Type": "application/json" },

                body: JSON.stringify({ name: newName, category: newCategory, location: newLocation })

            });

            fetchShops();

        }

        // Load shops on page load

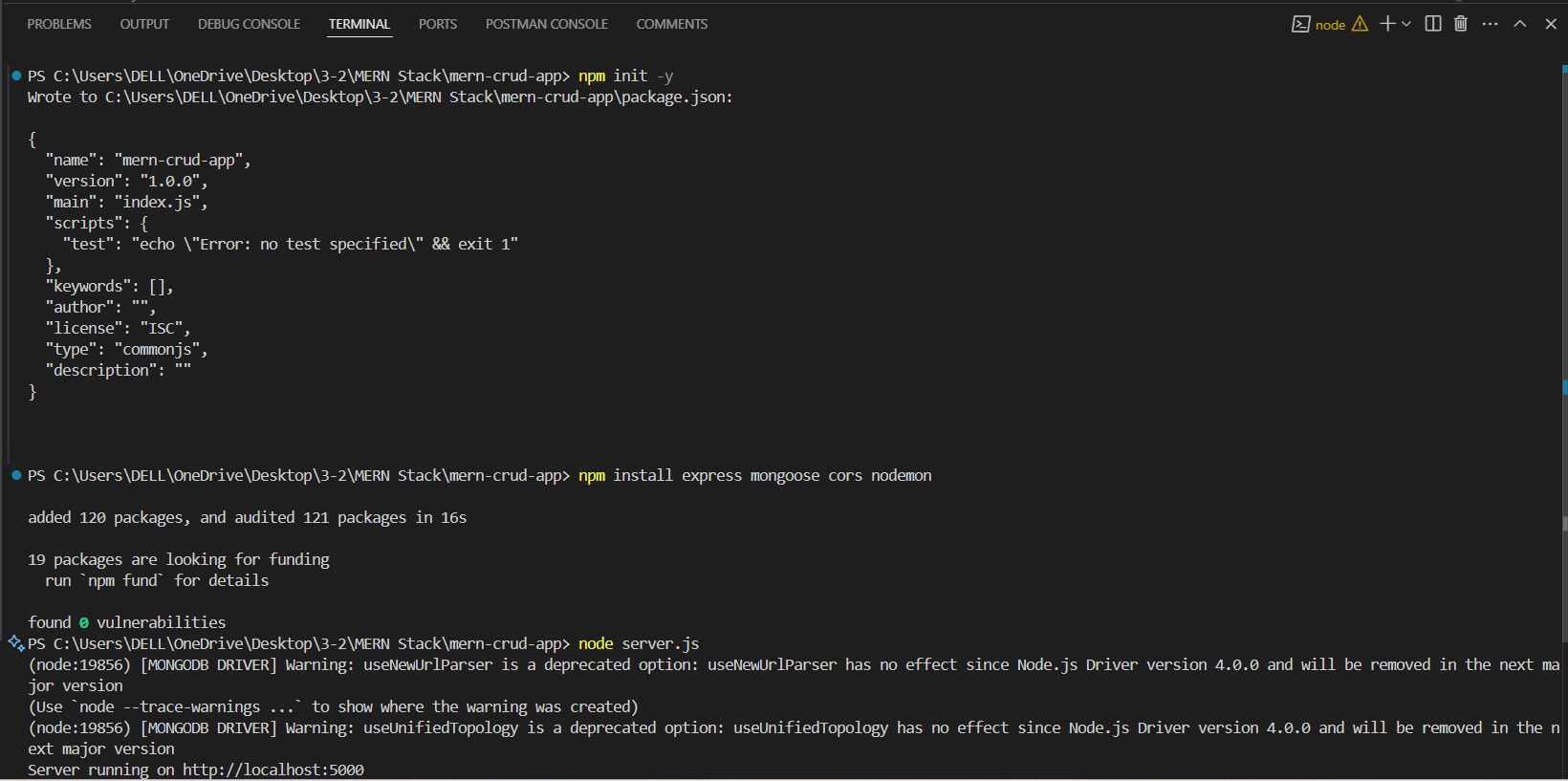
        fetchShops();

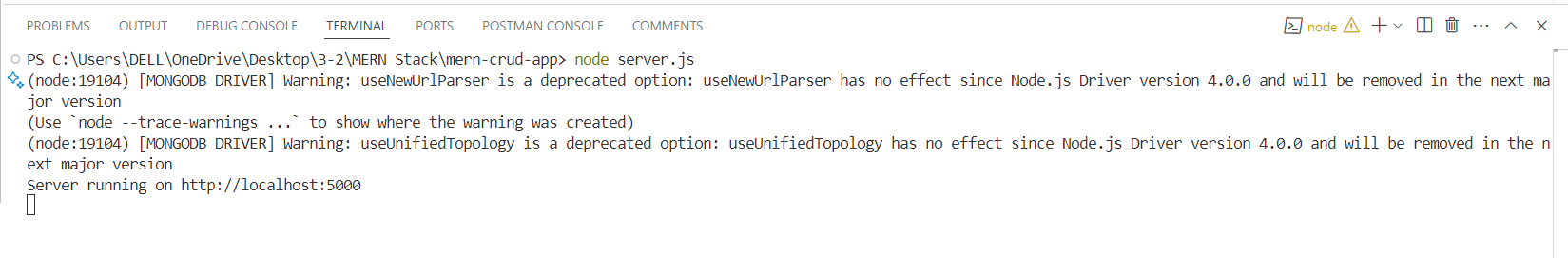
    </script>

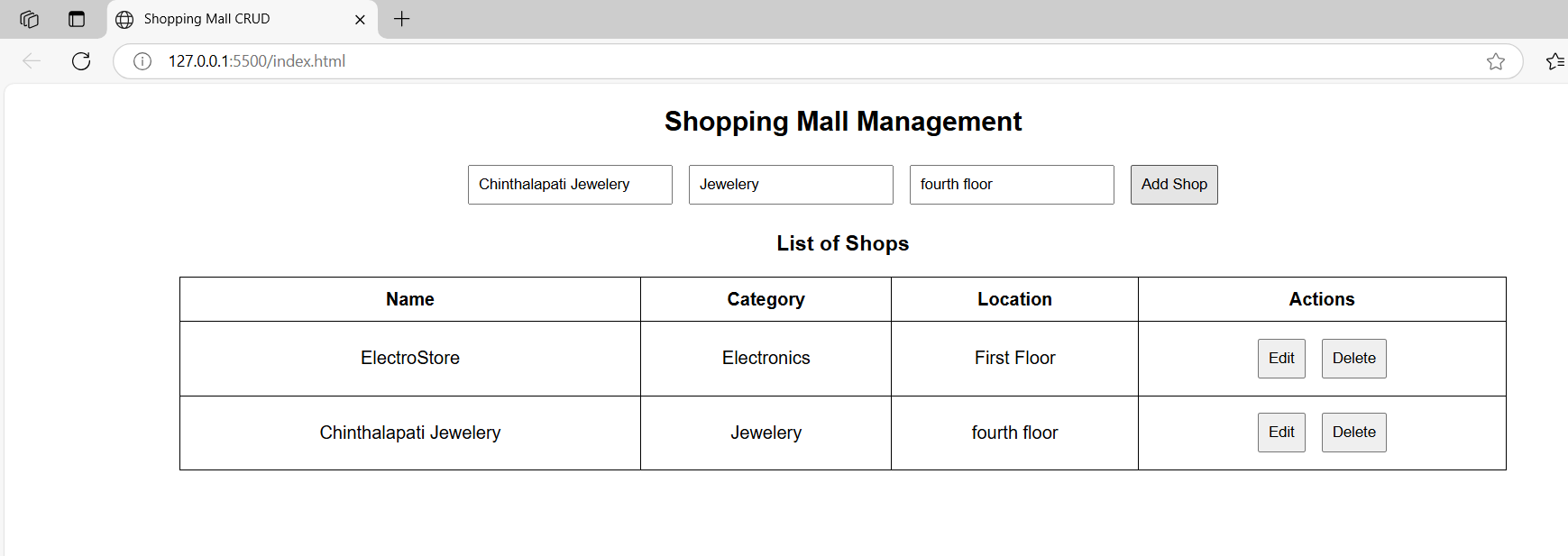
</body>

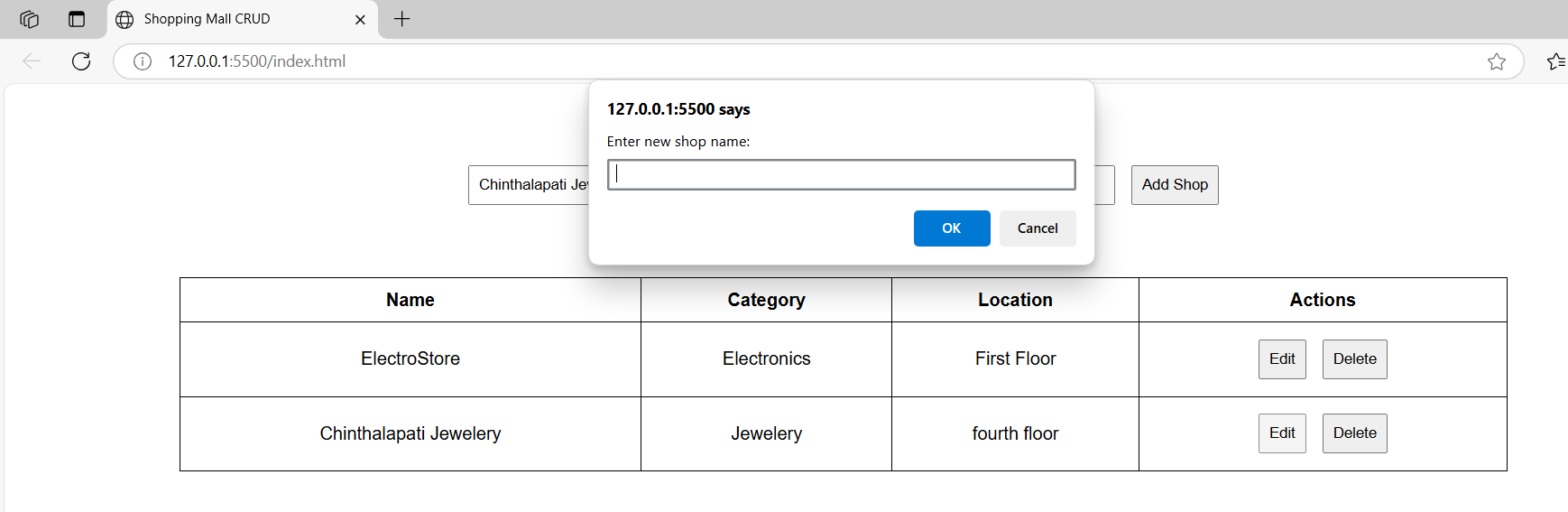
</html>

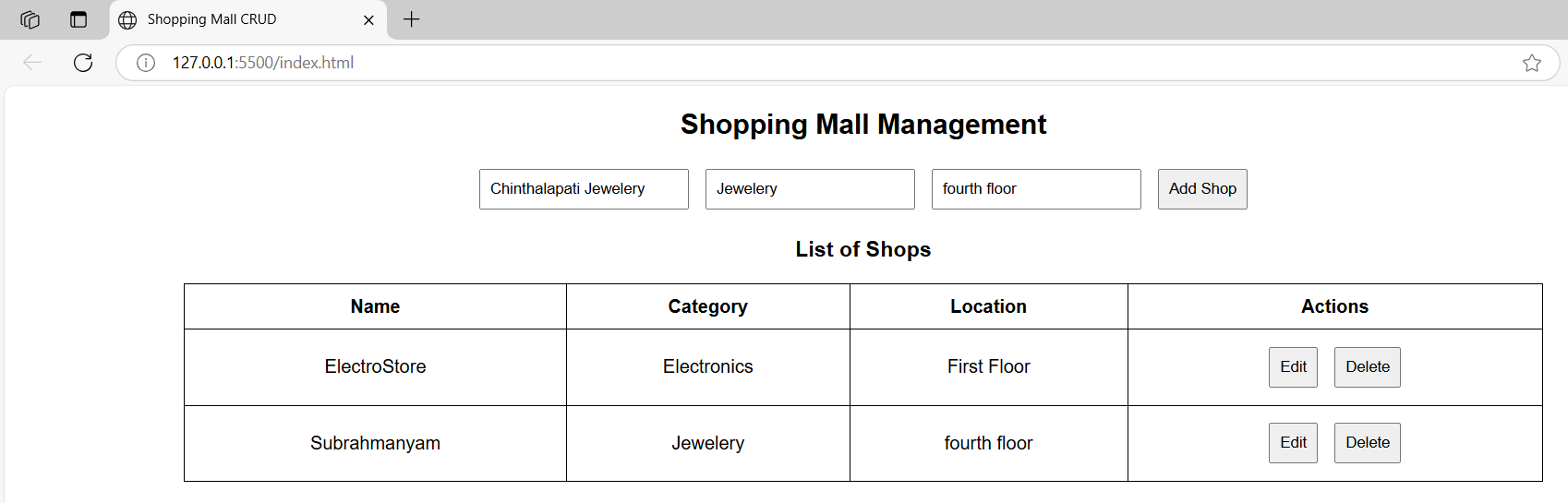
**Output:**

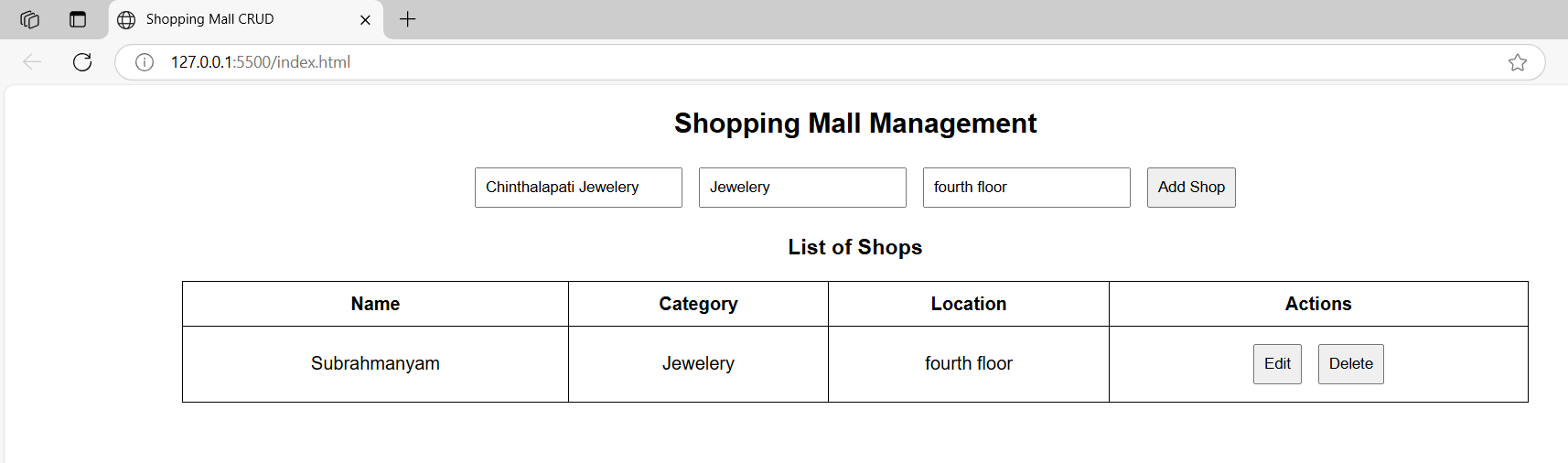
****

****

****

****

****

****

**EXPERIMENT - 8**

**IMPLEMENT REACT ELEMENTS AND COMPONENTS**

**Aim:**

To implement and demonstrate the usage of React elements and components, including functional and class components.

**Theory:**

**React Elements**

* The **smallest building blocks** of a React application.
* Represent **UI elements** such as buttons, headings, paragraphs, or divs.
* Can be created using **React.createElement()** or **JSX syntax** (e.g., <h1>Hello</h1>).
* **Immutable** once created, meaning they cannot be changed after rendering.
* React Elements are responsible for **describing what should appear on the screen**.

**React Components**

* **Reusable UI pieces** that return **React Elements**.
* Help in structuring **complex UIs** by breaking them into smaller parts.
* Two types of components:
  + **Functional Components**: Defined as functions that return JSX, recommended for most cases.
  + **Class Components**: Defined using ES6 classes, primarily used when lifecycle methods are needed.
* Components **can be nested inside other components** to create a **hierarchical UI structure**.
* React components allow **code reusability, better maintainability, and efficient rendering**.

**Program :**

**App.js**

import React from "react";

import Header from "./Header";

import Main from "./Main";

import Footer from "./Footer";

function App() {

    return (

        <div style={{ textAlign: "center", fontFamily: "Arial, sans-serif" }}>

            <Header />

            <Main />

            <Footer />

        </div>

    );

}

export default App;

**Header.js**

import React from "react";

function Header() {

    return (

        <header>

            <h1>React Elements & Components - 22501A0533</h1>

        </header>

    );

}

export default Header;

**WelcomeElement.js**

import React from "react";

const WelcomeElement = () => {

    return React.createElement("h2", {}, "Welcome to React Elements, Subrahmanyam!");

};

export default WelcomeElement;

**FunctionalComponent.js**

import React from "react";

function FunctionalComponent(props) {

    return <h3>Hello, {props.name}! This is a Functional Component.</h3>;

}

export default FunctionalComponent;

**Main.js**

import React from "react";

import WelcomeElement from "./WelcomeElement";

import FunctionalComponent from "./FunctionalComponent";

import ClassComponent from "./ClassComponent";

function Main() {

    return (

        <main>

            <WelcomeElement />

            <FunctionalComponent name="Subrahmanyam" />

            <ClassComponent name="Subrahmanyam" />

        </main>

    );

}

export default Main;

**ClassComponent.js**

import React, { Component } from "react";

class ClassComponent extends Component {

    render() {

        return <h3>Hello, {this.props.name}! This is a Class Component.</h3>;

    }

}

export default ClassComponent;

**Footer.js**

import React from "react";

function Footer() {

    return (

        <footer>

            <p>© 2024 React Elements & Components - Subrahmanyam</p>

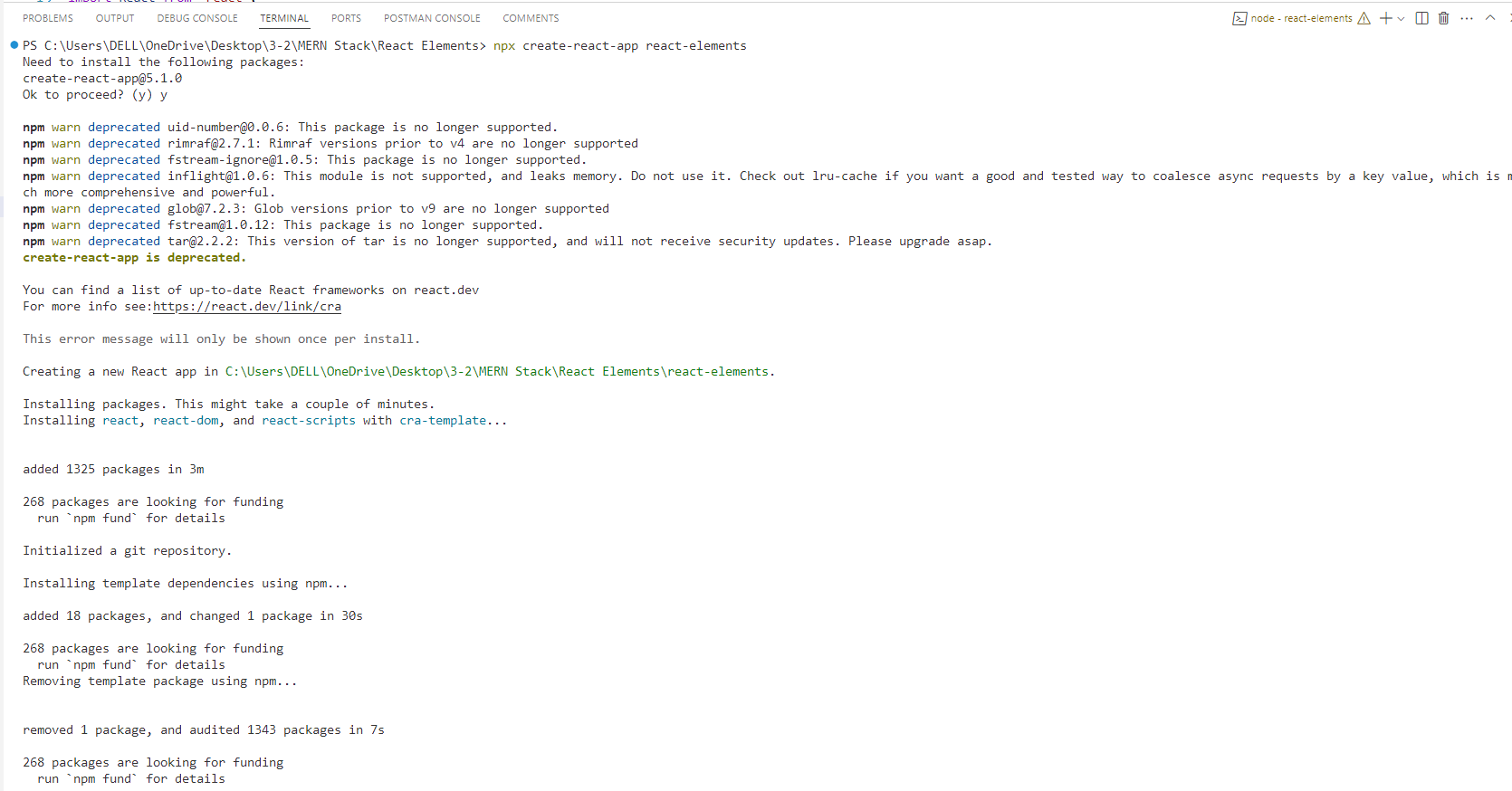
        </footer>

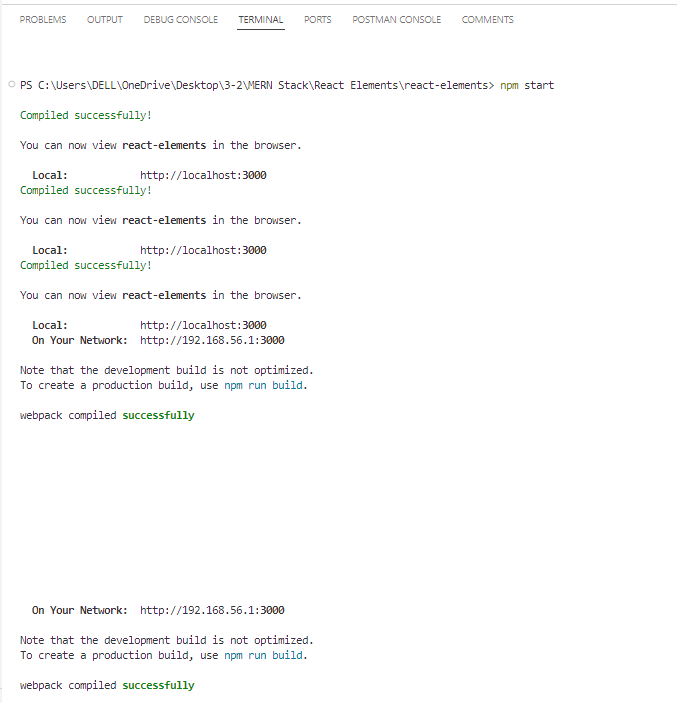
    );

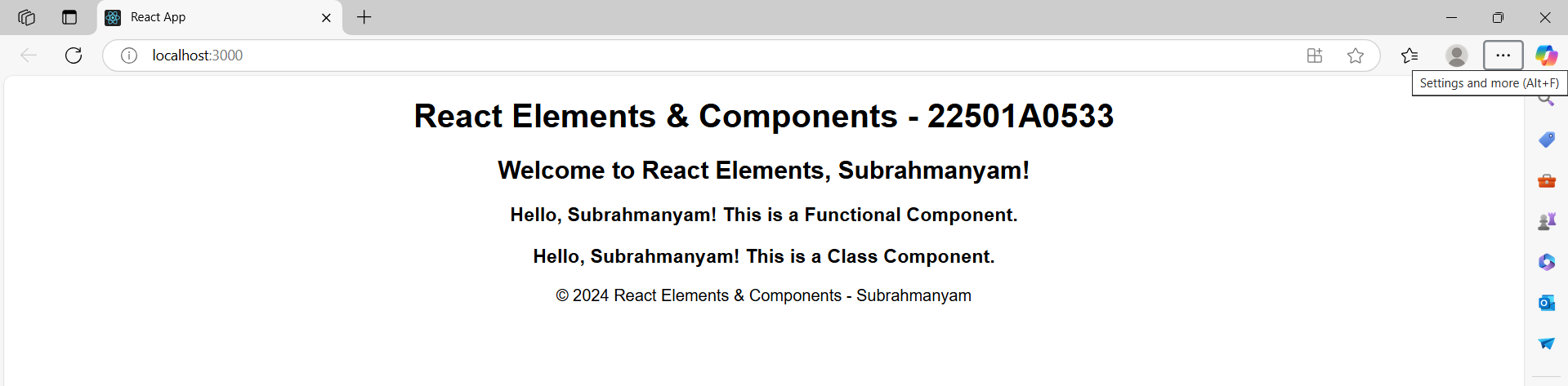
}

export default Footer;

**Output:**







**EXPERIMENT - 9**

**DEVELOP A SINGLE PAGE APPLICATION (SPA)**

**Aim:**

Develop a Single Page Application (SPA)

**Theory:**

**What is a Single Page Application (SPA)?**

* A web application that dynamically updates content without reloading the entire page.
* Uses JavaScript frameworks like React, Angular, or Vue.js.
* Improves performance and user experience by fetching only necessary data.

**Key Features of a SPA:**

* Uses client-side routing (e.g., React Router).
* Loads only required components when navigating between views.
* Reduces server requests by managing state on the client side.

**Steps to Develop an SPA with React:**

1. **Install Node.js and npm**
   * Ensure Node.js is installed (node -v and npm -v to verify).
2. **Create a React App**

npx create-react-app my-spa

cd my-spa

npm start

* + This sets up a new React project and runs the development server.

1. **Install React Router for Navigation**

npm install react-router-dom

* + Enables client-side routing without full-page reloads.

1. **Define Routes in the Application**
   * Use BrowserRouter, Routes, and Route components from react-router-dom.
   * Example routes: Home, Services, Projects, Contact.
2. **Implement Components for Different Views**
   * Create separate functional components for each page.
   * Example: Home.js, Services.js, Projects.js, Contact.js.
3. **Navigation with React Router**
   * Use Link components instead of <a> tags to prevent full-page reloads.
4. **Manage State Efficiently**
   * Use useState and useEffect for handling application state.
   * For complex state management, consider Redux or Context API.
5. **Deploy the Application**
   * Build the project using:

npm run build

* + Deploy to platforms like Vercel, Netlify, or Firebase Hosting.

**Program:  
  
App.js:**

import React, { useState } from "react";

import Header from "./Header";

import Footer from "./Footer";

import Home from "./Home";

import Services from "./Services";

import Projects from "./Projects";

import Contact from "./Contact";

function App() {

    const [page, setPage] = useState("home");

    const renderPage = () => {

        switch (page) {

            case "home":

                return <Home />;

            case "services":

                return <Services />;

            case "projects":

                return <Projects />;

            case "contact":

                return <Contact />;

            default:

                return <Home />;

        }

    };

    return (

        <div style={{ textAlign: "center", fontFamily: "Arial, sans-serif" }}>

            <Header setPage={setPage} />

            <main style={{ marginTop: "20px" }}>{renderPage()}</main>

            <Footer />

        </div>

    );

}

export default App;

**Home.js**

import React from "react";

function Home() {

    return <h2>Welcome to Subrahmanyam's Home Page!</h2>;

}

export default Home;

**Header.js**

import React from "react";

function Header({ setPage }) {

    return (

        <header>

            <h1>React SPA Example by Subrahmanyam</h1>

            <nav>

                <button onClick={() => setPage("home")}>Home</button>

                <button onClick={() => setPage("services")}>Services</button>

                <button onClick={() => setPage("projects")}>Projects</button>

                <button onClick={() => setPage("contact")}>Contact</button>

            </nav>

        </header>

    );

}

export default Header;

**Footer.js**

import React from "react";

function Footer() {

    return (

        <footer style={{ marginTop: "20px" }}>

            <p>© 2025 React SPA Example by Subrahmanyam | All Rights Reserved</p>

        </footer>

    );

}

export default Footer;

**Services.js**

import React from "react";

function Services() {

    return <h2>Our Services: Web Development, App Development</h2>;

}

export default Services;

**Projects.js**

import React from "react";

function Services() {

    return <h2>Our Services: Web Development, App Development</h2>;

}

export default Services;

**Contact.js**

import React from "react";

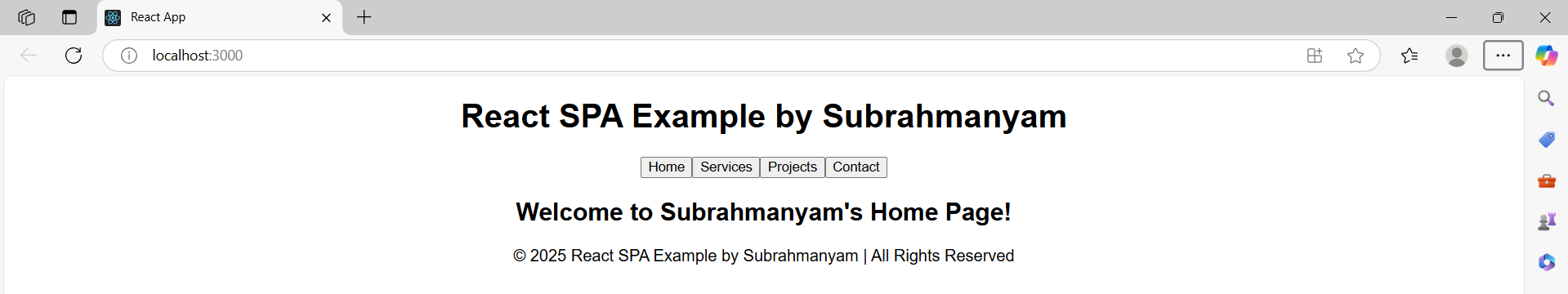
function Contact() {

    return <h2>Contact Subrahmanyam at: 22501a0533@pvpsit.ac.in</h2>;

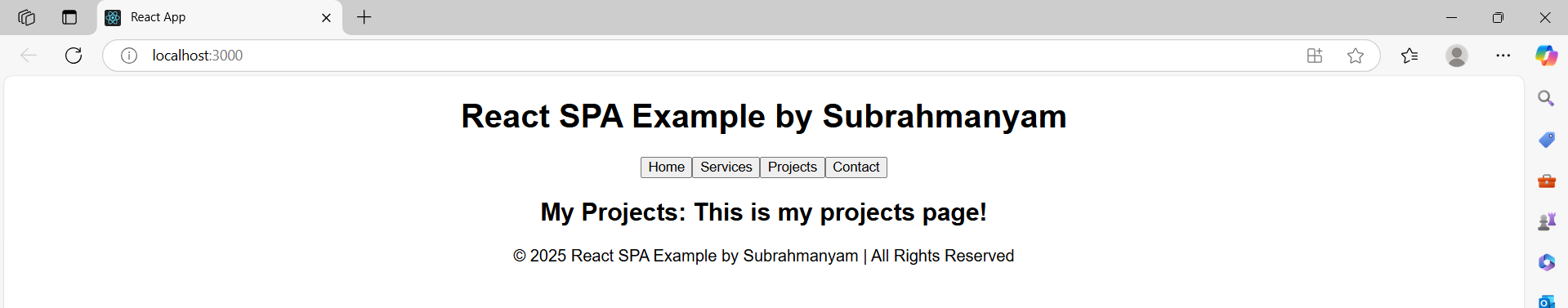
}

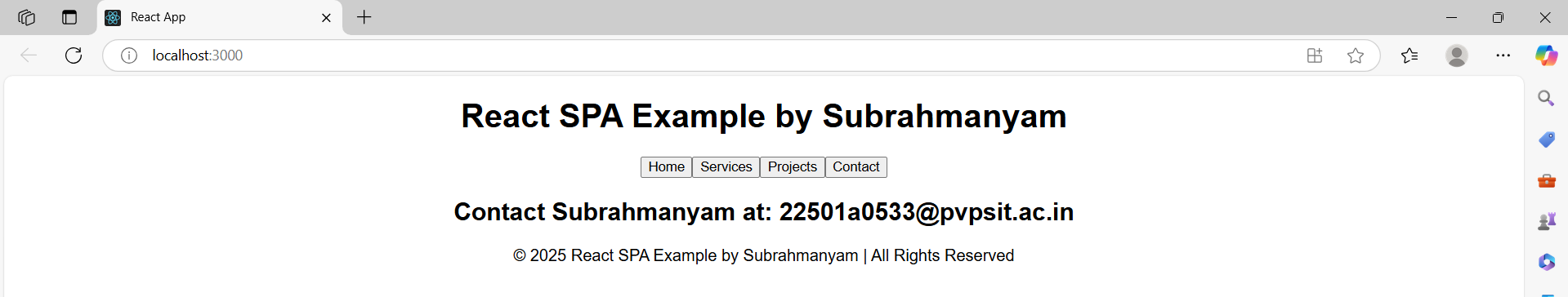
export default Contact;

**Output**

****

****

****

****