ವಿಶ್ವೇಶ್ವರಯ್ಯ ತಾಂತ್ರಿಕ ವಿಶ್ವವಿದ್ಯಾಲಯ, ಬೆಳಗಾವಿ

VISVESVARAYA TECHNOLOGICAL UNIVERSITY - BELAGAVI

BANGALORE INSTITUTE OF TECHNOLOGY
K.R.ROAD, V.V.PURA, BENGALURU -560 004

Department of Information Science and Engineering

CODE: BIS601

Full Stack Development

As per Choice Based Credit System Scheme (CBCS)

FOR VI SEMESTER ISE AS PRESCRIBED BY VTU

Prepared By:

Prof. Deeksha Chandra
Assistant Professor
Dept. of ISE, BIT

BANGALORE INSTITUTE OF TECHNOLOGY

VISION:

Establish and develop the Institute as the Centre of higher learning, ever abreast with expanding horizon of knowledge in the field of Engineering and Technology with entrepreneurial thinking, leadership excellence for life-long success and solve societal problems.

MISSION:

- Provide high quality education in the Engineering disciplines from the undergraduate through doctoral levels with creative academic and professional programs.
- Develop the Institute as a leader in Science, Engineering, Technology, Management and Research and apply knowledge for the benefit of society.
- Establish mutual beneficial partnerships with Industry, Alumni, Local, State and Central Governments by Public Service Assistance and Collaborative Research.
- Inculcate personality development through sports, cultural and extracurricular activities and engage in social, economic and professional challenges.

Bangalore Institute of Technology
K R Road, VV Puram, Bangalore- 560004
Department of Information Science and Engineering

VISION:

Empower every student to be innovative, creative and productive in the field of Information Technology by imparting quality technical education, **developing Professional Skills** and inculcating human values.

MISSION:

- To evolve continually as a centre of excellence in offering quality Information Technology **Education.**
- To nurture the students to meet the global competency in industry for **Employment.**
- To promote collaboration with industry and academia for constructive interaction to empower **Entrepreneurship.**
- To provide reliable, contemporary and integrated technology to support and facilitate **Life Long Learning.**

PROGRAM EDUCATIONAL OBJECTIVES

- Uplift the students through Information Technology **Education.**
- Provide exposure to emerging technologies and train them to **Employable** in Multi-disciplinary industries.
- Motivate them to become good professional Engineers and **Entrepreneur.**
- Inspire them to prepare for **Higher Learning and Research**.

PROGRAMME SPECIFIC OUTCOMES (PSOs)

- To provide our graduates **with Core Competence in Information Technology and Management.**
- To prepare our graduates with **relevant skills for Higher Education, Entrepreneurship, Professional career & Social values.**

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

	FULL STACK DEVELOPMENT		Semester	6
	Course Code	**BIS601**	CIE Marks	50
	Teaching Hours/Week (L: T:P: S)	3:0:2:0	SEE Marks	50
	Total Hours of Pedagogy	40 hours Theory + 8-10 Lab slots	Total Marks	100
	Credits	04	Exam Hours	03
	Examination type (SEE)	Theory		

Course objectives:

- To understand the essential javascript concepts for web development.
- To style Web applications using bootstrap.
- To utilize React JS to build front end User Interface.
- To understand the usage of API's to create web applications using Express JS.
- To store and model data in a no sql database.

Teaching-Learning Process (General Instructions)

These are sample Strategies, which teachers can use to accelerate the attainment of the various course outcomes.

1. Lecturer method (L) need not to be only a traditional lecture method, but alternative effective teaching methods could be adopted to attain the outcomes.
2. Use of Video/Animation to explain functioning of various concepts.
3. Encourage collaborative (Group Learning) Learning in the class.
4. Ask at least three HOT (Higher order Thinking) questions in the class, which promotes critical thinking.
5. Adopt Problem Based Learning (PBL), which fosters students' Analytical skills, develop design thinking skills such as the ability to design, evaluate, generalize, and analyze information rather than simply recall it.
6. Introduce Topics in manifold representations.
7. Show the different ways to solve the same problem with different circuits/logic and encourage the students to come up with their own creative ways to solve them.
8. Discuss how every concept can be applied to the real world - and when that's possible, it helps improve the students' understanding.

Module-1

Basic JavaScript Instructions, Statements, Comments, Variables, Data Types, Arrays, Strings, Functions, Methods & Objects, Decisions & Loops.

Text Book 1: Chapter 2, 3, 4

Module-2

Document Object Model: DOM Manipulation, Selecting Elements, Working with DOM Nodes, Updating Element Content & Attributes, Events, Different Types of Events, How to Bind an Event to an Element, Event Delegation, Event Listeners.

Text Book 1: Chapter: 5, 6, 13

Module-3

Form enhancement and validation. Introduction to MERN: MERN components, Server less Hello world.

React Components: Issue Tracker, React Classes, Composing Components, Passing Data Using Properties, Passing Data Using Children, Dynamic Composition.

Text Book 2: Chapter 1, 2, 3

Module-4

	React State: Initial State, Async State Initialization, Updating State, Lifting State Up, Event Handling, Stateless Components, Designing Components, State vs. Props, Component Hierarchy, Communication, Stateless Components. **Express**, REST API, GraphQL, Field Specification, Graph Based, Single Endpoint, Strongly Typed, Introspection, Libraries, The About API GraphQL Schema File, The List API, List API Integration, Custom Scalar types, The Create API, Create API Integration, Query Variables, Input Validations, Displaying Errors. **Text Book 2: Chapter 4, 5**
	Module-5
	MongoDB: Basics, Documents, Collections, Databases, Query Language, Installation, The Mongo Shell, MongoDB CRUD Operations, Create, Read, Projection, Update, Delete, Aggregate, MongoDB Node.js Driver, Schema Initialization, Reading from MongoDB, Writing to MongoDB. **Modularization and Webpack** ,Back-End Modules Front-End Modules and Webpack Transform and Bundle, Libraries Bundle ,Hot Module Replacement, Debugging DefinePlugin: Build Configuration, Production Optimization. **Text Book 2: Chapter 6, 7**
PRACTICAL COMPONENT OF IPCC	
Sl.NO	**Experiments**
1.	a. Write a script that Logs "Hello, World!" to the console. Create a script that calculates the sum of two numbers and displays the result in an alert box. b. Create an array of 5 cities and perform the following operations: Log the total number of cities. Add a new city at the end. Remove the first city. Find and log the index of a specific city.
2.	a. Read a string from the user, Find its length. Extract the word "JavaScript" using substring() or slice(). Replace one word with another word and log the new string. Write a function isPalindrome(str) that checks if a given string is a palindrome (reads the same backward).
3.	Create an object student with properties: name (string), grade (number), subjects (array), displayInfo() (method to log the student's details) Write a script to dynamically add a passed property to the student object, with a value of true or false based on their grade. Create a loop to log all keys and values of the student object.
4.	Create a button in your HTML with the text "Click Me". Add an event listener to log "Button clicked!" to the console when the button is clicked. Select an image and add a mouseover event listener to change its border color. Add an event listener to the document that logs the key pressed by the user.
5.	Build a React application to track issues. Display a list of issues (use static data). Each issue should have a title, description, and status (e.g., Open/Closed). Render the list using a functional component.
6.	Create a component Counter with A state variable count initialized to 0. Create Buttons to increment and decrement the count. Simulate fetching initial data for the Counter component using useEffect (functional component) or componentDidMount (class component). Extend the Counter component to Double the count value when a button is clicked. Reset the count to 0 using another button.
7.	Install Express (npm install express). Set up a basic server that responds with "Hello, Express!" at the root endpoint (GET /). Create a REST API. Implement endpoints for a Product resource: GET : Returns a list of products. POST : Adds a new product. GET /:id: Returns details of a specific product. PUT /:id: Updates an existing product. DELETE /:id: Deletes a product. Add middleware to log requests to the console. Use express.json() to parse incoming JSON payloads.

8.	Install the MongoDB driver for Node.js. Create a Node.js script to connect to the shop database. Implement insert, find, update, and delete operations using the Node.js MongoDB driver. Define a product schema using Mongoose. Insert data into the products collection using Mongoose. Create an Express API with a /products endpoint to fetch all products. Use fetch in React to call the /products endpoint and display the list of products. Add a POST /products endpoint in Express to insert a new product. Update the Product List, After adding a product, update the list of products displayed in React.

Course outcome (Course Skill Set)

At the end of the course, the student will be able to :

1. Apply Javascript to build dynamic and interactive Web projects .
2. Implement user interface components for JavaScript-based Web using React.JS
3. Apply Express/Node to build web applications on the server side.
4. Develop data model in an open source nosql database.
5. Demonstrate modularization and packing of the front-end modules .

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

Continuous Internal Evaluation:

- For the Assignment component of the CIE, there are 25 marks and for the Internal Assessment Test component, there are 25 marks.
- The first test will be administered after 40-50% of the syllabus has been covered, and the second test will be administered after 85-90% of the syllabus has been covered
- Any two assignment methods mentioned in the 22OB2.4, if an assignment is project-based then only one assignment for the course shall be planned. The teacher should not conduct two assignments at the end of the semester if two assignments are planned.
- For the course, CIE marks will be based on a scaled-down sum of two tests and other methods of assessment.

Internal Assessment Test question paper is designed to attain the different levels of Bloom's taxonomy as per the outcome defined for the course.

Semester-End Examination:

Theory SEE will be conducted by University as per the scheduled timetable, with common question papers for the course (**duration 03 hours).**

1. The question paper will have ten questions. Each question is set for 20 marks.
2. There will be 2 questions from each module. Each of the two questions under a module (with a maximum of 3 sub-questions), **should have a mix of topics** under that module.
3. The students have to answer 5 full questions, selecting one full question from each module.
4. Marks scored shall be proportionally reduced to 50 marks

Suggested Learning Resources:

Books

1. Jon Duckett, "JavaScript & jQuery: Interactive Front-End Web Development", Wiley, 2014.
2. Vasan Subramanian, Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node. Apress, 2019.

Web links and Video Lectures (e-Resources):

- https://github.com/vasansr/pro-mern-stack
- https://nptel.ac.in/courses/106106156
- https://archive.nptel.ac.in/courses/106/105/106105084/

Activity Based Learning (Suggested Activities in Class)/ Practical Based learning

- Course Project: Build Web applications using MERN stack. Students (group of 2) can choose any real-world problem from domains such as finance, marketing, medical, or enterprise projects **(25 marks)**.

Bangalore Institute of Technology

K. R. Road, V.V. Pura, Bengaluru 560004

Department of Information Science and Engineering

Full stack Development Laboratory (BIS601)

PRE-REQUISITES:

- Basic knowledge about computer science.
- Fundamental knowledge on front end and back end tools.
- Requires the information about web development.
- Knowledge of scripting languages.
- Basic knowledge about the database and its connectivity.

COURSE LEARNING OBJECTIVES (CLO)

The main objectives of this course are to,

CLO 1: To understand the essential JavaScript concepts for web development.

CLO 2: To style Web applications using bootstrap.

CLO 3: To utilize React JS to build front end User Interface.

CLO 4: To understand the usage of APIs to create web applications using Express JS.

CLO 5: To store and model data in a no SQL database.

CLO 6: To become knowledgeable about the most recent web development technologies.

COURSE OUTCOMES (CO)

On the completion of this laboratory course, the students will be able to:

CO 1: Understand the JavaScript and MERN components to build dynamic and interactive web projects.

CO 2: Implement user interface components for JavaScript-based web using React.JS. Apply Express/Node to build web applications on the server side. Demonstrate modularization and packing of the front-end modules.

CO 3: Differentiate between state and props in component communication. Analyze the role of various components in the MERN stack.

CO 4: Develop real-world web applications using various technologies learned in the course. (MERN stack)

Mapping of COs-POs and COs-PSOs
Full stack Development Laboratory (BIS601)

		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
BIS601	CO1												
	CO2	2											2
	CO3	2	2										2
	CO4	3	2	2		2	1			3	3		2
	AVG	2.33	2	2		2	1			3	3		2

		PSO1	PSO2
Full stack Development Laboratory BIS601	CO1		
	CO2	2	3
	CO3	2	3
	CO4	2	3
	AVG	2	3

FULL STACK DEVELOPMENT LABORATORY

Subject Code: BIS601 **Hours/Week: 0:0:2:0**

List of Programs

Sl. No.	Name of Experiment
1.	a. Write a script that Logs "Hello, World!" to the console. Create a script that calculates the sum of two numbers and displays the result in an alert box. b. Create an array of 5 cities and perform the following operations: Log the total number of cities. Add a new city at the end. Remove the first city. Find and log the index of a specific city.
2.	Read a string from the user, Find its length. Extract the word "JavaScript" using substring() or slice(). Replace one word with another word and log the new string. Write a function isPalindrome(str) that checks if a given string is a palindrome (reads the same backward).
3.	Create an object student with properties: name (string), grade (number), subjects (array), displayInfo() (method to log the student's details) Write a script to dynamically add a passed property to the student object, with a value of true or false based on their grade. Create a loop to log all keys and values of the student object.
4.	Create a button in your HTML with the text "Click Me". Add an event listener to log "Button clicked!" to the console when the button is clicked. Select an image and add a mouseover event listener to change its border color. Add an event listener to the document that logs the key pressed by the user.
5.	Build a React application to track issues. Display a list of issues (use static data). Each issue should have a title, description, and status (e.g., Open/Closed). Render the list using a functional component.
6.	Create a component Counter with A state variable count initialized to 0. Create Buttons to increment and decrement the count. Simulate fetching initial data for the Counter component using useEffect (functional component) or componentDidMount (class component). Extend the Counter component to Double the count value when a button is clicked. Reset the count to 0 using another button.
7.	Install Express (npm install express). Set up a basic server that responds with "Hello, Express!" at the root endpoint (GET /). Create a REST API. Implement endpoints for a Product resource: GET : Returns a list of products. POST : Adds a new product. GET /:id: Returns details of a specific product. PUT /:id: Updates an existing product. DELETE /:id: Deletes a product. Add middleware to log requests to the console. Use express.json() to parse incoming JSON payloads.
8.	Install the MongoDB driver for Node.js. Create a Node.js script to connect to the shop database. Implement insert, find, update, and delete operations using the Node.js MongoDB driver. Define a product schema using Mongoose. Insert data into the products collection using Mongoose. Create an Express API with a /products endpoint to fetch all products. Use fetch in React to call the /products endpoint and display the list of products. Add a POST /products endpoint in Express to insert a new product. Update the Product List, After adding a product, update the list of products displayed in React.

FULL STACK DEVELOPMENT LABORATORY

Subject Code: BIS601 **CIE Marks: 25**
Hours/Week: 0:0:2:0 **Total Hours: 24**

Evaluation Criteria

Lab Write-up and Execution rubrics for Daily Conduction (Max: 15 marks)

	Criteria	Good (Average Performance)	Excellent (Outstanding Performance)
a.	**Understandin g ofproblem and approach to solve. (4 Marks) CO1**	Demonstrates a strong understanding of JavaScript, MERN components, and their integration to build dynamic and interactive web projects. (4)	Shows a moderate understanding of JavaScript and MERN stack concepts. (2)
b.	**Execution (5 Marks) CO2**	Implements user interface components using React.js efficiently, applies Express/Node to develop server-side logic, and demonstrates effective modularization and packaging of front-end modules. (5)	Demonstrates basic implementation of UI components, Express/Node, and modularization but lacks optimization. (3)
c.	**Viva Formulation of program / Program Explanation. (3 Marks) CO3**	Demonstrates different strategies and concepts to derive solutions, effectively explains state and props usage in React component communication. (3)	Provides intuitive but limited explanation of strategies and concepts for problem-solving, with some gaps in state and props understanding. (2)
d.	**Documentation (3 Marks) CO1**	Clearly interprets various technologies used in the MERN stack with appropriate examples, showcasing real-world application development. (3)	Documents only a few concepts with minimal real-world application examples. (2)

Lab Write-up and Execution rubrics for Internals (Max: 10 marks)

	Criteria	Good (Average Performance)	Excellent (Outstanding Performance)
a.	**Understandin g ofproblem and approach to solve. (5 Marks) CO1**	Demonstrates strong knowledge of JavaScript, MERN stack components, and their integration to build dynamic and interactive web projects. (5)	Shows moderate understanding of JavaScript, MERN stack concepts, and their role in web development. (3)
b.	**Execution (10 Marks) CO2**	Implements UI components effectively using React.js, applies Express/Node.js to develop server-side logic, and demonstrates modularization and packaging of front-end modules. (10)	Basic implementation of UI components, Express/Node.js, and modularization with some gaps in efficiency. (6)
c.	**Viva Formulation of program / Program Explanation. (5 Marks) CO2**	Demonstrates different strategies and concepts to derive solutions, effectively explains state vs. props in React component communication, and traces program logic well. (5)	Provides an intuitive but limited explanation of strategies and concepts with some misunderstanding of state and props. (3)

FULL STACK DEVELOPMENT LABORATORY

Subject Code: BIS601 **CIE Marks: 25**
Hours/Week: 0:0:2:0 **Total Hours: 24**

Lesson Planning / Schedule of Experiments

Sl. No	Name of Experiment	WEEK
1	a. Write a script that Logs "Hello, World!" to the console. Create a script that calculates the sum of two numbers and displays the result in an alert box. b. Create an array of 5 cities and perform the following operations: Log the total number of cities. Add a new city at the end. Remove the first city. Find and log the index of a specific city	Week1
2	Read a string from the user, Find its length. Extract the word "JavaScript" using substring() or slice(). Replace one word with another word and log the new string. Write a function isPalindrome(str) that checks if a given string is a palindrome (reads the same backward).	Week2
3	Create an object student with properties: name (string), grade (number), subjects (array), displayInfo() (method to log the student's details) Write a script to dynamically add a passed property to the student object, with a value of true or false based on their grade. Create a loop to log all keys and values of the student object	Week3
4	Create a button in your HTML with the text "Click Me". Add an event listener to log "Button clicked!" to the console when the button is clicked. Select an image and add a mouseover event listener to change its border color. Add an event listener to the document that logs the key pressed by the user.	Week4
5	**TEST- 1**	Week5
6	Build a React application to track issues. Display a list of issues (use static data). Each issue should have a title, description, and status (e.g., Open/Closed). Render the list using a functional component.	Week6
7	Create a component Counter with A state variable count initialized to 0. Create Buttons to increment and decrement the count. Simulate fetching initial data for the Counter component using useEffect (functional component) or componentDidMount (class component). Extend the Counter component to Double the count value when a button is clicked. Reset the count to 0 using another button	Week7
8	Install Express (npm install express). Set up a basic server that responds with "Hello, Express!" at the root endpoint (GET /). Create a REST API. Implement endpoints for a Product resource: GET : Returns a list of products. POST : Adds a new product. GET /:id: Returns details of a specific product. PUT /:id: Updates an existing product. DELETE /:id: Deletes a product. Add middleware to log requests to the console. Use express.json() to parse incoming JSON payloads.	Week8
9	Install the MongoDB driver for Node.js. Create a Node.js script to connect to the shop database. Implement insert, find, update, and delete operations using the Node.js MongoDB driver. Define a product schema using Mongoose. Insert data into the products collection using Mongoose. Create an Express API with a /products endpoint to fetch all products. Use fetch in React to call the /products endpoint and display the list of products. Add a POST /products endpoint in Express to insert a new product. Update the Product List, After adding a product, update the list of products displayed in React.	Week9
10	**TEST-2**	Week10

FULL STACK DEVELOPMENT LABORATORY

Subject Code: BIS601 **CIE Marks: 25**
Hours/Week: 0:0:2 **Total Hours: 24**

INDEX
DAILY CONDUCTION

SI. No.	Contents	Page No.	Date of Execution	Date of Submission	Daily Conduction				Total Marks 15M	Staff Signature
					Understanding CO1 4M	Execution CO2 5M	Analysis CO3 3M	Record CO1 3M		
1	Log "Hello, World!" and sum.									
2	Manipulate string, extract, replace, check.									
3	Create and modify student object.									
4	Add button, image, key events.									
5	Build React issue tracker component.									
6	Create Counter with state management.									
7	Setup Express server with CRUD.									
8	Install MongoDB, connect, CRUD, fetch, display.									
Total										
Average										
Average of Daily Conduction										

LAB INTERNAL MARKS				
TEST-1 (20 Marks)	Write- up CO1 5M	Execution CO2 10M	Viva CO2 5M	Total Marks
TEST-2 (20 Marks)	Write- up CO1 5M	Execution CO2 10M	Viva CO2 5M	Total Marks
			Total Marks=20	
			Reduced to 10	

FINAL L A B MARKS		
	Max. Marks	Marks Scored
Daily Conduction	15	
Lab Internals	10	
TOTAL	25	
Signature of the faculty		

Full-Stack Development Using Node.js and MongoDB

Full-stack development with Node.js (backend) and MongoDB (database) involves building web applications that cover both frontend (client-side) and backend (server-side) development. Below is a point-by-point explanation:

Experiments

Experiment 1:

1. a. Write a script that Logs "Hello, World!" to the console. Create a script that calculates the sum of two numbers and displays the result in an alert box.

   b. Create an array of 5 cities and perform the following operations:

Log the total number of cities. Add a new city at the end. Remove the first city. Find and log the index of a specific city.

1. Install Node.js:

- **Download:** Go to the Node.js official website and download the latest version suitable for Windows.
- **Install:** Run the downloaded installer and follow the on-screen instructions to install Node.js. This will also install npm (Node Package Manager), which is useful for managing JavaScript packages.

2. Set Up Your Environment

- **Text Editor:** Use a text editor like Visual Studio Code or any other text editor of your choice.
- **Command Line:** Use the Command Prompt or PowerShell to run your JavaScript files.

3. Running the Script:

- **Save Your Script:** Save the above JavaScript code in a file with a .js extension, e.g., script.js.
- Execute the Script:
  - ♦ Open the Command Prompt or PowerShell.
  - ♦ Navigate to the directory where your script.js file is located.
  - ♦ Run the script using Node.js by typing:

Windows Power Shell

```
Node script.js
```

- This will execute the JavaScript code and you will see the output in the console.

- By following these steps, you can run and test the JavaScript programs on your Windows desktop
- **<u>To know the node.js version:</u>**

```
node –v
```

- **<u>Experiment 1a.js</u>**
- **Script to log "Hello, World!" to the console**

```
console.log("Hello,World!");
```

- Script to calculate the sum of two numbers and display the result in an alert box

```
Function calculateSum(a,b)
{
return sum = a + b;
}
```

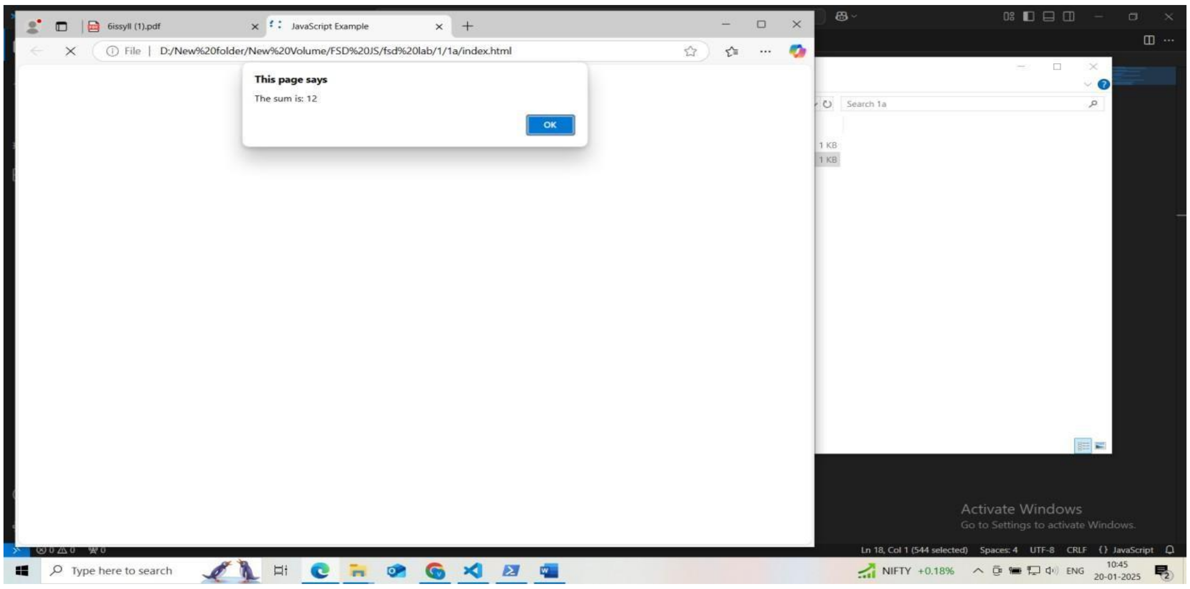
```
Experiment usage
```

```
if (typeof alert === "function")
{
 alert("The sum of 5 and 7 is: " + calculateSum(5,7));
}
else
{
console.log("The sum of 5 and 7 is: " + calculateSum(5, 7));
}
```

<u>index.html to make working of alert function:</u>

OUTPUT:

Just click on index.html icon to see the result.



```
D:/New%20folder/New%20Volume/FSD%20JS/fsd%20lab/1/1a/index.html
```

```
This page says
The sum is: 12
```

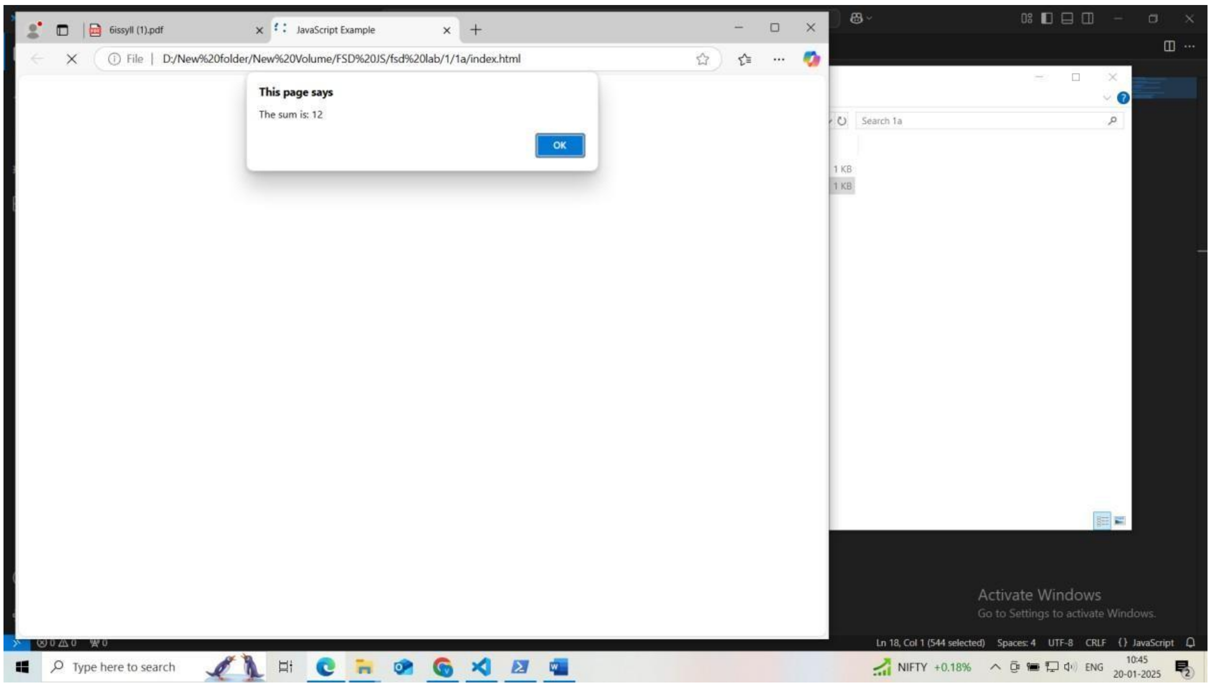
Some other way like whole thing implemented in single html file

1a.html

Program:

```
 <!DOCTYPE html>
<head>
    <title>Dynamic Input Experiment</title>
    <script>
        // Script to log "Hello, World!" to the console
        console.log("Hello, World!");

        // Script to calculate the sum of two numbers and display the
result in an alert box

        // Read numbers from user input
        let num1 = parseFloat(prompt("Enter the first number:"));
        let num2 = parseFloat(prompt("Enter the second number:"));


        // Validate input
        let sum = num1 + num2; alert("The sum is: " + sum);
        // Experiment usage

    </script>
</head>
</html>
```

OUTPUT:


This page says

The sum is: 12

OK
JavaScript Example
D:/New%20folder/New%20Volume/FSD%20JS/fsd%20lab/1/1a/index.html

1b.js

Program:

```
const prompt = require('prompt-sync')();

// Prompt the user to enter 5 city names, separated by commas
let input = prompt("Enter 5 cities separated by commas: ");
let cities = input.split(',').map(city => city.trim());
console.log("Initial cities:", cities);

// Log the total number of cities
console.log("Total number of cities:", cities.length);

// Add a new city at the end
let newCity = prompt("Enter a city to add to the end: ");
cities.push(newCity);
console.log("Cities after adding a new one:", cities);

// Remove the first city
console.log("Removing the first city:", cities[0]); // Log the first city
before removal
cities.shift();
console.log("Cities after removing the first one:", cities);

// Find and log the index of a specific city
let searchCity = prompt("Enter a city to find its index: ");
let cityIndex = cities.indexOf(searchCity);

console.log("Index of", searchCity + ":", cityIndex !== -1 ? cityIndex :
"City notfound");
```

1bb.html

Program:

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <title>Dynamic Cities Array</title>
</head>
<body>
        <h1>Dynamic Cities Array</h1>
        <script src="1b.js"></script>
</body>
</html>
```

- Execution

First install node js

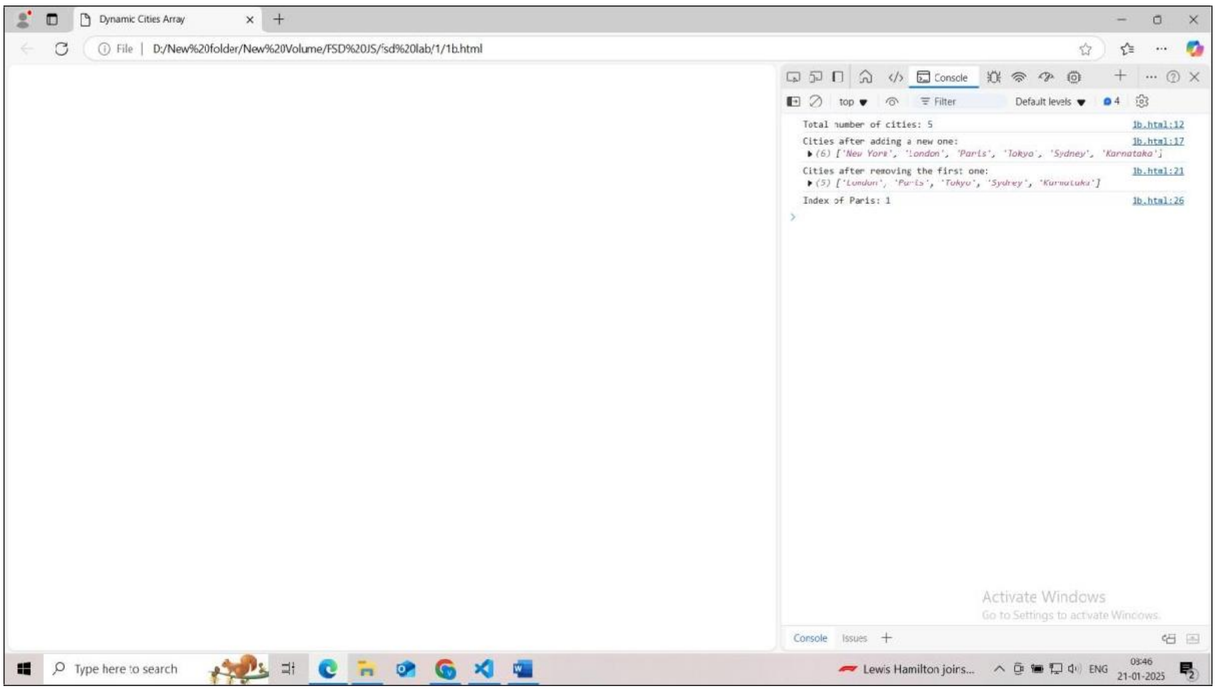
- Next run

```
npm install prompt-sync
```

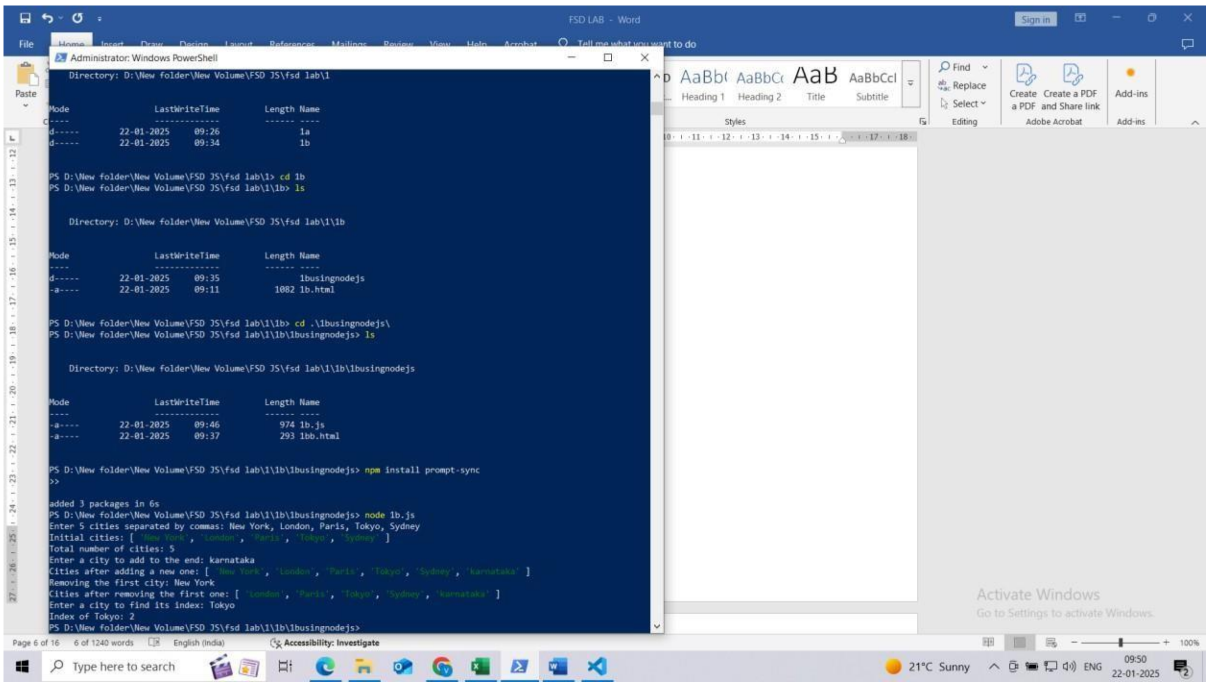
node 1b.js

<u>OUTPUT</u>: node 1b.js

Input: New York, London, Paris, Tokyo, Sydney



```
Total number of cities: 5                                                  1b.html:12
Cities after adding a new one:                                             1b.html:17
 ▸(6) ['New York', 'London', 'Paris', 'Tokyo', 'Sydney', 'Karnataka']
Cities after removing the first one:                                       1b.html:21
 ▸(5) ['London', 'Paris', 'Tokyo', 'Sydney', 'Karnataka']
Index of Paris: 1                                                          1b.html:26
```



```
    Directory: D:\New folder\New Volume\FSD JS\fsd lab\1


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        22-01-2025     09:26                1a
d-----        22-01-2025     09:34                1b


PS D:\New folder\New Volume\FSD JS\fsd lab\1> cd 1b
PS D:\New folder\New Volume\FSD JS\fsd lab\1\1b> ls


    Directory: D:\New folder\New Volume\FSD JS\fsd lab\1\1b


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        22-01-2025     09:35                1busingnodejs
-a----        22-01-2025     09:11           1082 1b.html


PS D:\New folder\New Volume\FSD JS\fsd lab\1\1b> cd .\1busingnodejs\
PS D:\New folder\New Volume\FSD JS\fsd lab\1\1b\1busingnodejs> ls


    Directory: D:\New folder\New Volume\FSD JS\fsd lab\1\1b\1busingnodejs


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        22-01-2025     09:46            974 1b.js
-a----        22-01-2025     09:37            293 1bb.html


PS D:\New folder\New Volume\FSD JS\fsd lab\1\1b\1busingnodejs> npm install prompt-sync
>>

added 3 packages in 6s
PS D:\New folder\New Volume\FSD JS\fsd lab\1\1b\1busingnodejs> node 1b.js
Enter 5 cities separated by commas: New York, London, Paris, Tokyo, Sydney
Initial cities: [ 'New York', 'London', 'Paris', 'Tokyo', 'Sydney' ]
Total number of cities: 5
Enter a city to add to the end: karnataka
Cities after adding a new one: [ 'New York', 'London', 'Paris', 'Tokyo', 'Sydney', 'karnataka' ]
Removing the first city: New York
Cities after removing the first one: [ 'London', 'Paris', 'Tokyo', 'Sydney', 'karnataka' ]
Enter a city to find its index: Tokyo
Index of Tokyo: 2
PS D:\New folder\New Volume\FSD JS\fsd lab\1\1b\1busingnodejs>
```

Experiment 2:

Read a string from the user, Find its length. Extract the word "JavaScript" using substring() or slice(). Replace one word with another word and log the new string. Write a function isPalindrome(str) that checks if a given string is a palindrome (reads the same backward).

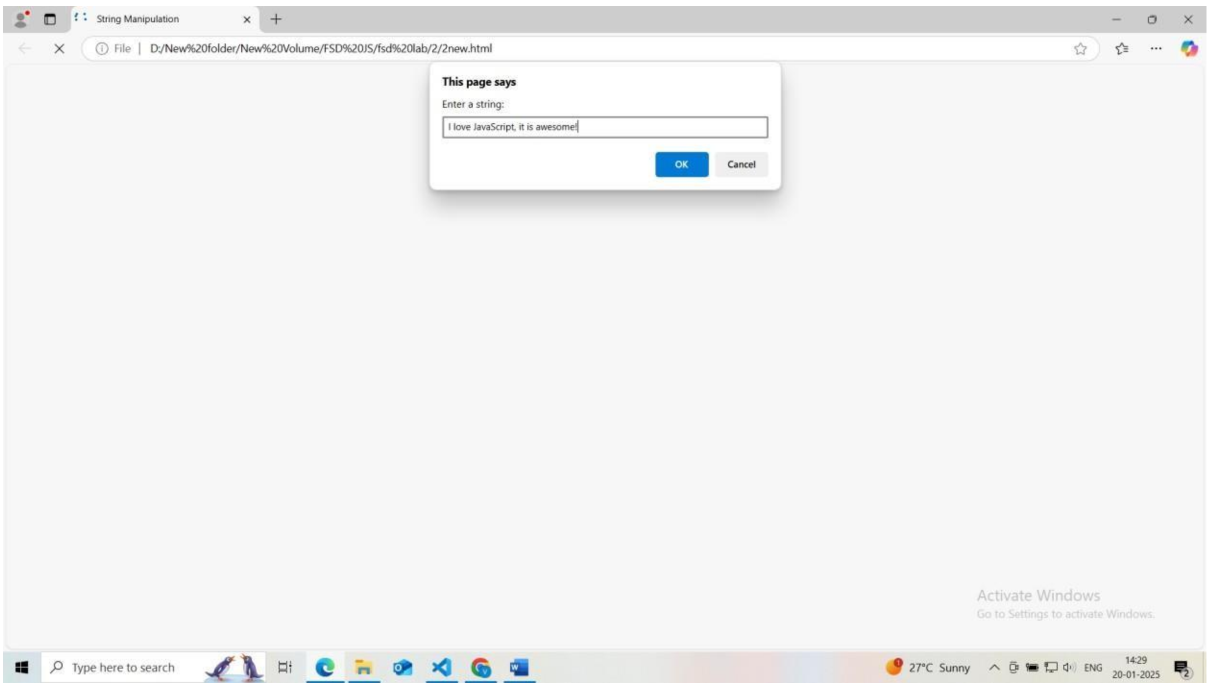
Program:

```
<!DOCTYPE html>
<head>

<title>String Manipulation</title>
<script>
// Function to check if a string is a palindrome function
isPalindrome1(str) {
for (let i = 0; i < str.length / 2; i++)
{
          if (str.charAt(i) != str.charAt(str.length - 1 - i))
          {
               return false;
          }

}
return true;
}
function isPalindrome(str)
{
let cleanedStr = str.replace(/[^A-Za-z0-9]/g,
'').toLowerCase();
return cleanedStr ===
cleanedStr.split('').reverse().join('');
}

// Read string from user and perform operations
let userInput = prompt("Enter a string:");
let lengthOfString = userInput.length;

// Extract 'JavaScript' and replace if present Let
extractedWord = userInput.includes("JavaScript") ?
userInput.substring(userInput.indexOf("JavaScript"),
userInput.indexOf("JavaScript") + "JavaScript".length) :
"JavaScript not found";
let newString = userInput.replace("JavaScript",
"TypeScript");
let palindromeCheck = isPalindrome(userInput);
```

```
// Log results
            console.log(`New String after Replacement:
${newString}`);

// Display alerts
alert(`Length: ${lengthOfString}\nExtracted Word:
${extractedWord}\nNew String: ${newString}\nPalindrome:
${palindromeCheck}`);
</script>

</head>
</html>
```

OUTPUT:

Give the input string as ------------- I love JavaScript, it is awesome!

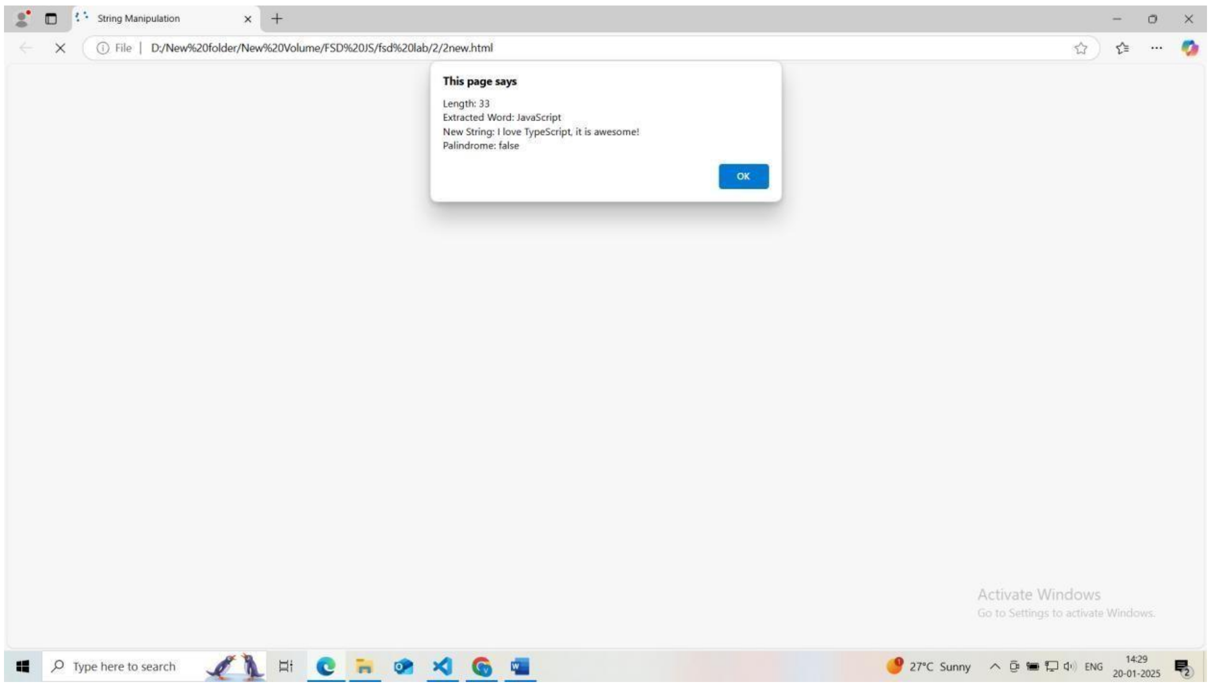


This page says

Enter a string:

```
I love JavaScript, it is awesome!
```

String Manipulation
File | D:/New%20folder/New%20Volume/FSD%20JS/fsd%20lab/2/2new.html
OK
Cancel
Activate Windows
Go to Settings to activate Windows.
Type here to search
27°C Sunny
ENG
14:29
20-01-2025



String Manipulation
File | D:/New%20folder/New%20Volume/FSD%20JS/fsd%20lab/2/2new.html
This page says
OK
Activate Windows
Go to Settings to activate Windows.
Type here to search
27°C Sunny
ENG
14:29
20-01-2025

```
Length: 33
Extracted Word: JavaScript
New String: I love TypeScript, it is awesome!
Palindrome: false
```

Experiment 3:

Create an object student with properties: name (string), grade (number), subjects (array), displayInfo() (method to log the student's details) Write a script to dynamically add a passed property to the student object, with a value of true or false based on their grade. Create a loop to log all keys and values of the student object.

One student

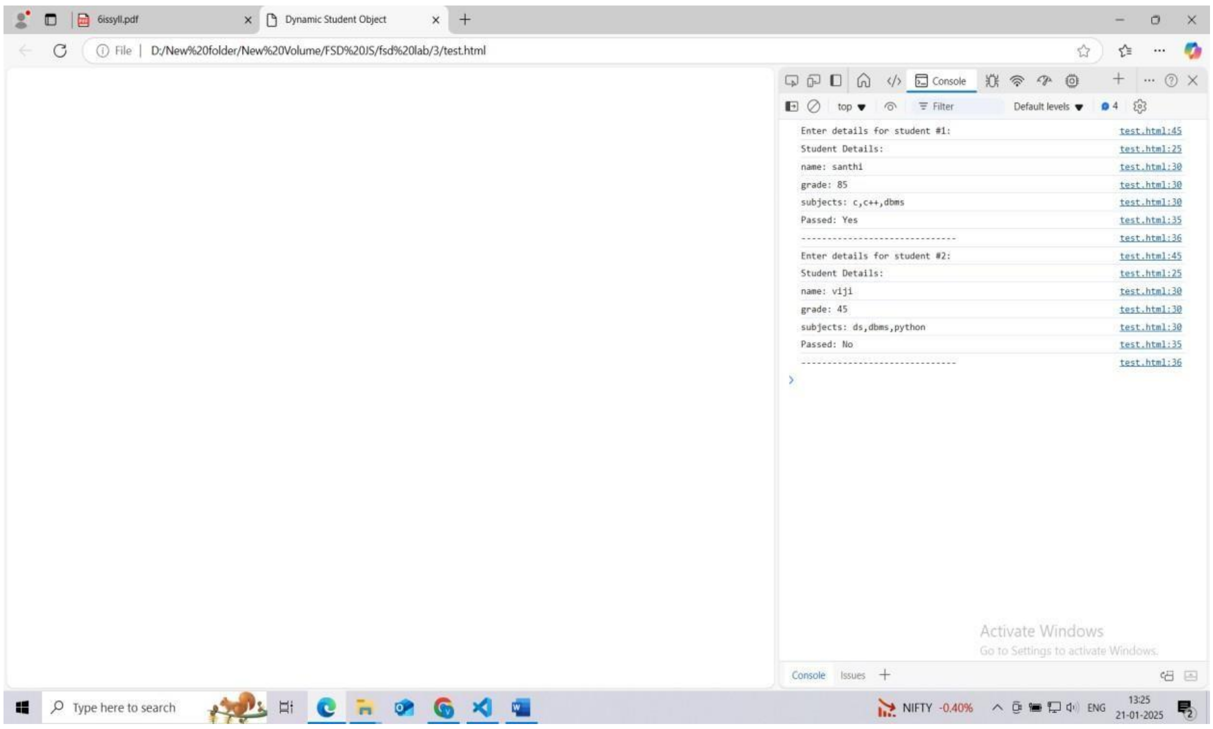
```
<!DOCTYPE html>
<head>
        <title>Dynamic Student Object</title>
           <script>
               // Function to create a student
               object function createStudent()
               {
                   // Collect inputs for the
                   student object let student =
                {
                name: prompt("Enter the student's name:"),
                grade: parseInt(prompt("Enter the student's
                grade:")), subjects: prompt("Enter the
                student's subjects separated by
                commas:").split(',').map(subject =>
                subject.trim()),
                };
           return student;
               }

       // Function to display student
          details function
          displayInfo(student)
          {
           console.log("Student Details:");

           // Loop through the properties of the student
object and log the keys and values
           for (let key in student)
          {
               if (typeof student[key] !== 'function')
               {
                   console.log(`${key}: ${student[key]}`);
               }
           }
```

```
  // Log whether the student passed based on the 'passed'
  function

  console.log("Passed:", student.passed() ? "Yes" :
  "No"); console.log("-------------------------------------------------------------------------");
 }
        let student = createStudent();  // Collect details and
return student object
        // Dynamically add 'passed' property based on
        grade student.passed = student.grade >= 40;
        displayInfo(student);
        </script>
</head>
</html>
```

Many students:

```
<!DOCTYPE html>
<head>
    <title>Dynamic Student Object</title>
    <script>
        // Function to create a student
        object function createStudent()
            {
            // Collect dynamic inputs for the
            student object let student =
                {
                name: prompt("Enter the student's name:"),
                grade: parseFloat(prompt("Enter the student's
                grade:")), subjects: prompt("Enter the student's
                subjects separated by
commas:").split(',').map(subject => subject.trim()),

                // Dynamically add 'passed' property
                based on grade passed: function()
                  {
                    return this.grade >= 50;
                }
            };
            return student;
        }
        // Function to display student
        details function
        displayInfo(student)
            {
            console.log("Student Details:");

            // Loop through the properties of the student
object and log the keys and values
            for (let key in student)
                 {
                if (typeof student[key] !== 'function')
                 {

                  console.log(`${key}: ${student[key]}`);
                  }

 // Log whether the student passed based on the 'passed'
 function
            console.log("Passed:", student.passed() ? "Yes" :
            "No"); console.log("------------------------------------------------------------------
            ------");
        }

        // Collect details for multiple students
        let numStudents = parseInt(prompt("How many students'
details would you like to enter?"));

        // Loop to collect details for
        each student for (let i = 0; i <
        numStudents; i++)
                 {
            console.log(`Enter details for student #${i +
            1}:`);
            let student = createStudent(); // Collect details
and return student object
            displayInfo(student);
                 }


    </script>
</head>
</html>
```

OUTPUT:



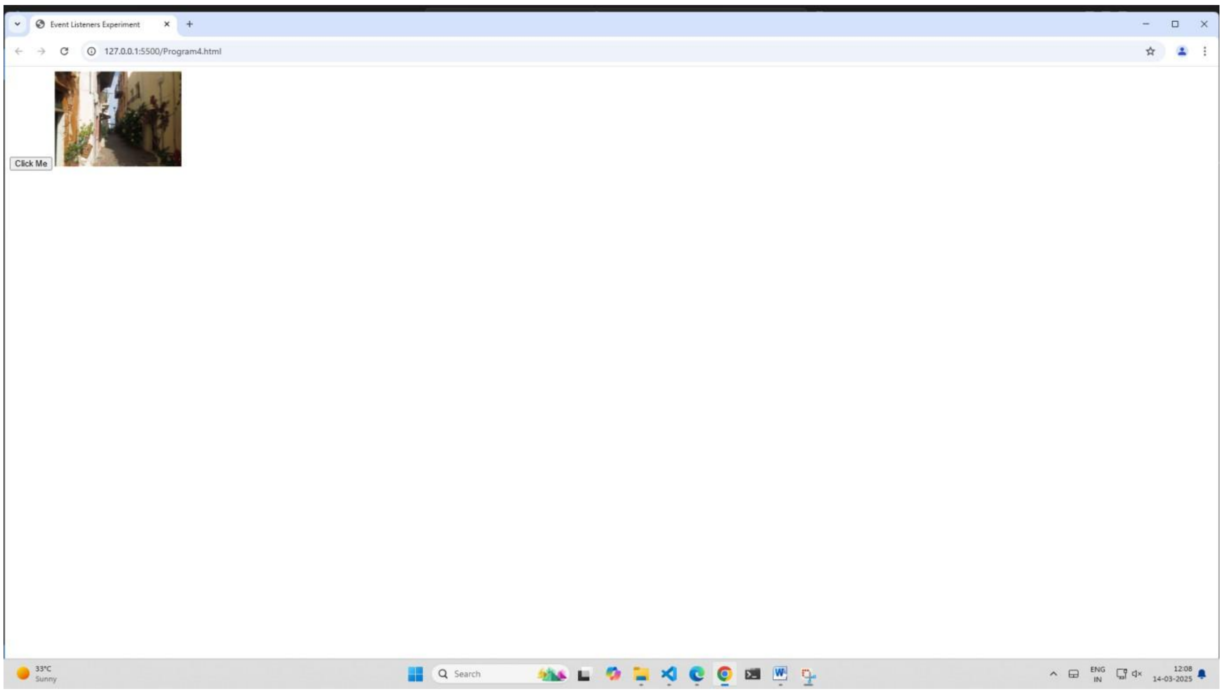
```
Enter details for student #1:          test.html:45
Student Details:                       test.html:25
name: santhi                           test.html:30
grade: 85                              test.html:30
subjects: c,c++,dbms                   test.html:30
Passed: Yes                            test.html:35
------------------------------         test.html:36
Enter details for student #2:          test.html:45
Student Details:                       test.html:25
name: viji                             test.html:30
grade: 45                              test.html:30
subjects: ds,dbms,python               test.html:30
Passed: No                             test.html:35
------------------------------         test.html:36
```

Experiment 4:

Create a button in your HTML with the text "Click Me". Add an event listener to log "Button clicked!" to the console when the button is clicked. Select an image and add a mouseover event listener to change its border color. Add an event listener to the document that logs the key pressed by the user.

```
<!DOCTYPE html>
<head>
<title>Event Listeners Experiment</title>
</head>
<body>
<!-- Button to click -->
<button id="clickButton">Click Me</button>
<!-- Experiment image with a reliable URL -->
<img id="ExperimentImage"
src="https://www.w3schools.com/html/img_chania.jpg"
alt="Experiment Image" width="200">
<script>
        // Event listener for the button click
document.getElementById('clickButton').addEventListener('click
', function()
{
console.log("Button clicked!");
});
        // Event listener for mouseover on the image to change
        its border color
document.getElementById('ExperimentImage').addEventListener('m
ouseover', function() {
this.style.border = '5px solid red'; // Change border color to
red
});
        // Event listener to log the key pressed by the user
document.addEventListener('keydown', function(event) {
console.log("Key pressed: " + event.key);
});
        // Event listener for mouseout on the image to change
        its border color
document.getElementById('ExperimentImage').addEventListener('m
ouseout', function() {
this.style.border = 'none'; // remove the border
});
</script>
</body>
</html>
```

Output:


Event Listeners Experiment
127.0.0.1:5500/Program4.html
Click Me

Experiment 5:

Build a React application to track issues. Display a list of issues (use static data). Each issue should have a title, description, and status (e.g., Open/Closed). Render the list using a functional component.

Step 1: Install Node.js and npm

Before you start, ensure you have **Node.js** and **npm** (Node Package Manager) installed on your machine.

- Download and install the latest version of Node.js.
- After installing, you can check the versions of Node.js and npm in the terminal:

```
node -v
npm -v
```

Step 2: Create a folder

- Create a folder in **any location**.
- Open folder in **vscode**.

Step 3: Open Integrated vscode Terminal

- Type below command in your terminal to create **react app**.

```
npx create-react-app issue-tracker
```

- Once the process is complete, you'll see a folder named **`issue-tracker`** in your directory. Navigate into the folder:

```
cd issue-tracker
```

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

const issues = [
  {
    id: 1,
    title: "Bug in login page",
    description: "The login page throws an error when submitting invalid credentials.",
    status: "Open",
  },
  {
    id: 2,
    title: "UI glitch on homepage",
    description: "There is a UI misalignment issue on the
homepage for smaller screens.",
    status: "Closed",
  },
  {
    id: 3,
    title: "Missing translation for settings page",
    description: "The settings page is missing translations
for the Spanish language.",
    status: "Open",
  },
  {
    id: 4,
    title: "Database connection error",
    description: "Intermittent database connection issue
during peak hours.",
    status: "Open",
  },
];

const Issue = ({ title, description, status }) => {
  return (
    <div className="issue">
      <h3>{title}</h3>
      <p>{description}</p>
      <span                                className={`status
${status.toLowerCase()}`}>{status}</span>
    </div>
  );
};

const App = () => {
  return (
    <div className="App">
      <h1>Issue Tracker</h1>
      <div className="issue-list">
```

```
        {issues.map((issue) => (
          <Issue
            key={issue.id}
            title={issue.title}
            description={issue.description}
            status={issue.status}
          />
        ))}
      </div>
    </div>
  );
};
```

App.css:

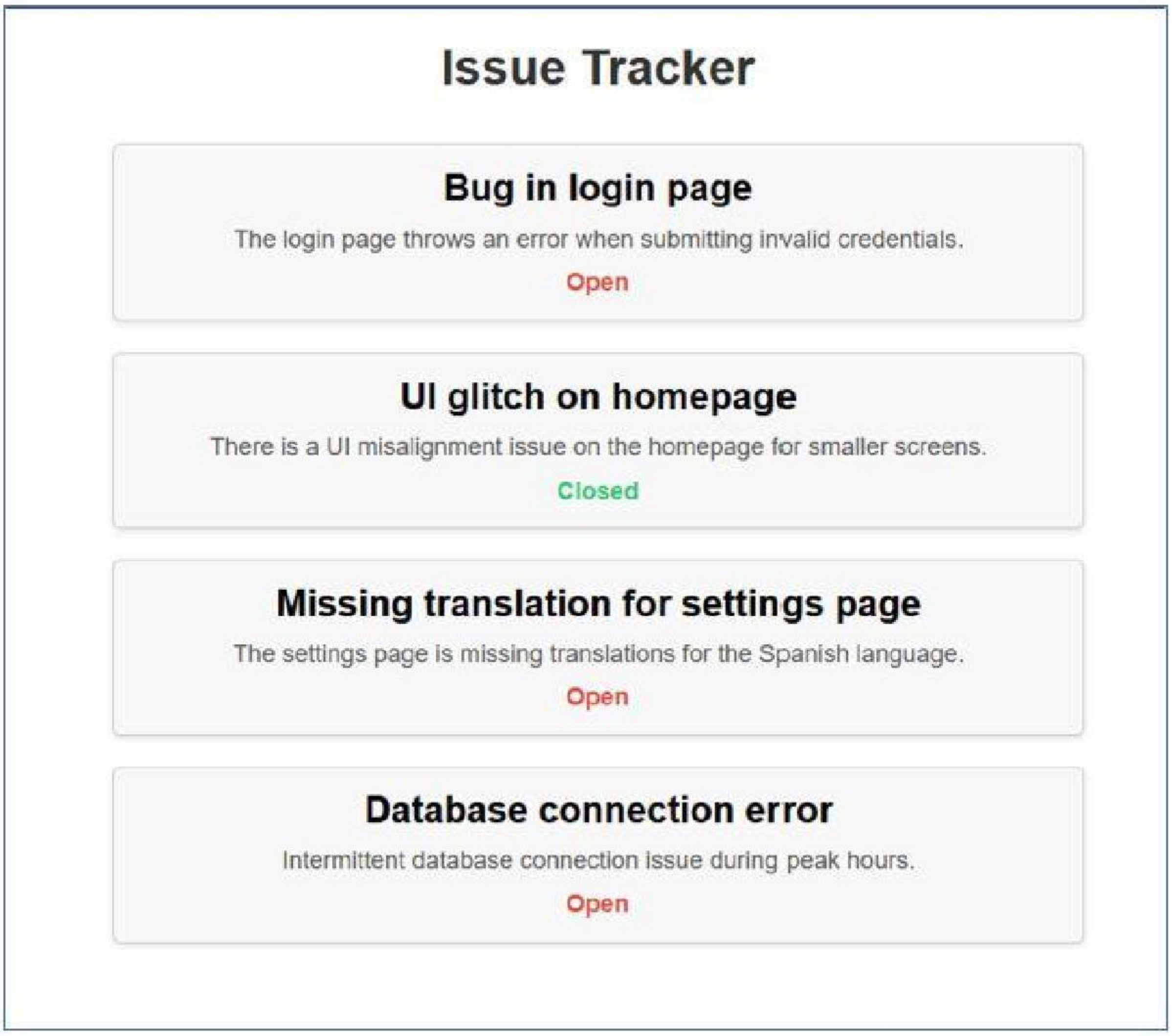
```
.App {
  font-family: Arial, sans-serif;
  text-align: center;
  margin: 20px;
}

h1 {
  color: #333;
}

.issue-list {
  display: flex;
  flex-direction: column;
  align-items: center;
}

.issue {
  background-color: #f9f9f9;
  border: 1px solid #ccc;
  border-radius: 5px;
  padding: 15px;
  margin: 10px;
  width: 80%;
  max-width: 600px;
  box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}

.issue h3 {
  margin: 0 0 10px;
  font-size: 1.5em;
}

.issue p {
  margin: 0 0 10px;
  color: #555;
}

.status {
  font-weight: bold;
}

.status.open {
  color: #e74c3c;
}

.status.closed {
  color: #2ecc71;
}
```

Step 5: Start the Development Server

Now, you can start the development server by using below command:

```
npm start
```

OUTPUT



Issue Tracker

Bug in login page

The login page throws an error when submitting invalid credentials.

Open

UI glitch on homepage

There is a UI misalignment issue on the homepage for smaller screens.

Closed

Missing translation for settings page

The settings page is missing translations for the Spanish language.

Open

Database connection error

Intermittent database connection issue during peak hours.

Open

Experiment 6:

Create a component Counter with A state variable count initialized to 0. Create Buttons to increment and decrement the count. Simulate fetching initial data for the Counter component using useEffect (functional component) or componentDidMount (class component). Extend the Counter component to Double the count value when a button is clicked. Reset the count to 0 using another button.

Step 1: Create Project

- Create a folder and open with **vscode**.
- Open **integrated vscode terminal**.
- After then **create a react** app using below command.

```
npx create-react-app counter-app
```

- After creating successfully then change the directory using below command.

```
cd counter-app
```

Step 2: Create File inside the src folder

- Create two separate file `Counter.js` and `CounterClass.js` inside the **src** folder.
- After then copy the below **code and paste** within the respective file `Counter.js` and `CounterClass.js`
- After then Modify the `App.js` file present in the `src` folder. Copy the below code and paste it.
- After then Modify the `App.css` file present in the `src` folder. Copy the below code and paste it.

Counter.js:

```
import React, { useState, useEffect } from 'react';
const Counter = () => {
  const [count, setCount] = useState(0);
  useEffect(() => {
    console.log('Fetching initial data...');
  }, []);
  const increment = () => setCount(prevCount => prevCount +
1);
  const decrement = () => setCount(prevCount => prevCount -
1);
  const doubleCount = () => setCount(prevCount => prevCount *
2);
  const resetCount = () => setCount(0);
  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={increment}>Increment</button>
      <button onClick={decrement}>Decrement</button>
      <button onClick={doubleCount}>Double</button>
      <button onClick={resetCount}>Reset</button>
    </div>
  );
};

export default Counter;
```

CounterClass.js:

```
import React, { Component } from 'react';

class Counter extends Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  componentDidMount() {
    console.log('Fetching initial data...');
  }

  increment = () => {
    this.setState(prevState => ({ count: prevState.count + 1
}));
  };

  decrement = () => {
    this.setState(prevState => ({ count: prevState.count - 1
}));
  };

  doubleCount = () => {
    this.setState(prevState => ({ count: prevState.count * 2
}));
  };

  resetCount = () => {
    this.setState({ count: 0 });
  };

  render() {
    return (
      <div>
        <h1>Count: {this.state.count}</h1>
        <button onClick={this.increment}>Increment</button>
        <button onClick={this.decrement}>Decrement</button>
        <button onClick={this.doubleCount}>Double</button>
        <button onClick={this.resetCount}>Reset</button>
      </div>
    );
  }
}

export default Counter;
```

App.js:

```
import React from 'react';
import './App.css';
import Counter from './CounterClass';

function App() {
  return (
    <div className="App">
      <h1 className='head'>Welcome to the Counter App</h1>
      <Counter />
    </div>
  );
}

export default App;
```

App.css:

```
.App {
  border-radius: 7px;
  background: #ffeaea;
  text-align: center;
}

#root {
  margin: 20px auto;
  width: 40%;
  height: 400px;
}

h1 {
  color: #fff;
  font-size: 25px;
  border-radius: 6px;
  margin: 20px auto;
  background: #000000;
  width: fit-content;
  padding: 10px 60px;
}

.head {
  width: 100%;
  font-size: 25px;
  margin: 0;
  border-radius: 6px;
  color: #fff;
  background: #000000;
  padding: 10px 0;
}

button {
  margin: 10px;
  padding: 8px 10px;
  font-size: 16px;
  border: none;
  font-weight: 600;
  border-radius: 5px;
  cursor: pointer;
  background: #e02020;
  color: #fff;
}
```

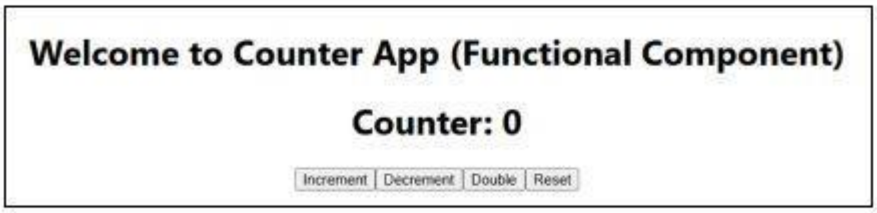
Step 3: Start the Development Server

Now, you can start the development server by using below command:

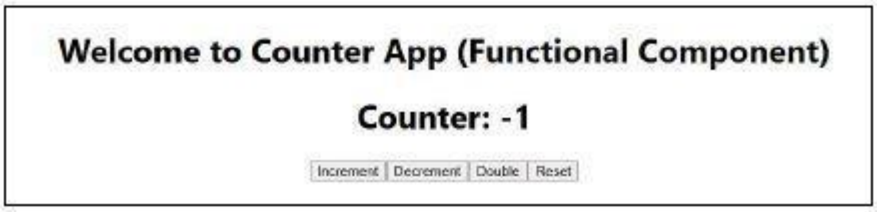
```
npm start
```

This will start a local development server and automatically open your new React application in your default web browser. The server will reload the page automatically whenever you make changes to your code.

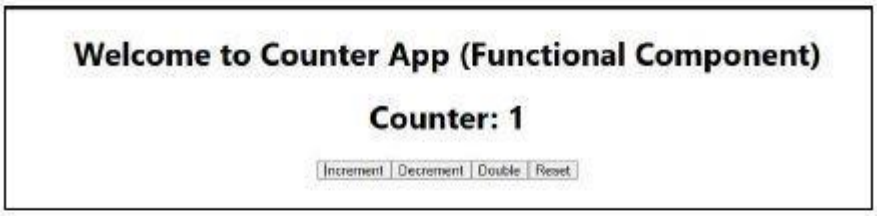
OUTPUT:


Welcome to Counter App (Functional Component)

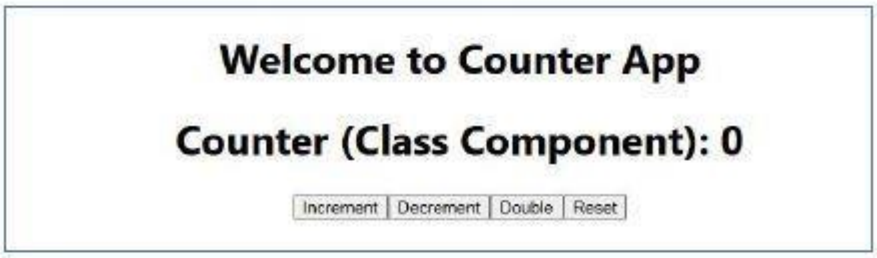
Counter: 0


Welcome to Counter App (Functional Component)

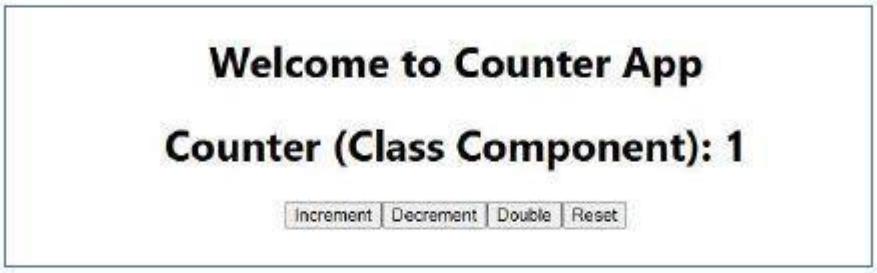
Counter: -1


Welcome to Counter App (Functional Component)

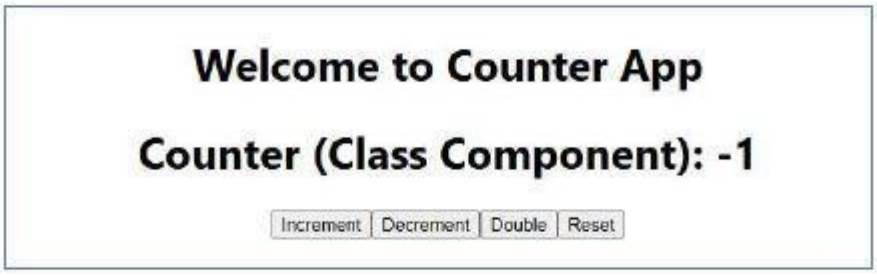
Counter: 1


Welcome to Counter App

Counter (Class Component): 0


Welcome to Counter App

Counter (Class Component): 1


Welcome to Counter App

Counter (Class Component): -1

Experiment 7:

Install Express (npm install express).
Set up a basic server that responds with "Hello, Express!" at the root endpoint (GET /).
Create a REST API. Implement endpoints for a Product resource: GET : Returns a list of products. POST : Adds a new product. GET /:id: Returns details of a specific product. PUT /:id: Updates an existing product. DELETE /:id: Deletes a product. Add middleware to log requests to the console. Use express.json() to parse incoming JSON payloads.

Step 1: Create Project

- Create a folder and open with **vscode**.
- Open **integrated vscode terminal**.

Step 2: Install Express:

- Execute one by one below command.

```
mkdir express-api

cd express-api

npm init -y

npm install express
```

Step 3: Create server.js

- Create **`server.js`** file inside the **`express-api`** folder. Right click on the **`express-api`** folder and create file **`server.js`**.
- After then copy the below code and paste it in that **`server.js`** file, save it.

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

// Middleware to log requests to the console
app.use((req, res, next) => {
    console.log(`${req.method} request made to: ${req.url}`);
    next();
});

// Middleware to parse JSON payloads
app.use(express.json());

// Root endpoint to return "Hello, Express!"
```

```
app.get('/', (req, res) => {
    res.send('Hello, Express!');
});

// Sample in-memory data for products
let products = [
    { id: 1, name: 'Laptop', price: 1000 },
    { id: 2, name: 'Phone', price: 500 }
];

// GET /products: Returns a list of all products
app.get('/products', (req, res) => {
    res.json(products);
});

// POST /products: Adds a new product
app.post('/products', (req, res) => {
    const { name, price } = req.body;
    if (!name || !price) {
        return res.status(400).json({ message: 'Name and price
are required' });
    }
    const newProduct = { id: products.length + 1, name, price
};
    products.push(newProduct);
    res.status(201).json(newProduct);
});

// GET /products/:id: Returns details of a specific product
app.get('/products/:id', (req, res) => {
    const product = products.find(p => p.id ===
parseInt(req.params.id));
    if (!product) {
        return res.status(404).json({ message: 'Product not
found' });
    }
    res.json(product);
});

// PUT /products/:id: Updates an existing product
app.put('/products/:id', (req, res) => {
    const product = products.find(p => p.id ===
parseInt(req.params.id));
    if (!product) {
        return  res.status(404).json({  message:  'Product  not
found' });
    }
    const { name, price } = req.body;
    if (!name || !price) {
        return res.status(400).json({ message: 'Name and price
are required' });
    }
    product.name = name;
    product.price = price;
    res.json(product);
});

// DELETE /products/:id: Deletes a product
app.delete('/products/:id', (req, res) => {
    const  productIndex  =  products.findIndex(p  =>  p.id  ===
parseInt(req.params.id));
    if (productIndex === -1) {
        return  res.status(404).json({  message:  'Product  not
found' });
    }
    products.splice(productIndex, 1);
    res.status(204).send();
});

// Start the server on port 3000
const PORT = 3000;
app.listen(PORT, () => {
    console.log(`Server is running on
http://localhost:${PORT}`);
});
```

Step 4: Download and Install Postman:

- **Visit the Postman download page**:
  - Go to the Postman download page.
- **Download Postman**:
  - Click on the **"Download"** button for Windows (or your respective operating system). The website will automatically detect your OS.
- **Run the installer**:
  - Once the download is complete, open the `.exe` file to begin the installation process.
  - Follow the on-screen instructions to install Postman.
- **Launch Postman**:
  - After installation, you can launch Postman from your desktop or Start Menu.

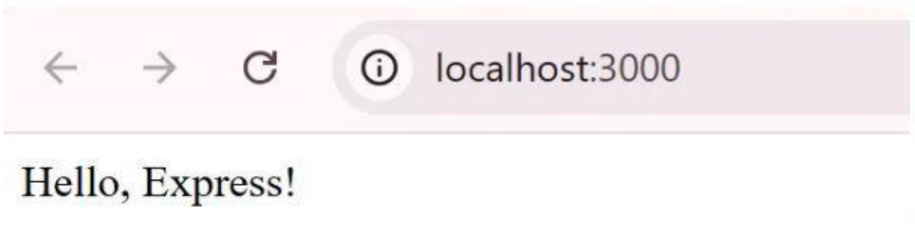
- **Create a Postman Account (Optional)**
  - **Sign up for a Postman account**:
    - Once Postman is installed, open the application. You can **sign up for a free Postman account** or continue using Postman without an account. Signing up will allow you to sync collections across devices, but it is optional.

Step 5: Run the Server:

- Run the server using below command. It will start server to communicate with the postman.

```
node server.js
```

```
node server.js
```



localhost:3000

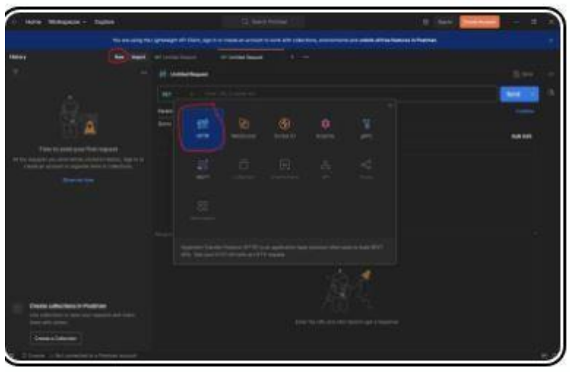
Hello, Express!

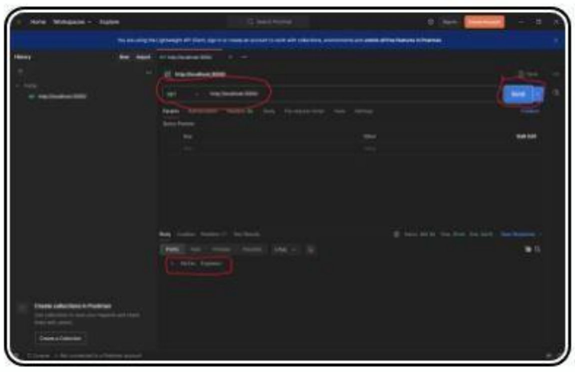
Step 6: Open Postman:

- Once your server is up and running, follow these steps to test the various endpoints.

Step 7: Create a New Request:

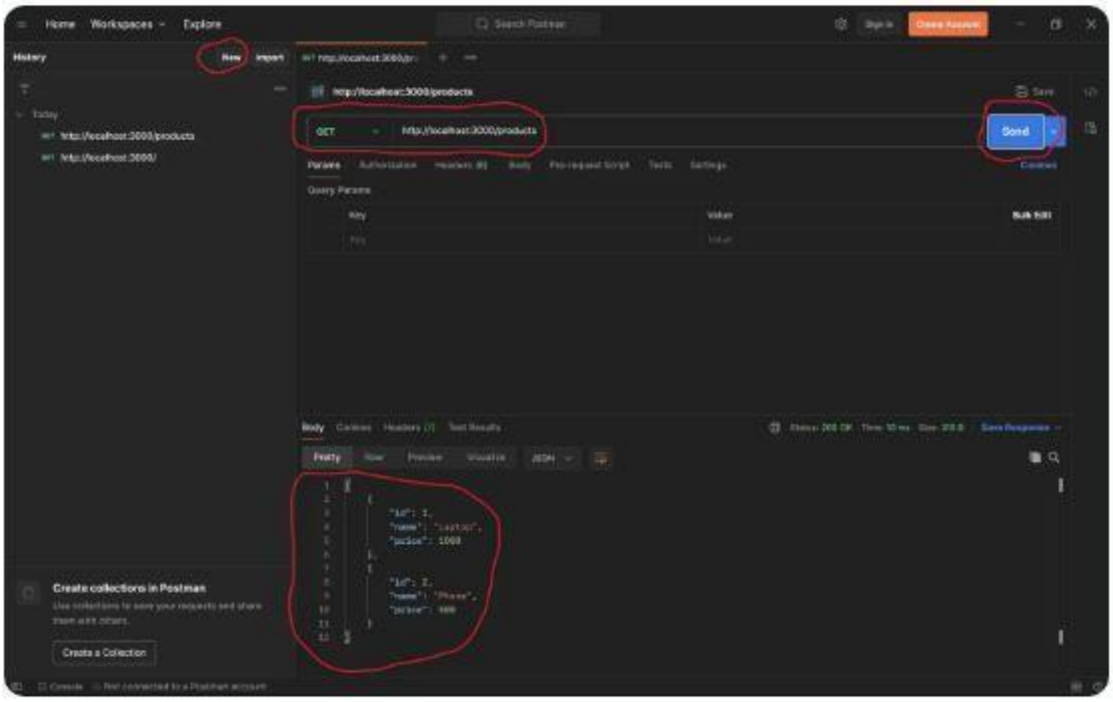
- **Open Postman**.
- Click on the **New** button on the top left, then select **Http** from the options.
- Enter `url` as per **screenshot** or **generated on the terminal** while **running server**.
- **Click on send button** and then you will get **output hello express** as per screenshot.





Step 8: Test Fetching All Products: `GET /products`

- **Steps to Test:**
  - Again follow step 7 and this time Choose **Method**: **`GET`** and the **url** is [`http://localhost:3000/products`](http://localhost:3000/products)
  - Then click on send button to see the output of the all product.

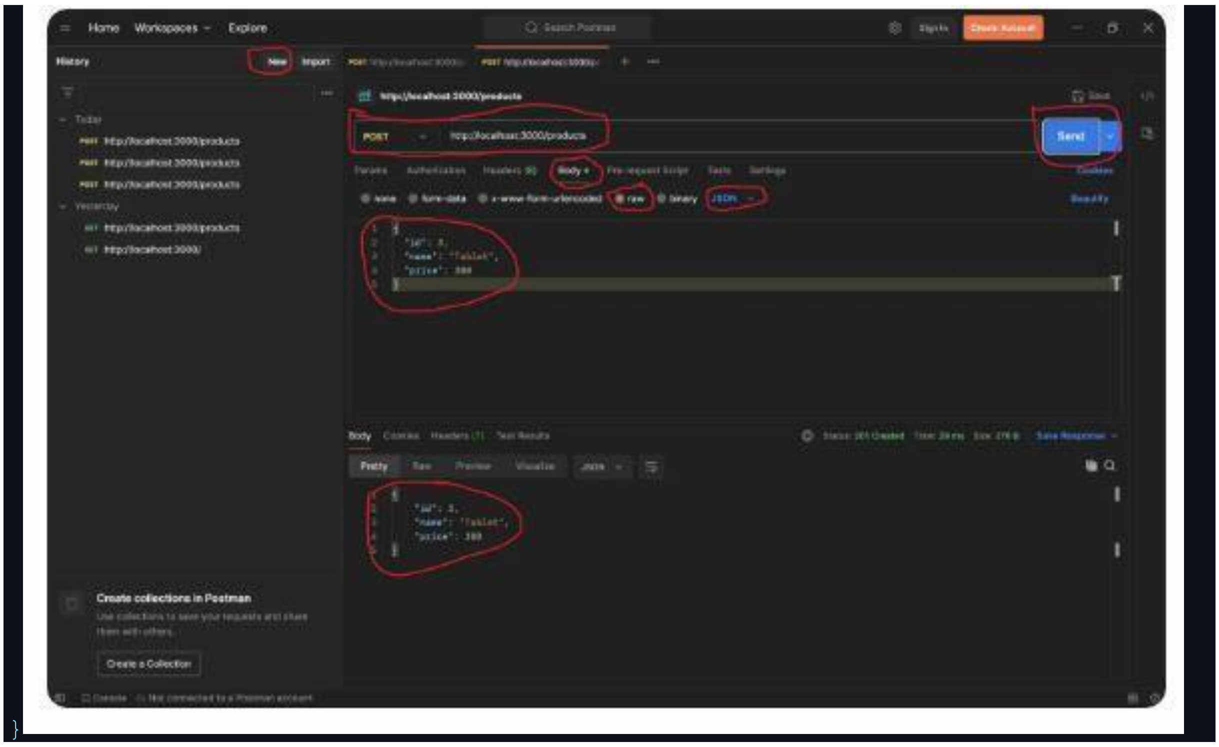


```
[
    {
        "id": 1,
        "name": "Laptop",
        "price": 1000
    },
    {
        "id": 2,
        "name": "Phone",
        "price": 500
    }
]
```

Step 9: Test Adding a New Product: `POST /products`

- **Steps to Test:**
  - Again follow step 7 and this time Choose **Method**: **POST** and the **url** is [`http://localhost:3000/products`](http://localhost:3000/products)
  - Then click on send button you can see message like **"message": "Name and price are required"**
  - **Go to the Body tab** in Postman.
  - **Select `raw` and choose `JSON`** from the dropdown.
  - **Enter the following JSON data**:

```
{
  "id": 3,
  "name": "Tablet",
  "price": 300
}
```



POST http://localhost:3000/products

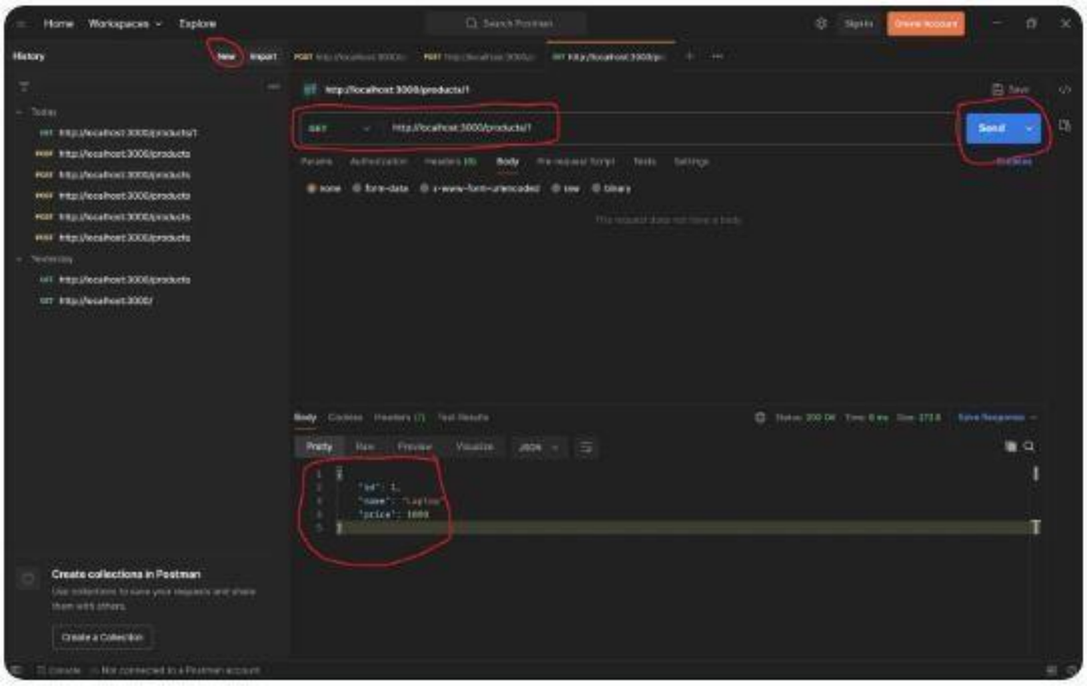
```
{
    "id": 3,
    "name": "Tablet",
    "price": 300
}
```

Status: 201 Created

```
{
    "id": 3,
    "name": "Tablet",
    "price": 300
}
```

Step 10: Test Fetching a Specific Product: `GET /products/:id`

- **Steps to Test:**
    - **Method**: `GET`
    - **URL**: [`http://localhost:3000/products/1`](http://localhost:3000/products/1) (replace `1` with the ID of the product you want to fetch)
    - **Click Send**.



GET http://localhost:3000/products/1

Status: 200 OK

```
{
    "id": 1,
    "name": "Laptop",
    "price": 1000
}
```

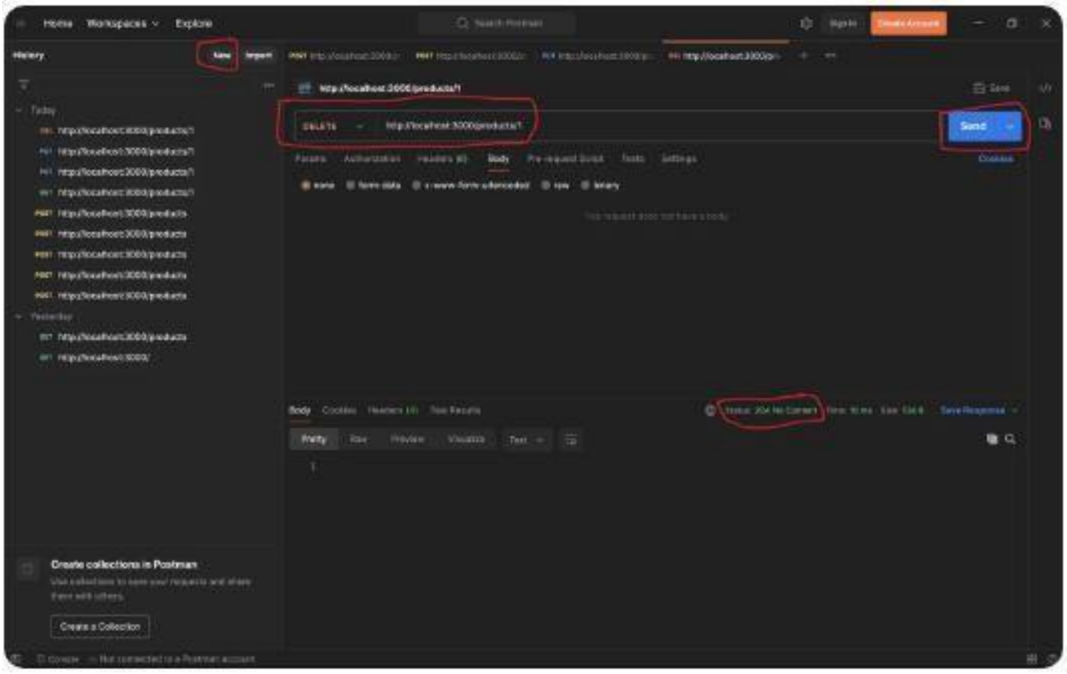
Step 11: Test Updating a Product: `PUT /products/:id`

- **Steps to Test:**
  - **Method**: `PUT`
  - **URL**: http://localhost:3000/products/1 (replace `1` with the ID of the product you want to update)
  - **Go to the Body tab** in Postman.
  - **Select `raw` and choose `JSON`** from the dropdown.
- **Enter the following JSON data**:

```
{
 "id": 1,
 "name": "Updated Laptop",
 "price": 1100
}
```

Step 12: Test Deleting a Product: `DELETE /products/:id`

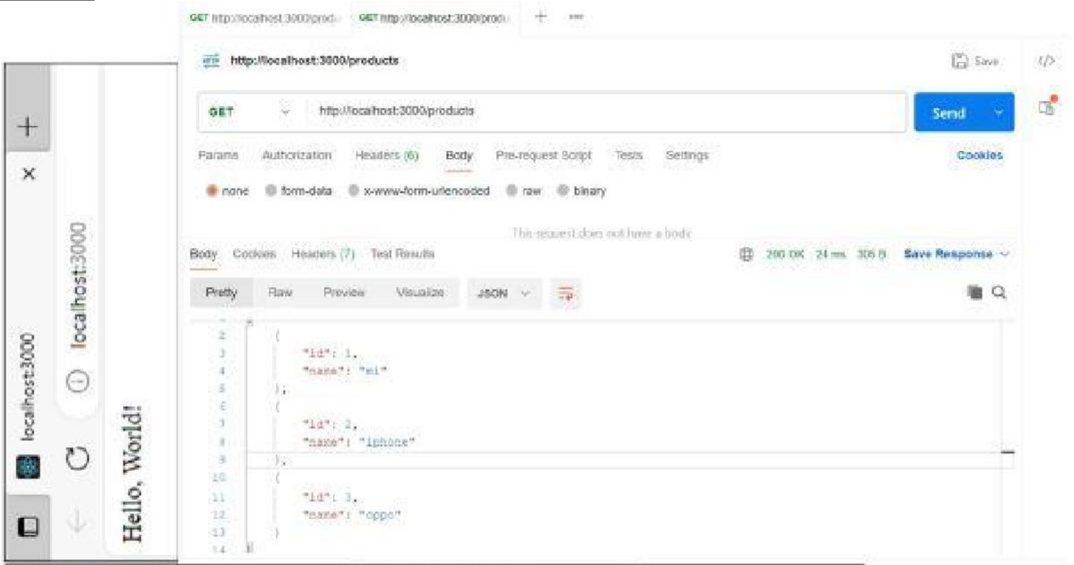
- Steps to Test:
  - **Method**: `DELETE`
  - **URL**: http://localhost:3000/products/1 (replace `1` with the ID of the product you want to delete)
  - **Click Send**.
  - **Expected Response:**
  - You should get a **204 No Content** response, indicating that the product has been deleted successfully.



DELETE http://localhost:3000/products/1

Status: 204 No Content

OUTPUT:



http://localhost:3000/products

GET http://localhost:3000/products

Params Authorization Headers (6) Body Pre-request Script Tests Settings

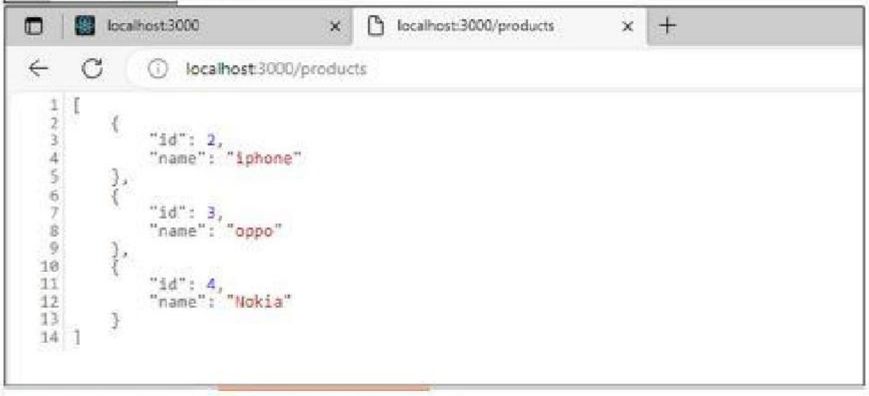
none form-data x-www-form-urlencoded raw binary

This request does not have a body

Body Cookies Headers (7) Test Results 200 OK 24 ms 306 B

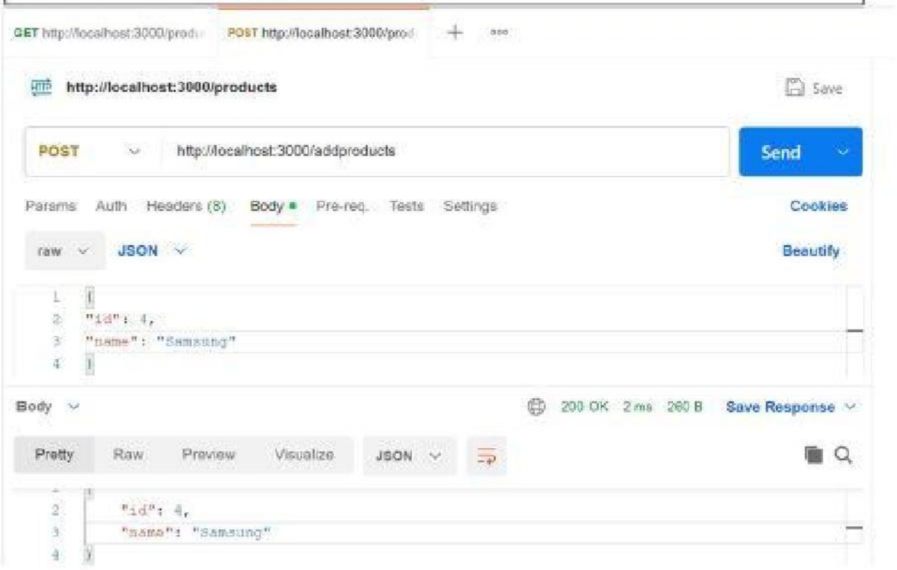
Pretty Raw Preview Visualize JSON

```
    {
        "id": 1,
        "name": "mi"
    },
    {
        "id": 2,
        "name": "iphone"
    },
    {
        "id": 3,
        "name": "oppo"
    }
]
```



localhost:3000/products

```
[
    {
        "id": 2,
        "name": "iphone"
    },
    {
        "id": 3,
        "name": "oppo"
    },
    {
        "id": 4,
        "name": "Nokia"
    }
]
```


GET http://localhost:3000/produ
POST http://localhost:3000/prod
Save
Send
Cookies
Beautify
Save Response

http://localhost:3000/products

POST http://localhost:3000/addproducts

Params Auth Headers (8) Body Pre-req. Tests Settings

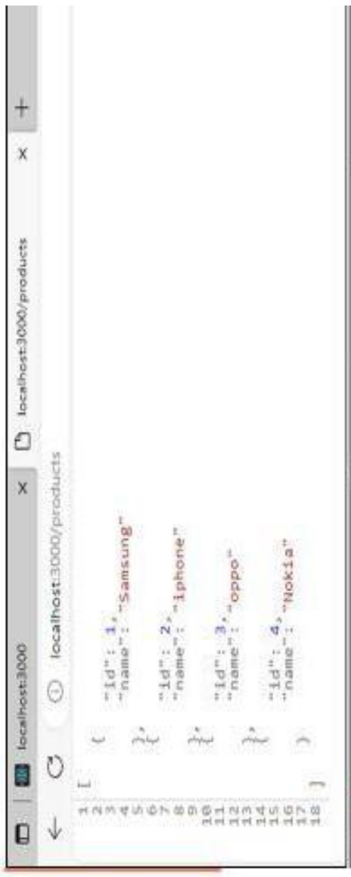
raw JSON

```
{
"id": 4,
"name": "Samsung"
}
```

Body 200 OK 2 ms 260 B

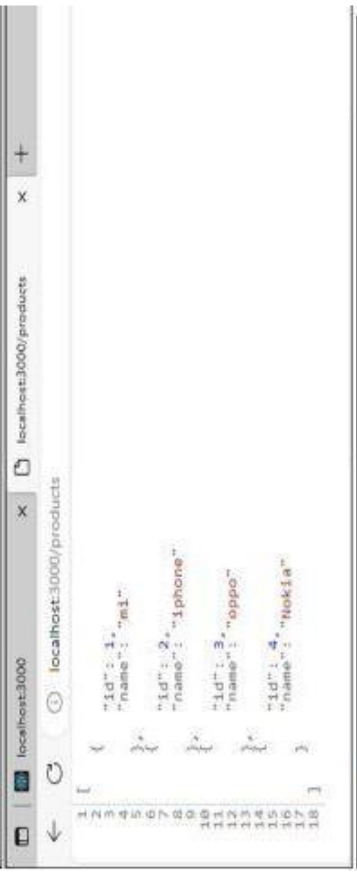
Pretty Raw Preview Visualize JSON

```
{
    "id": 4,
    "name": "Samsung"
}
```



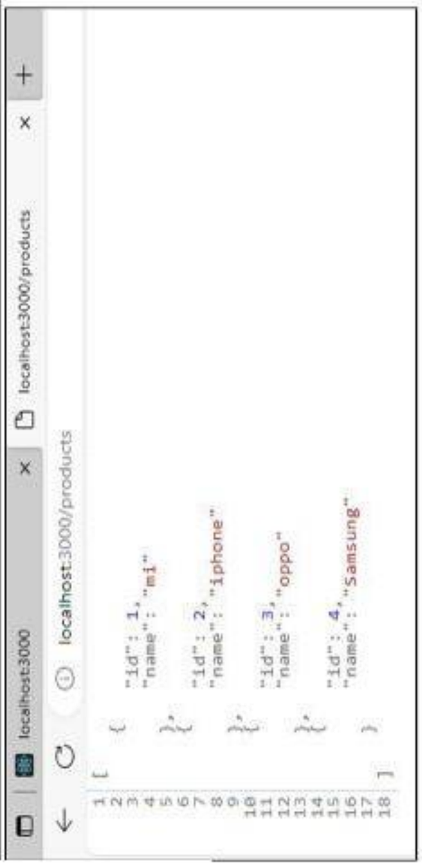
localhost:3000/products

```
[
  {
    "id": 1,
    "name": "Samsung"
  },
  {
    "id": 2,
    "name": "iphone"
  },
  {
    "id": 3,
    "name": "oppo"
  },
  {
    "id": 4,
    "name": "Nokia"
  }
]
```



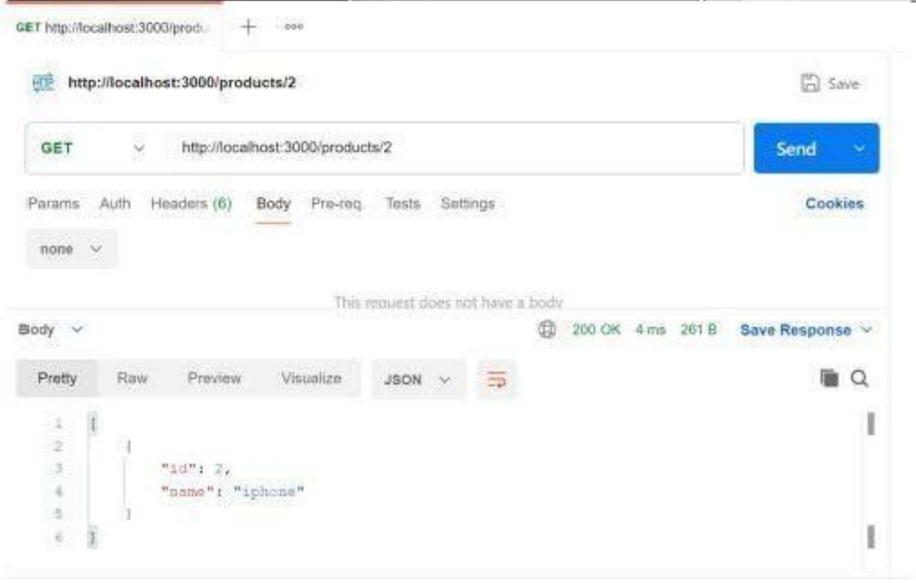
localhost:3000/products

```
[
  {
    "id": 1,
    "name": "mi"
  },
  {
    "id": 2,
    "name": "iphone"
  },
  {
    "id": 3,
    "name": "oppo"
  },
  {
    "id": 4,
    "name": "Nokia"
  }
]
```



localhost:3000/products

```
[
  {
    "id": 1,
    "name": "mi"
  },
  {
    "id": 2,
    "name": "iphone"
  },
  {
    "id": 3,
    "name": "oppo"
  },
  {
    "id": 4,
    "name": "Samsung"
  }
]
```



http://localhost:3000/products/2

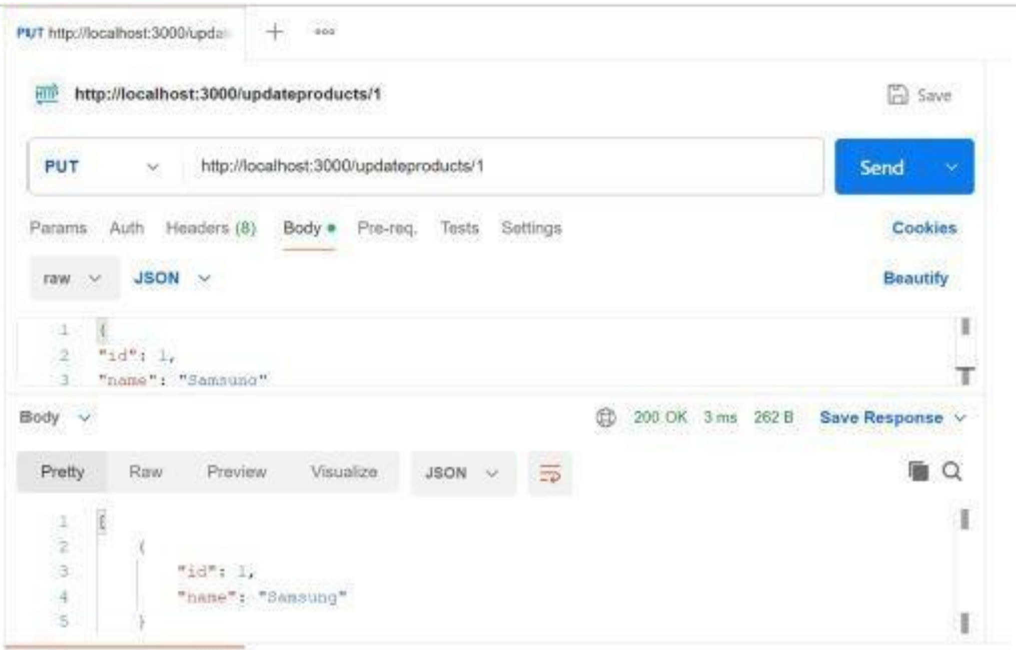
GET http://localhost:3000/products/2

Body: none

This request does not have a body

Body 200 OK 4 ms 261 B

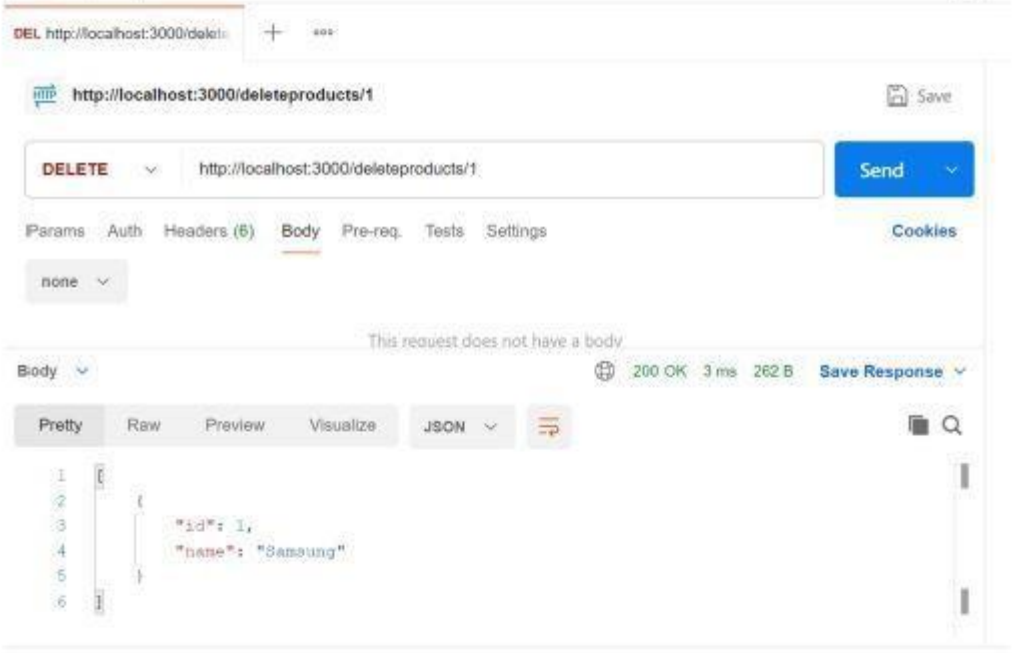
```
[
  {
    "id": 2,
    "name": "iphone"
  }
]
```


PUT http://localhost:3000/updateproducts/1

```
{
"id": 1,
"name": "Samsung"
```

200 OK 3 ms 262 B

```
[
    {
        "id": 1,
        "name": "Samsung"
    }
```


DELETE http://localhost:3000/deleteproducts/1

This request does not have a body

200 OK 3 ms 262 B

```
[
    {
        "id": 1,
        "name": "Samsung"
    }
]
```

Experiment 8:

Install the MongoDB driver for Node.js. Create a Node.js script to connect to the shop database. Implement insert, find, update, and delete operations using the Node.js MongoDB driver. Define a product schema using Mongoose. Insert data into the products collection using Mongoose. Create an Express API with a /products endpoint to fetch all products. Use fetch in React to call the /products endpoint and display the list of products. Add a POST /products endpoint in Express to insert a new product. Update the Product List, After adding a product, update the list of products displayed in React.

Step 1: Create Project:

- **Create a folder** and open with **vscode**.
- Open **integrated vscode terminal**.

Step 2: Set up the Backend:

- Run the following commands to set up the environment for your project:
- Run one by one command to install the mongoose.

```
mkdir shop-backend
cd shop-backend
```

Step 3: Initialize Node.js Project:

```
npm init –y
```

Step 4: Install Dependencies:

```
npm install express mongoose mongodb cors
```

Step 5: Create server.js file:

- **Right click on shop-backend folder** and create the **server.js** file.
- Copy the below **code and paste** it in that file, save it.

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

const app = express();
app.use(cors());
app.use(express.json());

// MongoDB Connection
```

```
mongoose.connect('mongodb://localhost:27017/shop', {
  useNewUrlParser: true,
  useUnifiedTopology: true
});

const db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection
error:'));
db.once('open', () => console.log('Connected to MongoDB'));

// Define Mongoose Schema for Products
const productSchema = new mongoose.Schema({
  name: String,
  price: Number,
  description: String
});

const Product = mongoose.model('Product', productSchema);

// CRUD API Endpoints

// Fetch all products
app.get('/products', async (req, res) => {
  try {
    const products = await Product.find();
    res.json(products);
  } catch (error) {
    res.status(500).send(error.message);
  }
});

// Add a new product
app.post('/products', async (req, res) => {
  try {
    const newProduct = new Product(req.body);
    await newProduct.save();
    res.status(201).json(newProduct);
  } catch (error) {
    res.status(400).send(error.message);
  }
});

// Update a product
app.put('/products/:id', async (req, res) => {
  try {
    const updatedProduct = await Product.findByIdAndUpdate(
      req.params.id,
      req.body,
      { new: true }
    );
    res.json(updatedProduct);
  } catch (error) {
    res.status(400).send(error.message);
  }
});

// Delete a product
app.delete('/products/:id', async (req, res) => {
  try {
    await Product.findByIdAndDelete(req.params.id);
    res.status(204).send();
  } catch (error) {
    res.status(500).send(error.message);
  }
});

// Start Express server
const PORT = 4000;
app.listen(PORT, () => console.log(`Server running on port
${PORT}`));
```

Step 6: Start the backend:

- **Start the backend server** using below command to see in the terminal **message server is running on port 4000, connected to MongoDB**.

```
node server.js
```

Step 6: Set up the Frontend:

- **Open new terminal** without **deleting running backend server** terminal.
- **Execute 1st command** to create **frontend folder.**
- After successfully creating **product-client** folder then **change the directory using 2nd command**.

```
npx create-react-app product-client
cd product-client
```

Step 7: Edit App.js and App.css file:

- Copy the below code and paste it into the **App.js** file and then save it.
- After then Copy the below code and paste it into the **App.css** file and then save it.

App.js:

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

const App = () => {
  const [products, setProducts] = useState([]);
  const [newProduct, setNewProduct] = useState({
    name: '',
    price: '',
    description: ''
  });
  const [editProduct, setEditProduct] = useState(null);
  const [errors, setErrors] = useState({});

  const fetchProducts = () => {
    fetch('http://localhost:4000/products')
      .then(response => response.json())
      .then(data => setProducts(data));
  };

  useEffect(() => {
    fetchProducts();
  }, []);

  const handleInputChange = (e) => {
    const { name, value } = e.target;
    if (editProduct) {
      setEditProduct({ ...editProduct, [name]: value });
    } else {
      setNewProduct({ ...newProduct, [name]: value });
    }
    setErrors({ ...errors, [name]: '' });
  };

  const validateInputs = (product) => {
    const newErrors = {};
    if (!product.name) newErrors.name = 'Product name is
required.';
    if (!product.price) newErrors.price = 'Product price is
required.';
    if (!product.description) newErrors.description =
'Product description is required.';
    setErrors(newErrors);
    return Object.keys(newErrors).length === 0;
  };

  const addProduct = () => {
    if (!validateInputs(newProduct)) return;

    fetch('http://localhost:4000/products', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(newProduct)
    })
      .then(response => response.json())
      .then(() => {
        fetchProducts();
        setNewProduct({ name: '', price: '', description: ''
});
      });
  };

  const updateProduct = () => {
    if (!validateInputs(editProduct)) return;


fetch(`http://localhost:4000/products/${editProduct._id}`, {
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(editProduct)
    })
      .then(response => response.json())
      .then(() => {
        fetchProducts();
        setEditProduct(null);
      });
  };

  const deleteProduct = (productId) => {
    if (window.confirm('Are you sure you want to delete this
product?')) {
      fetch(`http://localhost:4000/products/${productId}`, {
        method: 'DELETE'
      })
```

```
          .then(() => fetchProducts());
      }
    };

    return (
      <div className="container">
        <h1 className="title">Product Management</h1>

        {products.length === 0 ? (
          <p className="no-products-message">No products
available. Please add a new product.</p>
        ) : (
          <table className="product-table">
            <thead>
              <tr>
                <th>Name</th>
                <th>Price</th>
                <th>Description</th>
                <th>Actions</th>
              </tr>
            </thead>
            <tbody>
              {products.map(product => (
                <tr key={product._id} className="product-row">
                  <td>{product.name}</td>
                  <td>₹{product.price}</td>
                  <td>{product.description}</td>
                  <td>
                    <div className="action-buttons">
                      <button onClick={() =>
setEditProduct(product)} className="edit-button">Edit</button>
                      <button onClick={() =>
deleteProduct(product._id)} className="delete-
button">Delete</button>
                    </div>
                  </td>
                </tr>
              ))}
            </tbody>
          </table>
        )}

        <div className="form-section">
```

```
         <h2 className="form-title">{editProduct ? 'Edit
Product' : 'Add New Product'}</h2>

         <div className="input-group">
           <div className="input-wrapper">
             <input
               type="text"
               name="name"
               value={editProduct ? editProduct.name :
newProduct.name}
               onChange={handleInputChange}
               placeholder="Product Name"
               className="input-field"
             />
             {errors.name && <p className="error-
message">{errors.name}</p>}
           </div>

           <div className="input-wrapper">
             <input
               type="number"
               name="price"
               value={editProduct ? editProduct.price :
newProduct.price}
               onChange={handleInputChange}
               placeholder="Product Price ₹"
               className="input-field"
             />
             {errors.price && <p className="error-
message">{errors.price}</p>}
           </div>

           <div className="input-wrapper">
             <textarea
               name="description"
               value={editProduct ? editProduct.description :
newProduct.description}
               onChange={handleInputChange}
               placeholder="Product Description"
               className="input-field"
               rows={3}
             />
             {errors.description && <p className="error-
message">{errors.description}</p>}
```

```
            </div>
          </div>

          {editProduct ? (
            <button onClick={updateProduct} className="update-
button">Update Product</button>
          ) : (
            <button onClick={addProduct} className="add-
button">Add Product</button>
          )}
        </div>
      </div>
    );
  };

  export default App;
```

App.css:

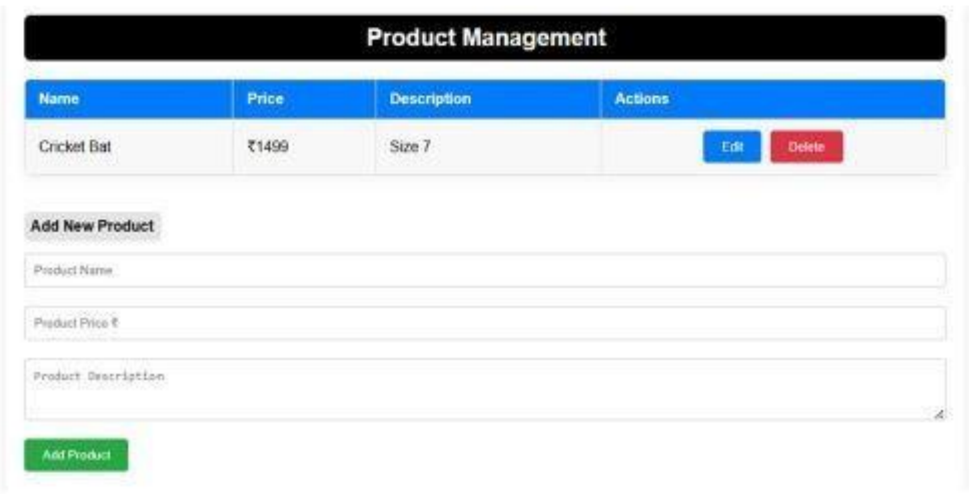
```
container {
  max-width: 1000px;
  margin: 0 auto;
  padding: 20px;
  font-family: Arial, sans-serif;
  background-color: #ffffff;
  box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
  border-radius: 8px;
}

.title {
  border-radius: 7px;
  color: #fff;
  background: #000;
  padding: 10px 0;
  font-size: 1.6rem;
  text-align: center;
  margin: 0;
}

.product-table {
  width: 100%;
  border-collapse: collapse;
  margin: 20px 0;
  background-color: #f9f9f9;
  box-shadow: 0 5px 15px rgba(0, 0, 0, 0.05);
  border-radius: 8px;
  overflow: hidden;
}

.product-table thead {
  background-color: #007bff;
  color: white;
  text-align: left;
  font-size: 1rem;
}

.product-table th,
.product-table td {
  padding: 12px 15px;
  border: 1px solid #ddd;
}

.product-table tr {
  transition: background-color 0.2s ease-in-out;
}

td button {
  display: block;
}

.product-table tbody tr:nth-child(even) {
  background-color: #f9f9f9;
}

.input-group {
  display: flex;
  flex-direction: column;
  gap: 10px;
}

.input-field {
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 4px;
}

.add-button,
.edit-button,
.update-button {
  padding: 10px 20px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  transition: background-color 0.2s;
}

.add-button {
  font-weight: 500;
  margin-top: 10px;
  background-color: #28a745;
  color: white;
}

.add-button:hover {
  background-color: #218838;
}

.edit-button {
  background-color: #007bff;
  color: white;
}

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

.update-button {
  margin-top: 20px;
  background-color: #ffc107;
  color: black;
  font-weight: 600;
}

.update-button:hover {
  background-color: #e0a800;
}

.action-buttons {
  display: flex;
  gap: 10px;
  justify-content: center;
  align-items: center;
```

```
}

.form-section {
  margin-top: 40px;
}

.form-title {
  width: fit-content;
  background: #0000001a;
  padding: 7px 7px;
  border-radius: 7px;
  font-size: 16px;
  color: #000000;
}

.delete-button {
  background-color: #dc3545;
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

.delete-button:hover {
  background-color: #c82333;
}

.no-products-message {
  text-align: center;
  font-size: 1.2rem;
  color: #666;
  margin-top: 20px;
  padding: 10px;
  background-color: #f9f9f9;
  border-radius: 8px;
}

.error-message {
  color: red;
  background-color: #ffe6e6;
  padding: 5px;
  border-radius: 5px;
  margin-top: 5px;
  font-size: 0.9rem;
}

.input-wrapper {
  display: flex;
  flex-direction: column;
  justify-content: center;
  margin-bottom: 10px;
}
```

Step 8: Start the development server:

- Copy the below command to run the frontend to see the app.

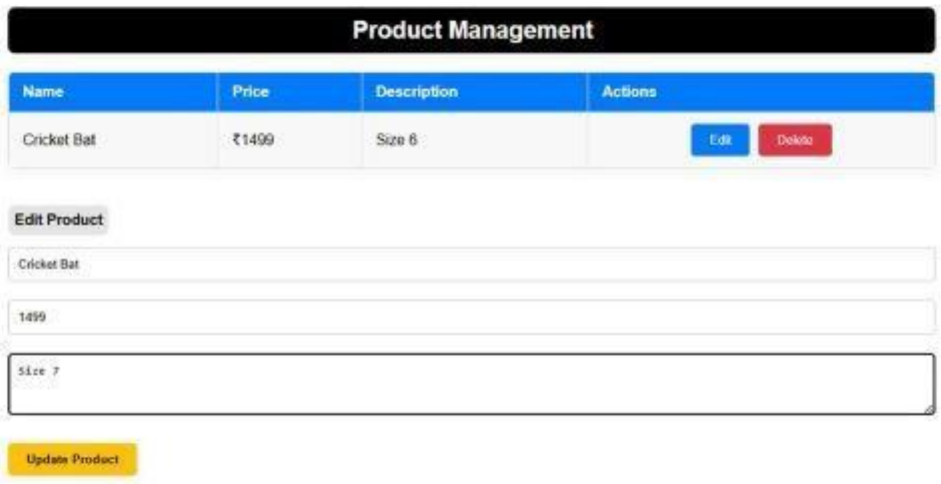
```
npm start
```



Product Management

Name	Price	Description	Actions
Cricket Bat	₹1499	Size 7	Edit Delete

Add New Product



Product Management

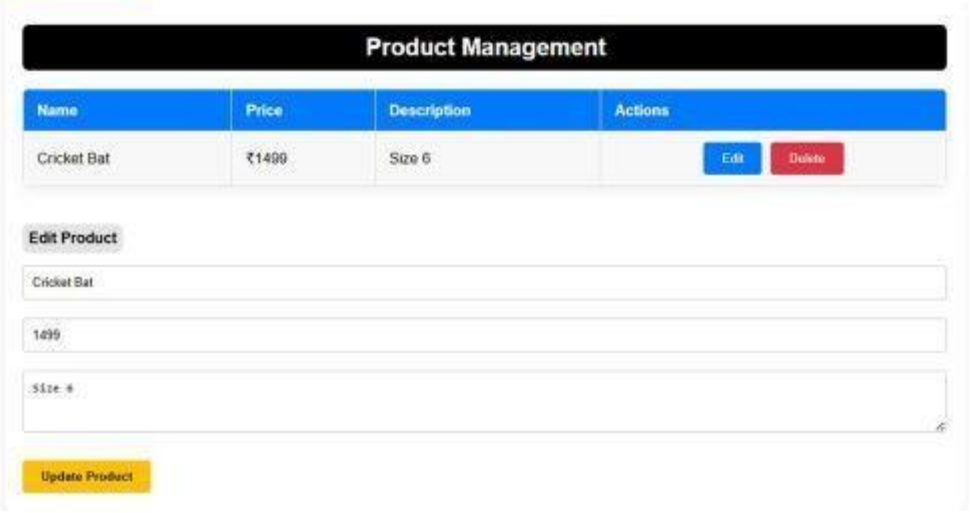
Name	Price	Description	Actions
Cricket Bat	₹1499	Size 6	Edit Delete

Edit Product

Cricket Bat

1499

Size 7



Product Management

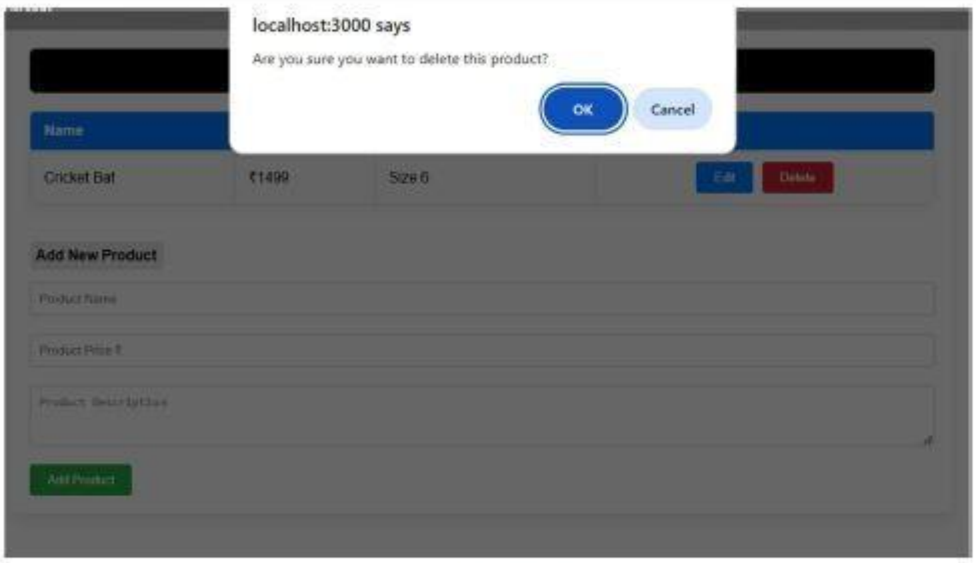
Name	Price	Description	Actions
Cricket Bat	₹1499	Size 6	Edit Delete

Edit Product

Cricket Bat

1499

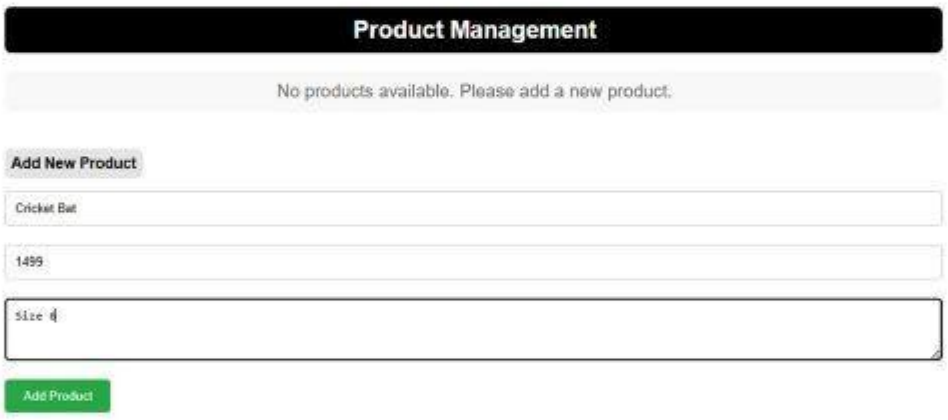
Size 6


localhost:3000 says

Are you sure you want to delete this product?

Name			
Cricket Bat	₹1499	Size 6	Edit Delete

Add New Product


Product Management

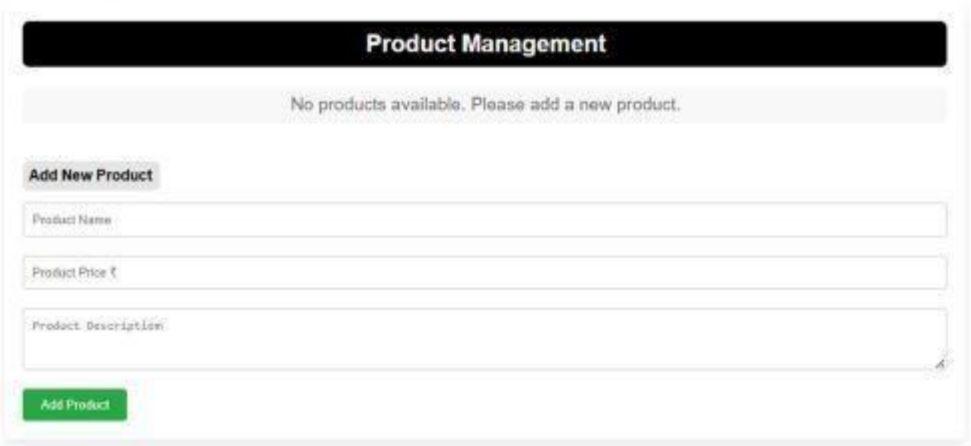
No products available. Please add a new product.

Add New Product

Cricket Bat

1499

Size 6


Product Management

No products available. Please add a new product.

Add New Product