FULL STACK DEVELOPMENT

DEPARTMENT OF ISE

BIS601

	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
	Crs	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

Text Book 1: Chapter 2

- Basic JavaScript Instructions,
- Statements,
- Comments,
- Variables,
- Data Types,
- Arrays,
- Strings,

Text Book 1: Chapter 3

- Functions, Methods & Objects,

Text Book 1: Chapter 4

- Decisions & Loops.

	Module-2
	Text Book 1: Chapter: 5 ○ Document Object Model: ○ DOM Manipulation, Selecting Elements, ○ Working with DOM Nodes, ○ Updating Element Content & Attributes, **Text Book 1: Chapter: 6** ○ Events, ○ Different Types of Events, ○ How to Bind an Event to an Element, ○ Event Delegation, ○ Event Listeners. **Text Book 1: Chapter: 13** ○ Form enhancement and validation.
	Module-3
	Text Book 2: Chapter 1 Introduction to MERN: ○ MERN components, **Text Book 2: Chapter 2** ○ Server less Hello world. **Text Book 2: Chapter 3** ○ React Components: ○ Issue Tracker, React Classes, ○ Composing Components, ○ Passing Data Using Properties, ○ Passing Data Using Children, ○ Dynamic Composition.
	Module-4

	Text Book 2: Chapter 4, 5 **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. **Text Book 2: Chapter 5**

	○ **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.
	Module-5
	Text Book2:Chapter 6 **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. **Text Book 2: Chapter 7** ○ Modularization and Webpack : ○ Back-End Modules ○ Front-End Modules ○ Webpack Transform and Bundle, ○ Libraries Bundle , ○ Hot Module Replacement, ○ Debugging DefinePlugin: ○ Build Configuration, Production Optimization.

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 crs 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 realworld problem from domains such as finance, marketing, medical, or enterprise projects **(25 marks)**.

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.

- **To know the node.js version:**

  node -v

- **Experiment1a.js**
  - 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;
}
```

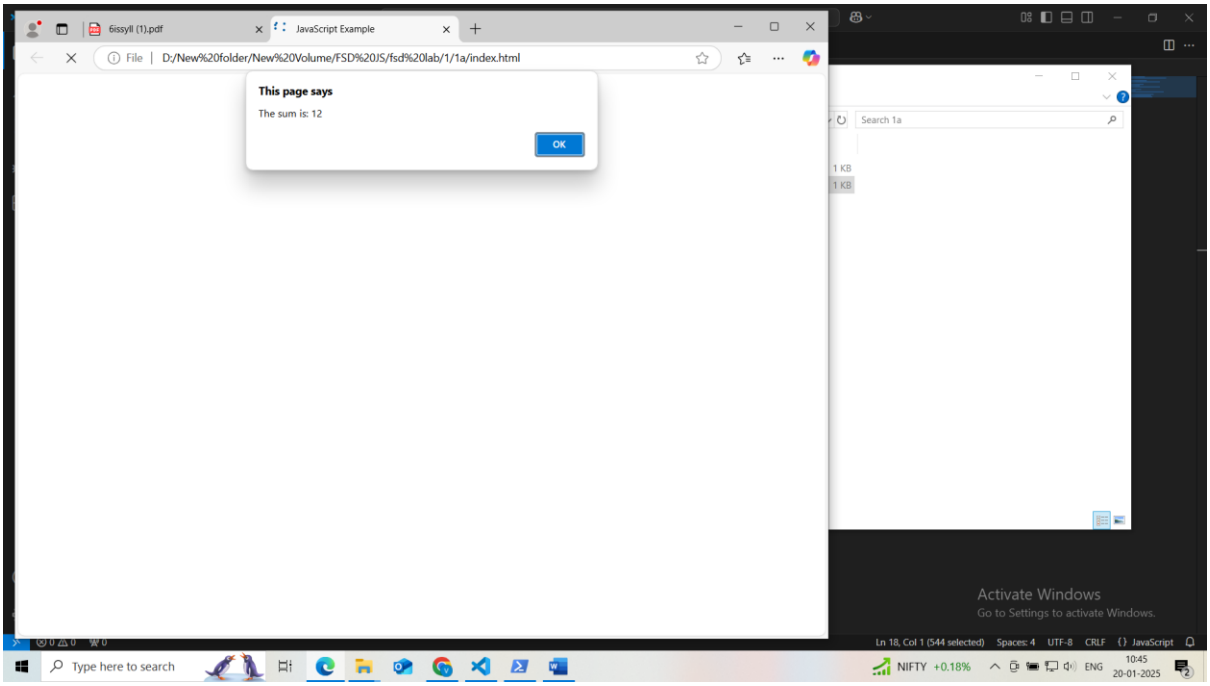
Experiment usage

```
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));
```

index.html to make working of alert function:

o/p:

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



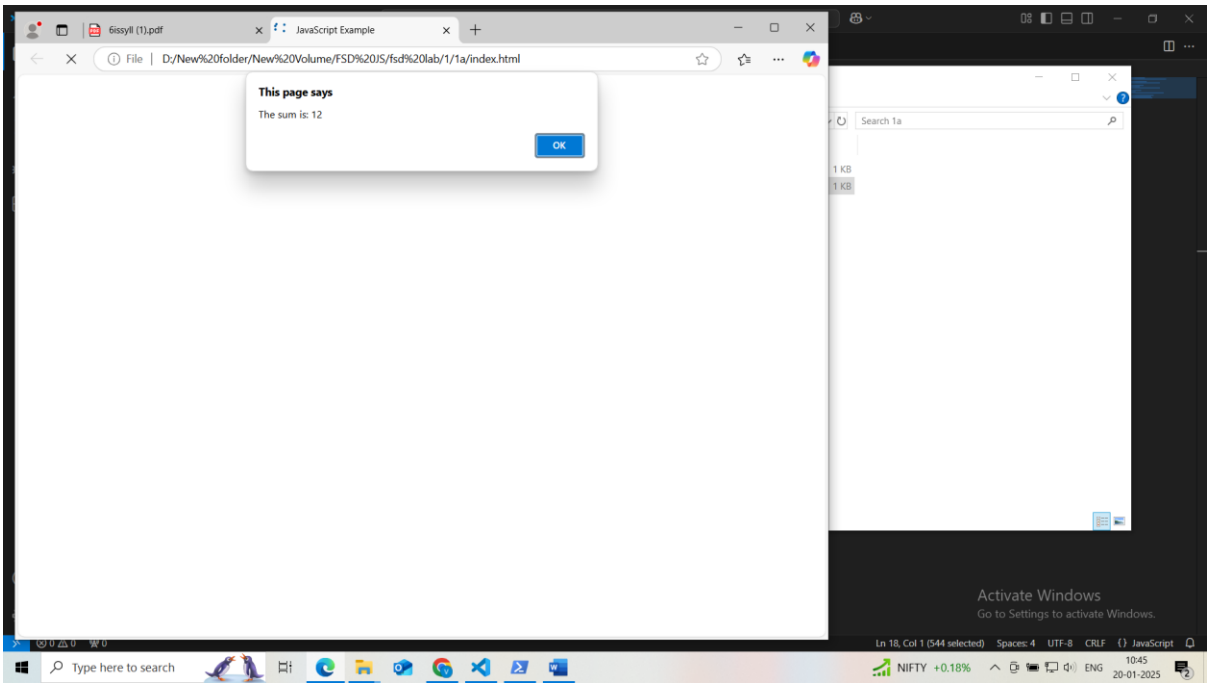
Some other way like whole thing implemented in single html file

- 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>
```

o/p


6issyll (1).pdf
JavaScript Example
File | D:/New%20folder/New%20Volume/FSD%20JS/fsd%20lab/1/1a/index.html
This page says
The sum is: 12
OK
Search 1a
1 KB
1 KB
Activate Windows
Go to Settings to activate Windows.
Ln 18, Col 1 (544 selected) Spaces: 4 UTF-8 CRLF { } JavaScript
Type here to search
NIFTY +0.18%
ENG
10:45
20-01-2025

- 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

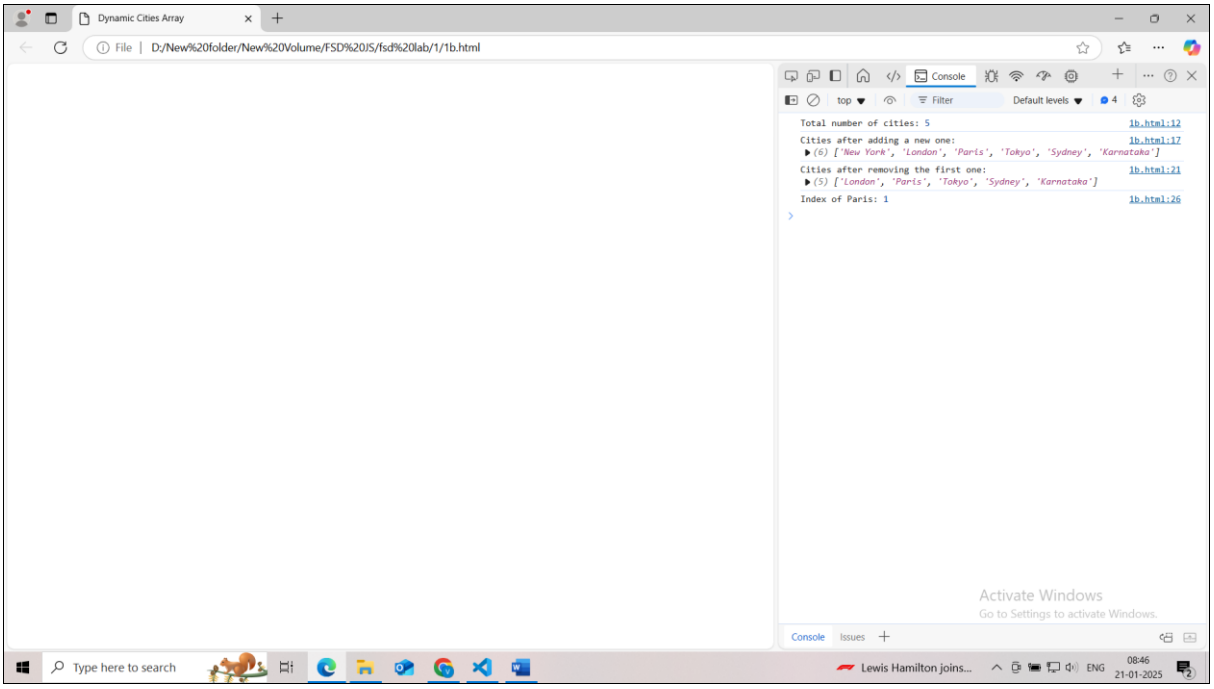
- Next run

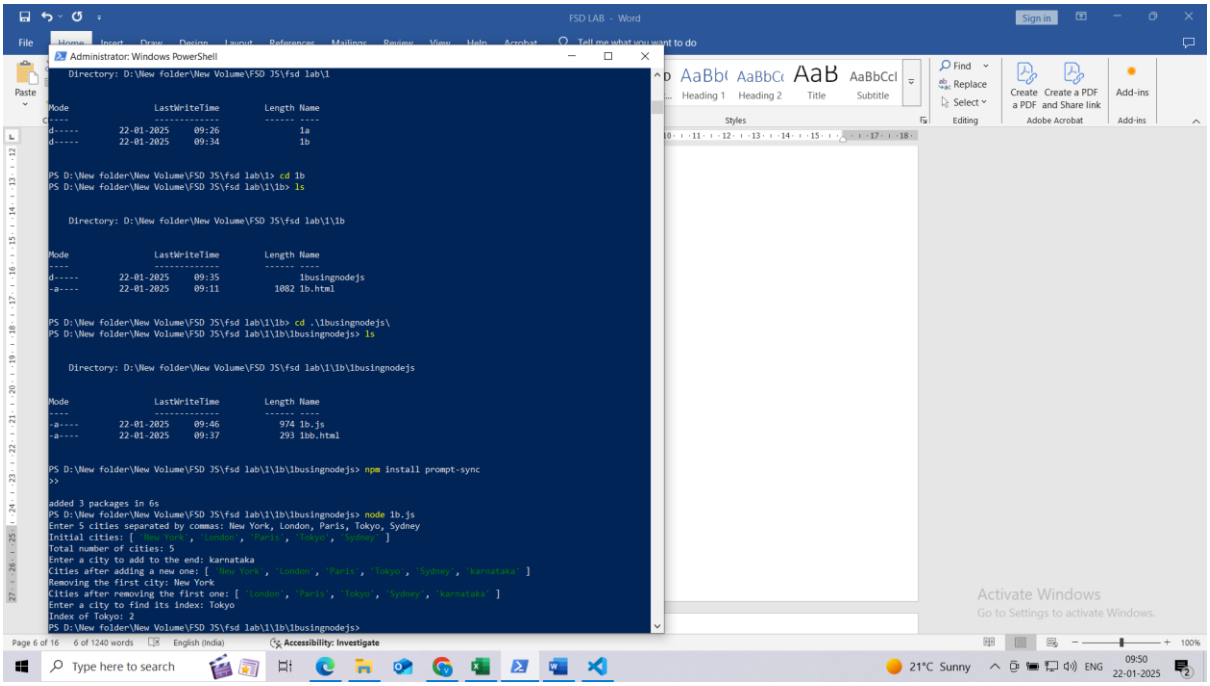
npm install prompt-sync

node 1b.js

- **o/p:** node 1b.js

  i/p: New York, London, Paris, Tokyo, Sydney




FSD LAB - Word
Administrator: Windows PowerShell
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>
Activate Windows
Go to Settings to activate Windows.
Type here to search
21°C Sunny
ENG
09:50
22-01-2025

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).

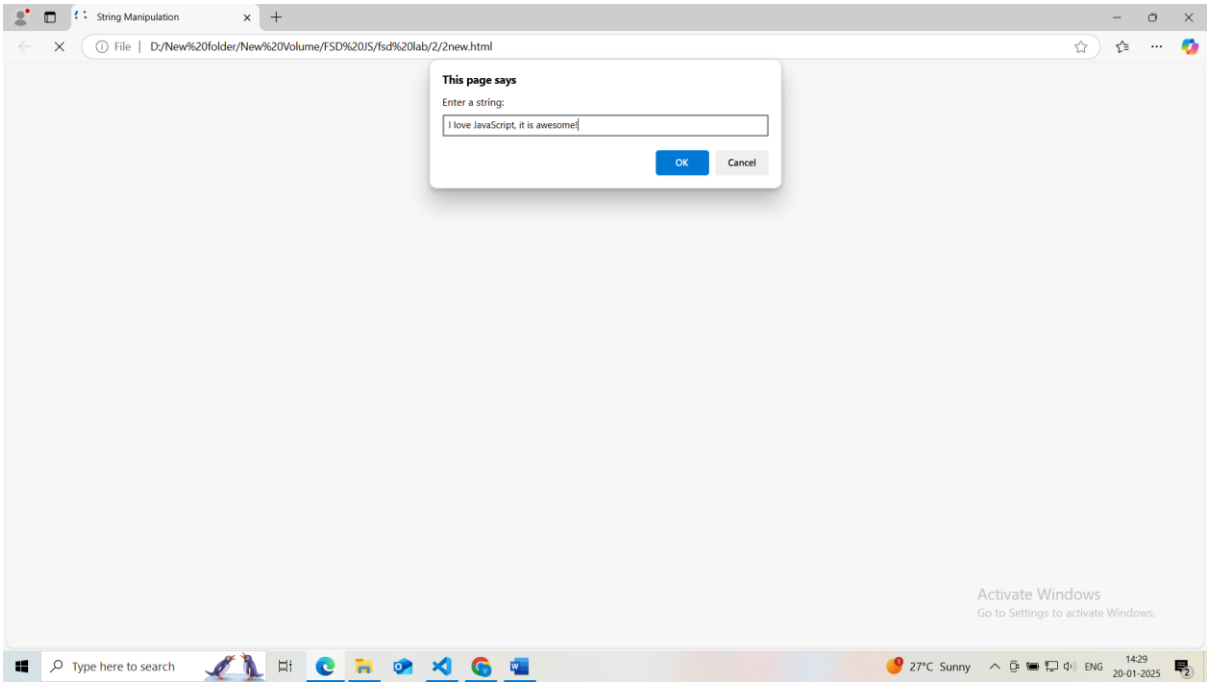
- **Program:**

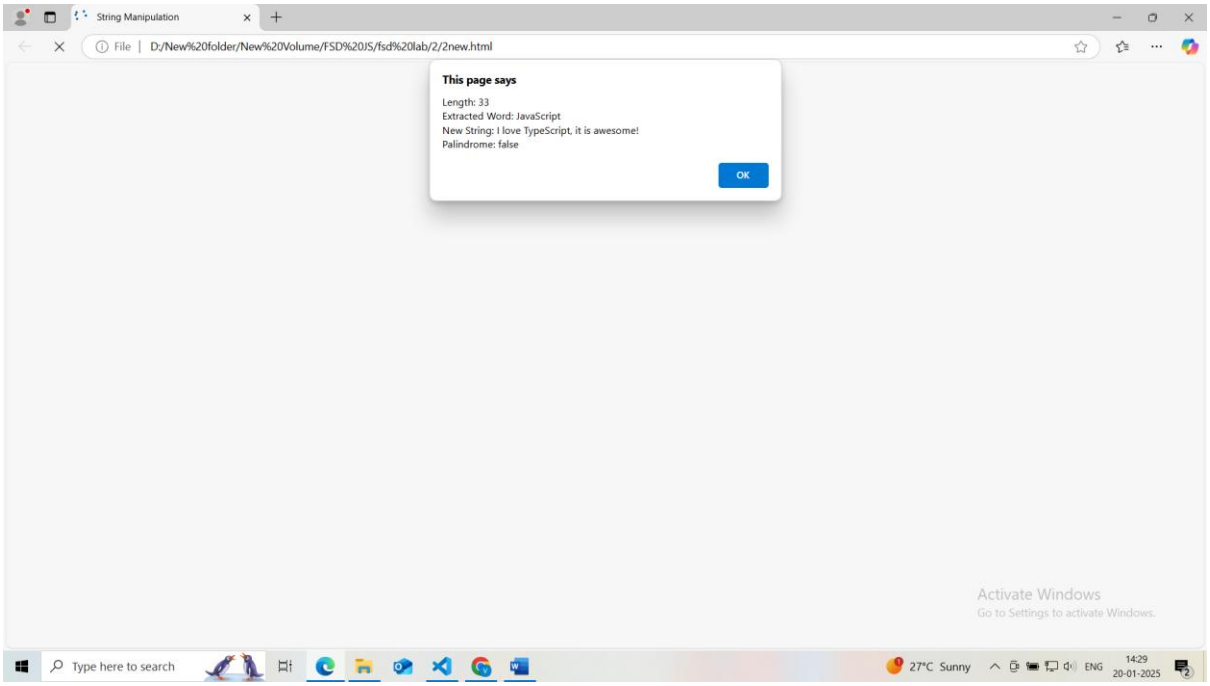
```
<!DOCTYPE html>
<head>

            <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>
```

o/p:

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




String Manipulation
File | D:/New%20folder/New%20Volume/FSD%20JS/fsd%20lab/2/2new.html
This page says
Length: 33
Extracted Word: JavaScript
New String: I love TypeScript, it is awesome!
Palindrome: false
OK
Activate Windows
Go to Settings to activate Windows.
Type here to search
27°C Sunny
ENG
14:29
20-01-2025

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("------------------------------");
            }
```

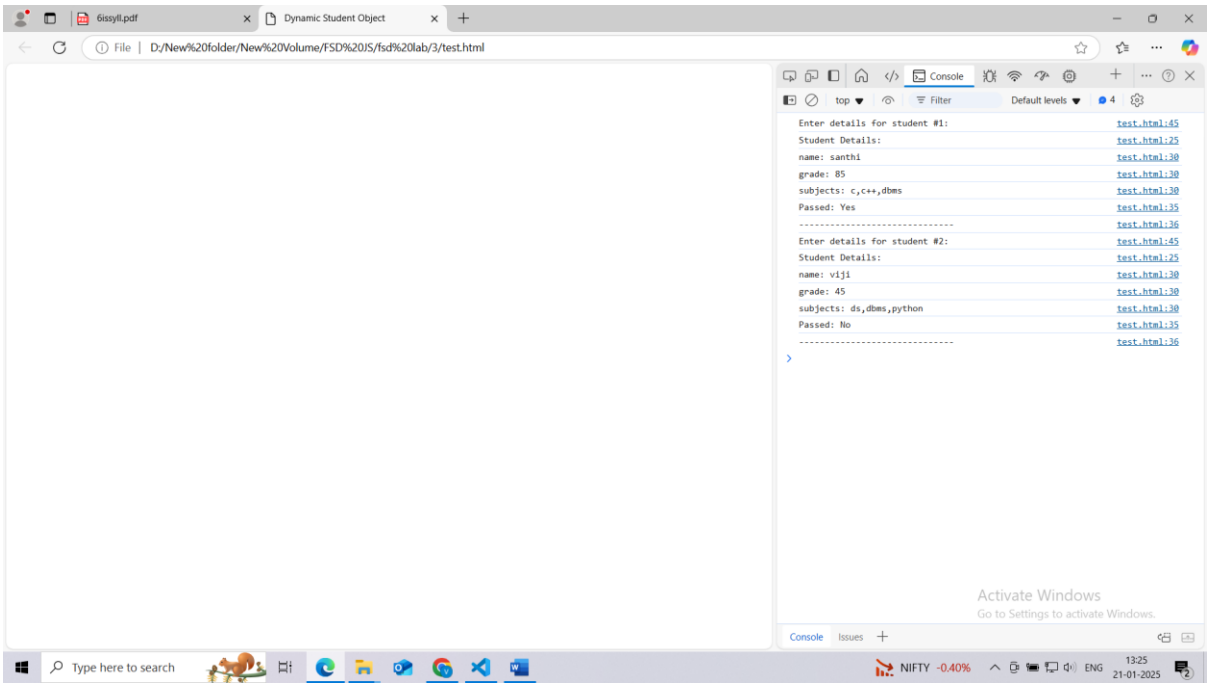
```
        let student = createStudent();  // Collect details and return
student object
        // Dynamically add 'passed' property based on grade
        student.passed = student.grade >= 40;
        displayInfo(student);
        </script>
</head>
</html>
```

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>
```

o/p:


6issyll.pdf
Dynamic Student Object
File | D:/New%20folder/New%20Volume/FSD%20JS/fsd%20lab/3/test.html
Console
top
Filter
Default levels
4
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
Activate Windows
Go to Settings to activate Windows.
Console
Issues
Type here to search
NIFTY -0.40%
ENG
13:25
21-01-2025

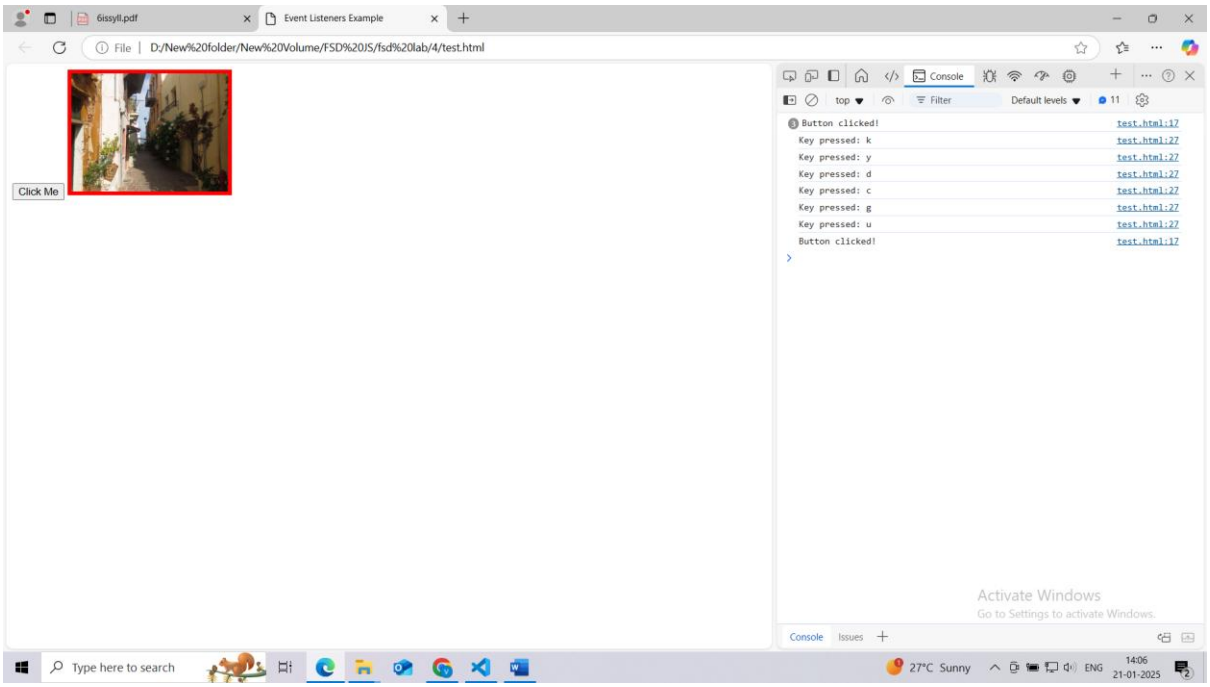
Experiment 4:

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('mouseove
r', 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('mouseout
', function() {
```

```
            this.style.border = 'none';  // remove the border
        });

    </script>

</body>
</html>
```

o/p:



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.

- **Initialize the Project**:

  Vite variations

  Vanilla

  Vue

  React

  Preact

  Lit

  Svelte

  Solid

  Qwik

  Angular

  Others

- npm create vite@latest issue-tracker --template react

  cd issue-tracker

- **Install Dependencies**:

  npm install

- **Start the Development Server**:

  npm run dev

- **Create the Functional Component**:

1. mkdir exp6
2. cd exp6
3. npm create vite@latest 5
4. cd 5
5. npm install
6. under src folder create Issuelist.jsx file

7. under src folder modify App.jsx file

IssueList.jsx

```
import React from 'react';

const issues = [
  { id: 1, title: 'Bug in Login', description: 'Error 404 on login',
status: 'Open' },
  { id: 2, title: 'UI Issue on Dashboard', description: 'Misaligned
elements', status: 'Closed' },
  { id: 3, title: 'API Timeout', description: 'Delayed response from
server', status: 'Open' },
];

function IssueList() {
  return (
    <div>
      <h1>Issue Tracker</h1>
      <ul> {/* Wrap list items inside an unordered list */}
        {issues.map(issue => (
          <li key={issue.id}> {/* Correctly assign 'key' inside <li> */}
            <h2>{issue.title}</h2> {/* Use curly braces to display the
title */}
            <p>{issue.description}</p> {/* Use curly braces to display
the description */}
            <p>Status: {issue.status}</p> {/* Use curly braces to display
the status */}
          </li>
        ))}
      </ul>
    </div>
  );
}
export default IssueList;
```

App.jsx

```
import React from 'react';
import IssueList from './IssueList';

function App() {
  return (
    <div>
      <IssueList />
    </div>
  );
```

```
}
export default App;
```

8. o/p: npm run dev

Run and View the Application:

- Start the development server if it's not already running:

npm run dev

- Visit http://localhost:3000 to view the issue tracker in the browser.

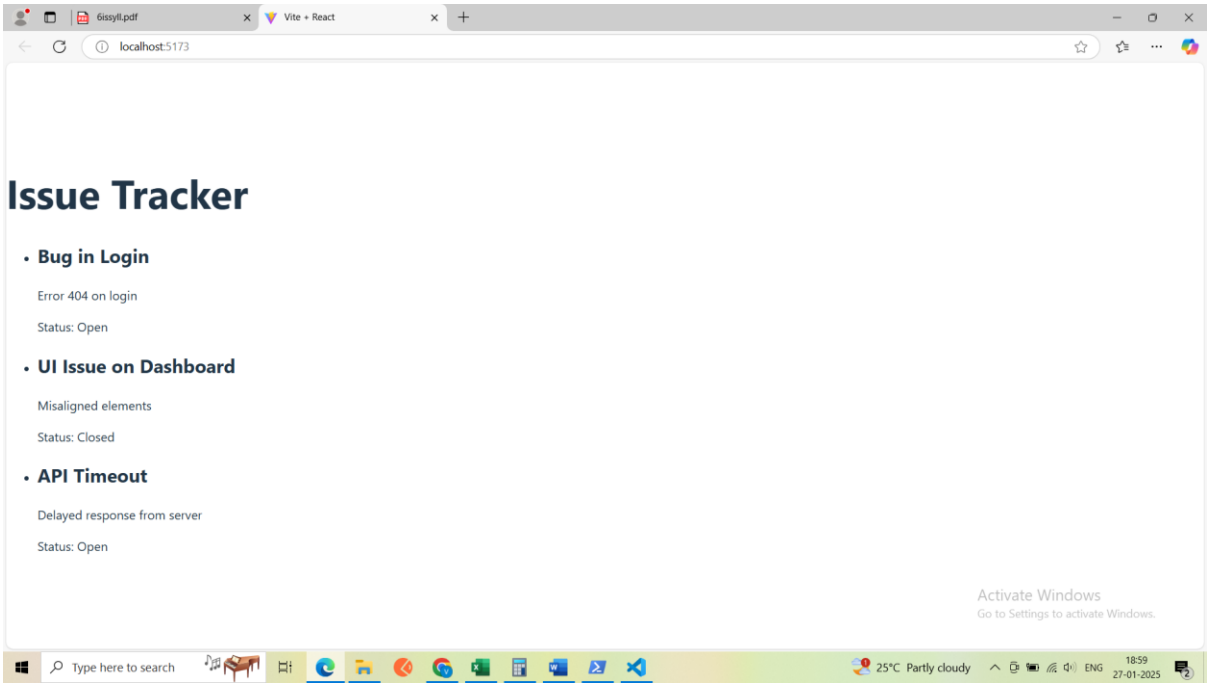
Or

VITE v6.0.11 ready in 1957 ms

➜ Local: http://localhost:5173/

➜ Network: use --host to expose

➜ press h + enter to show help



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 componentid count (class component). Extend the Counter component to Double the count value when a button is clicked. Reset the count to 0 using another button.

- Steps for reactjs:

- Old approach

==========

- First install GIT
- Ensure you are installing a compatible version of React, such as react@18. Run the following command to create the React app with a specific version:
- npx create-react-app counter-app

- **.Or using Vite**

  npm create vite@latest counter-app

  cd counter-app

  npm install

  npm run dev

- modify App.jsx In src folder

```
import React from 'react';
import Counter from './Counter';  // Ensure the correct import path
import './App.css';  // Optional, only if you have this file

function App() {
    return (
              <div className="App">
                   <h1>Welcome to Counter App</h1>
                <Counter />
                </div>
            );
}
export default App;
```

create src/Counter.jsx in src folder

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

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

    // Simulate fetching initial data for the Counter component
    useEffect(() => {
    setTimeout(() => {
        setCount(0);  // Set initial value after 2 seconds
     }, 2000);
     }, []);

     return (
            <div>
            <h1>Counter: {count}</h1>
            <button onClick={() => setCount(count +
1)}>Increment</button>
            <button onClick={() => setCount(count -
1)}>Decrement</button>
            <button onClick={() => setCount(count * 2)}>Double</button>
              <button onClick={() => setCount(0)}>Reset</button>
         </div>
         );
};

export default Counter;
```

steps for next js

1. npx create-next-app@latest counter-app

Options to Select:

1. **Would you like to use TypeScript? No** (Choose "Yes" if you want TypeScript, but if you prefer simplicity, select "No").
2. **Would you like to use ESLint? Yes** (Recommended for maintaining code quality, especially in team projects).
3. **Would you like to use Tailwind CSS? No** (Select "Yes" if you plan to style your app using Tailwind CSS, but it's unnecessary for this Counter Experiment).

4. **Would you like your code inside a src/ directory?**
   Yes (This organizes your files better, making the project structure clean and scalable).
5. **Would you like to use App Router? (recommended)**
   Yes (This uses the latest Next.js routing features introduced in version 13).

Would you like to use Turbopack for next dev?
Yes (Recommended for faster development builds; it's experimental but stable for many use cases).

6. **Would you like to customize the import alias (@/* by default)?**
   No (Stick with the default unless you have specific aliasing requirements).

TypeScript? No

ESLint? Yes

Tailwind CSS? No

Code inside `src/`? Yes

App Router? Yes

Turbopack for `next dev`? Yes

Import alias customization? No

Procedure

---

1. put Counter.jsx code In the folder like by creating Counter.js in src/app/Counter.js. and also add 'use client'; as first line in this file.
2. Moify the page.js content with src/App.Jsx by removing importing of App.jsx
3. create a folder like mkdir 6
4. cd 6
5. npm create vite@latest count
6. npm install
7. under src folder
8. **App.jsx**

```
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'
import Statemgtclass from './Statemgtclass'

function App() {


  return (

      <div>
```

```
        {/*<Statemgt/>*/}
        <Statemgtclass/>
        </div>

    )
}

export default App
```

using functional component

9. Statemgt.jsx

```
import React from 'react'
import { useState } from 'react'

const Statemgt = () => {
    const [number, setNumber] = useState(0)
    const incrememt = () => {
        setNumber(number + 1)
    }
    const decrememt = () => {
        if (number > 0)
            setNumber(number - 1)
    }
    const reset = () => {
        setNumber(0)
    }
    const double = () => {
        setNumber(number * 2)
    }

    return (
        <div><h1>number</h1>
            <h1>{number}</h1>
            <br></br>
            <button onClick={incrememt}>increment</button>
            <button onClick={decrememt}>decrement</button>
            <button onClick={double}>double</button>
            <button onClick={reset}>reset</button>
        </div>
    )
}

export default Statemgt
```

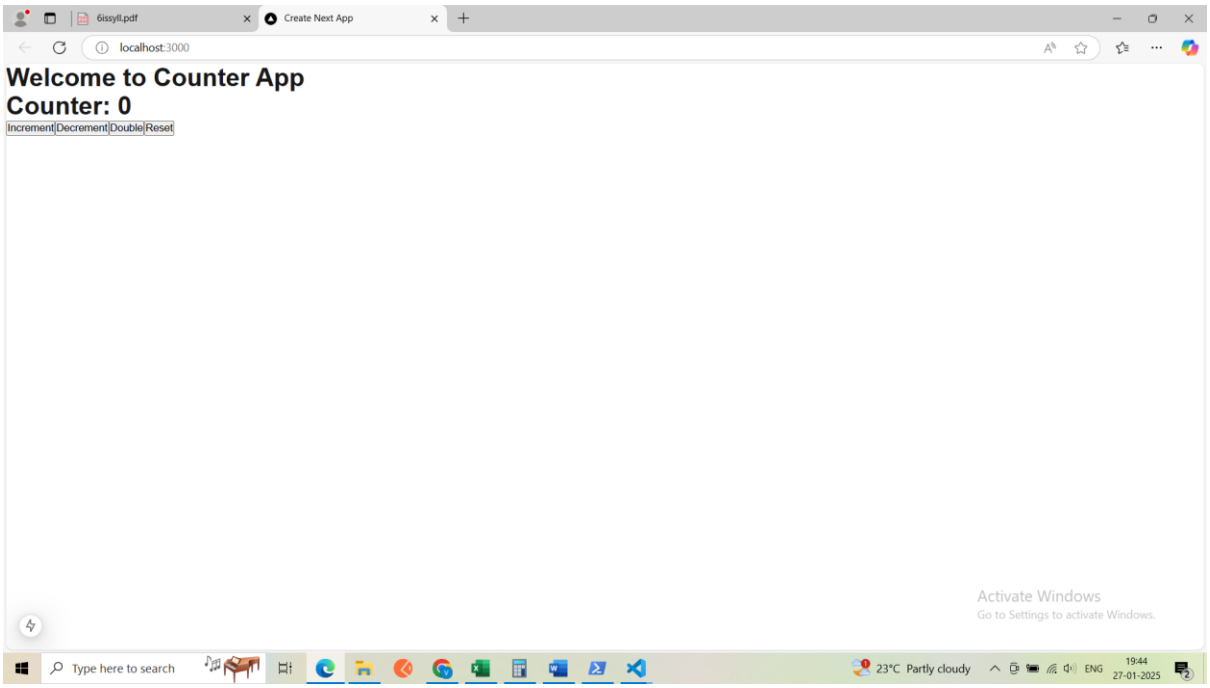
10. using class component

11. ==========

12. Statemgtclass.jsx

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

class Statemgtclass extends Component {
    state = { number: 0 };

    increment = () => {
        const value = this.state.number + 1;
        this.setState({ number: value });
    };

    decrement = () => {
        if (this.state.number > 0) { }
        const value = this.state.number - 1;
        this.setState({ number: value });
    };

    reset = () => {
        this.setState({ number: 0 });
    };

    double = () => {
        const value = this.state.number * 2;
        this.setState({ number: value });
    };

    render() {
        return (
            <div>
                <h1>Number</h1>
                <h1>{this.state.number}</h1>
                <br />
                <button onClick={this.increment}>Increment</button>
                <button onClick={this.decrement}>Decrement</button>
                <button onClick={this.double}>Double</button>
                <button onClick={this.reset}>Reset</button>
            </div>
        );
    }
}

export default Statemgtclass;
```

13. o/p: npm run dev



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-by-Step Process:

1. **Install Node.js and npm**
    - If you don't have Node.js installed, go to the official website: Node.js and install the LTS version. This will automatically install npm (Node Package Manager) as well.

1. create a folder expressjs
2. cd expressjs
3. npm init -y
4. npm install express or npm i express
5. npm i -D nodemon
6. now create a server.js file
7. under server.js file copy and paste the below code

server.js

```
import express, { json } from 'express'; // Use require instead of import
const app = express();
app.use(json()); // This is required to parse JSON data

app.get('/', (req, res) => {
    res.send('Hello, World!');
});
const products = [
    {
        id: 1,
        name: "mi"
    },
    {
        id: 2,
        name: "iphone"
    },
    {
```

```
        id: 3,
        name: "oppo"
    }
]
app.get('/products', (req, res) => {
    res.send(products);
});

app.get('/products/:id', (req, res) => {
    const newData = products.filter(item => item.id.toString() ===
req.params.id)
    res.send(newData);
});

app.post('/addproducts', (req, res) => {
    const { id, name } = req.body;
    const newProduct = { id, name };
    products.push(newProduct);
    res.send(newProduct);
});

app.put('/updateproducts/:id', (req, res) => {
    const product = products.find(item => item.id.toString() ===
req.params.id);
    Object.assign(product, req.body);
    res.send(product);
});

app.delete('/deleteproducts/:id', (req, res) => {
    const index = products.findIndex(item => item.id.toString() ===
req.params.id);
    const deletedProduct = products.splice(index, 1);
    res.send(deletedProduct);
});

app.listen(3000, () => {
    console.log('Server running on http://localhost:3000');
});
```

Under package.json ensure all the following things are there or not

```
{
  "name": "expressjs",
  "version": "1.0.0",
  "description": "",
  "license": "ISC",
  "author": "",
  "type": "module",
```

```
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js",
    "server": "nodemon server.js"
  },
  "dependencies": {
    "express": "^4.21.2"
  },
  "devDependencies": {
    "nodemon": "^3.1.9"
  }
}
```

o/p:

npm run server

next install the postman

Install Postman

1. Visit the Postman website and download the application for your operating system (Windows, Mac, or Linux).
2. Install Postman by following the prompts.

Next click on new there select http after selecting http here you can see the all apis like get, get/id, put/id, post, delete/id…etc

Get: http://localhost:3000/products

Post: http://localhost:3000/addproducts

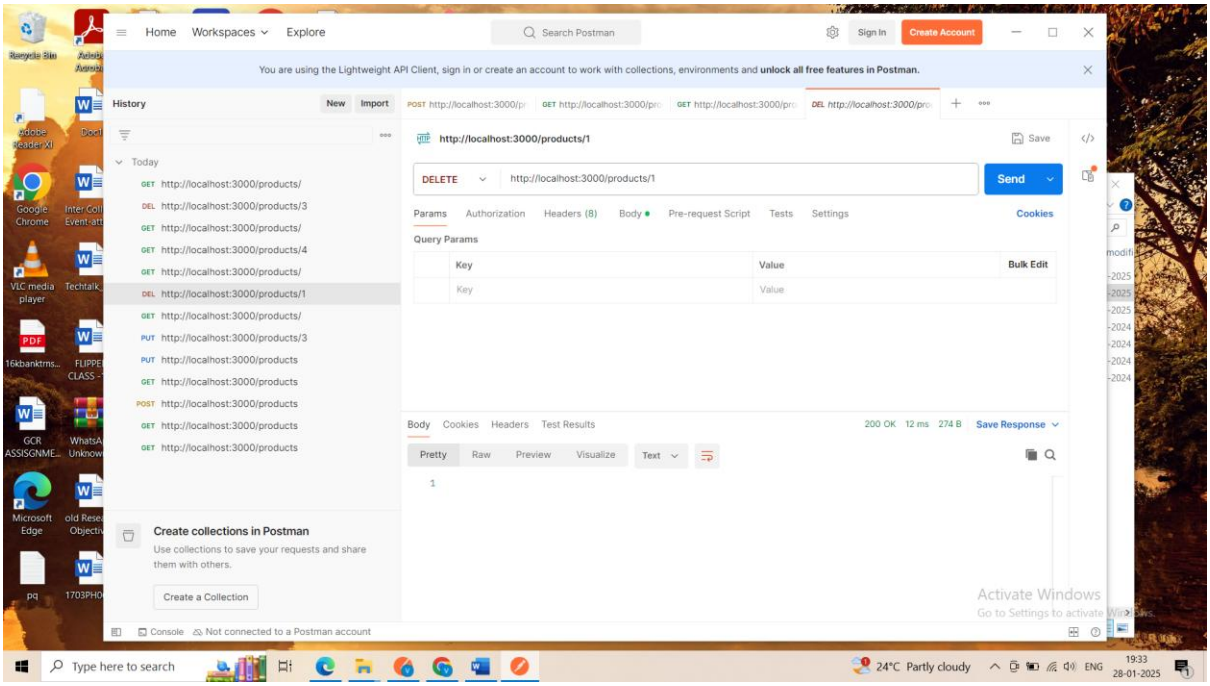
Put/id: http://localhost:3000/updateproducts/2

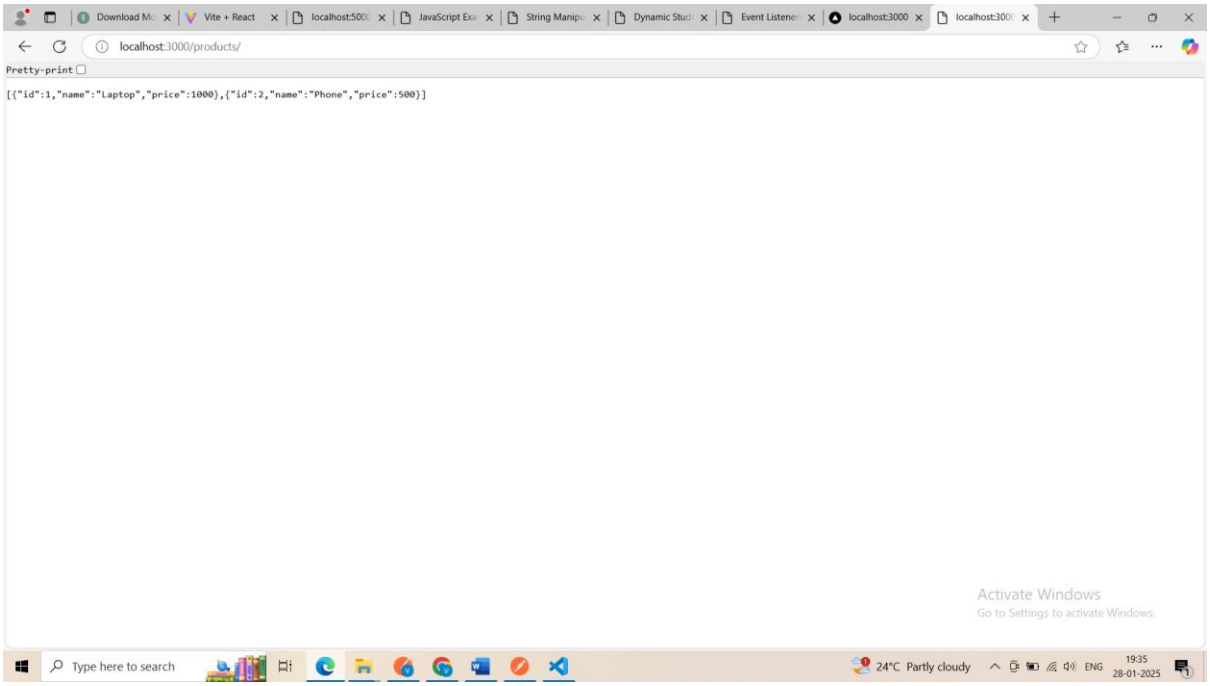
Get/id: http://localhost:3000/products/2

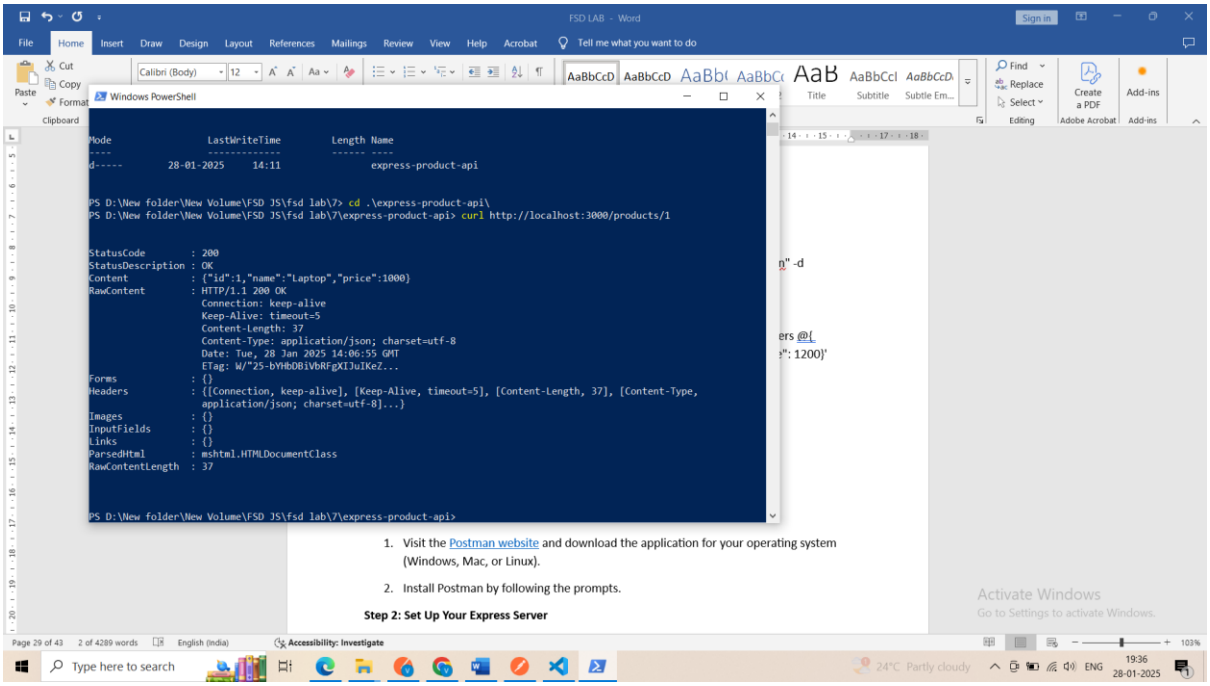
Delete: http://localhost:3000/deleteproducts/1

Json data Experiment

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






FSD LAB - Word
Windows PowerShell
Mode LastWriteTime Length Name
d----- 28-01-2025 14:11 express-product-api
PS D:\New folder\New Volume\FSD JS\fsd lab\7> cd .\express-product-api\
PS D:\New folder\New Volume\FSD JS\fsd lab\7\express-product-api> curl http://localhost:3000/products/1
StatusCode : 200
StatusDescription : OK
Content : {"id":1,"name":"Laptop","price":1000}
RawContent : HTTP/1.1 200 OK
Connection: keep-alive
Keep-Alive: timeout=5
Content-Length: 37
Content-Type: application/json; charset=utf-8
Date: Tue, 28 Jan 2025 14:06:55 GMT
ETag: W/"25-bYHbDBiVbRFgXIJuIKeZ...
Forms : {}
Headers : {[Connection, keep-alive], [Keep-Alive, timeout=5], [Content-Length, 37], [Content-Type, application/json; charset=utf-8]...}
Images : {}
InputFields : {}
Links : {}
ParsedHtml : mshtml.HTMLDocumentClass
RawContentLength : 37
PS D:\New folder\New Volume\FSD JS\fsd lab\7\express-product-api>
1. Visit the Postman website and download the application for your operating system (Windows, Mac, or Linux).
2. Install Postman by following the prompts.
Step 2: Set Up Your Express Server
Activate Windows
Go to Settings to activate Windows.
Type here to search
24°C Partly cloudy
ENG
19:36
28-01-2025

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: Setup Your Environment

1. **Install Node.js (please refer to previous experiment)**
2. **Initialize a Node.js Project**

   mkdir experiment8; cd experiment8;

   mkdir server; cd server

   npm init -y
3. Edit package.json

   add “type”: “module”
4. **Install Required Dependencies**

   npm install cors express mongoose

Step 2: Set Up MongoDB

1. **Install MongoDB and MongoDB Compass**

Step 3: Create following files under experiment8/server folder

products.js

```
import { Router } from 'express';
const router = Router();
import Product from './database.js';

// API Endpoints
// Get all the products
router.get('/', async (req, res) => {
    try {
        const products = await Product.find();
        res.json(products);
    } catch (error) {
        res.status(500).send(error.message);
```

```
    }
});

// Add a new product
router.post('/', async (req, res) => {
    try {
        const newProduct = new Product(req.body);
        await newProduct.save();
        res.status(201).json(newProduct);
    } catch (error) {
        res.status(500).send(error.message);
    }
});

// Update a product
router.put('/:id', async (req, res) => {
    try {
        const updatedProduct = await
Product.findByIdAndUpdate(req.params.id, req.body, {
            new: true,
        });
        res.json(updatedProduct);
    } catch (error) {
        res.status(500).send(error.message);
    }
});

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

export default router;
```

database.js

```
import mongoose from 'mongoose';

const DB_URL = 'MongoDB://localhost:27017/shop';

// MongoDB connection
mongoose.connect(DB_URL);

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

// Product schema using Mongoose
const productSchema = new mongoose.Schema({
  name: String,
  price: Number,
  description: String,
});

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

export default Product;
```

server.js

```
import mongoose from 'mongoose';

const DB_URL = 'MongoDB://localhost:27017/shop';

// MongoDB connection
mongoose.connect(DB_URL);

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

// Product schema using Mongoose
const productSchema = new mongoose.Schema({
  name: String,
```

```
  price: Number,
  description: String,
});

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

export default Product;
```

Step 4: Create a React Frontend

1. **Go back to** `experiment8` **folder**
2. **Run**

```
npm create vite@latest client --- --template react
```

Step 5: Create/modify following files under `experiment/client/src` **as below:**

ProductList.jsx

```
const ProductList = ({ products }) => {
return (
<>
    <h1>Product List</h1>
    <table>
        <thead>
            <tr>
                <th>Name</th>
                <th>Description</th>
                <th>Price</th>
            </tr>
        </thead>
        <tbody>
            {products.map((product) => (
            <tr key={product._id}>
                <td>{product.name}</td>
                <td>{product.description}</td>
                <td>${product.price}</td>
            </tr>
            ))}
        </tbody>
    </table>
</>
);
};

export default ProductList;
```

AddProduct.jsx

```
import React, { useState } from "react";

const AddProduct = ({ fetchProducts }) => {
    const [newProduct, setNewProduct] = useState({
        name: "",
        price: "",
        description: "",
    });

    async function handleAddProduct() {
        try {
            let response = await
                fetch("http://localhost:5000/products", {
                    method: "POST",
                    headers: {
                        "Content-Type": "application/json",
                    },
                    body: JSON.stringify(newProduct),
                });
            if (!response.ok) {
                throw new Error(`HTTP status = ${response.status}`);
            }
            setNewProduct({ name: "", price: "", description: "" });
            fetchProducts(); // Refresh the product list
        } catch (error) {
            console.error("Error adding product:", error);
        }
    }

    return (
        <>
            <h2>Add New Product</h2>
            <div>
                <input
                    type="text"
                    placeholder="Name"
                    value={newProduct.name}
                    onChange={(e) =>
                        setNewProduct({
                            ...newProduct,
                            name: e.target.value
                        })
                    }
```

```
                />
                <input
                    type="text"
                    placeholder="Description"
                    value={newProduct.description}
                    onChange={(e) =>
                        setNewProduct({
                            ...newProduct,
                            description: e.target.value
                        })
                    }
                />
                <input
                    type="number"
                    placeholder="Price"
                    value={newProduct.price}
                    onChange={(e) =>
                        setNewProduct({
                            ...newProduct,
                            price: e.target.value
                        })
                    }
                />
                <button onClick={handleAddProduct}>Add Product</button>
            </div>
        </>
    );
};

export default AddProduct;
```

App.jsx (note: replace the whole content)

```
import React, { useState, useEffect } from "react";
import ProductList from "./ProductList";
import AddProduct from "./AddProduct";

const App = () => {
    const [products, setProducts] = useState([]);

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

    async function fetchProducts() {
        const response = await
            fetch(`http://localhost:5000/products`);
        if (!response.ok) {
            const message = `Error while fetching the products:
${response.statusText}`;
            console.error(message);
            return;
        }
        const products = await response.json();
        setProducts(products);
    }

    return (
        <div style={{ padding: "20px" }}>
            <ProductList products={products} />
            <AddProduct fetchProducts={fetchProducts} />
        </div>
    );
};

export default App;
index.css(note: add following styles to index.css)
table, th, td {
    border: 1px solid black;
    border - collapse: collapse;
    padding - left: 5px;
    padding - right: 5px;
}

thead {
```

```
    background - color: lightgray;
}
```

App.css (note: remove the file)

Step 6: Run the Application

1. **Start the Backend Server**

```
cd experiment/server; node server.js
```

2. **Start the React Frontend**

```
cd experiment/client
npm run dev
```

3. **Access the Application**
   - Backend API: http://localhost:5000/products
   - React App: http://localhost:5173

Outcome

1. The React app displays a list of products fetched from the /products endpoint.

   Using Postman or Browser

   If your Node.js API is running, you can fetch the products using your GET endpoint:

   - Open **Postman** or your browser.

   http://localhost:5000/products

2. Adding a new product updates the list dynamically without refreshing the page.

You now have a fully functional MongoDB-Node.js-Express-React app.

How to check which data are there in MongoDB

1. Using MongoDB Shell

1. **Start the MongoDB Shell**:
   - Run mongosh (for modern versions) or mongo (for older versions) in your terminal.
2. **Switch to the shop Database**:

use shop

3. **List All Collections**:

show collections

You should see products.

4. **View All Documents in the products Collection**:

db.products.find().pretty()

This will display all the products in a readable format.

---

2. Using MongoDB Compass

1. **Open MongoDB Compass**:
    - Download and install MongoDB Compass if you don't already have it.
2. **Connect to Your MongoDB Server**:
    - Use the connection string MongoDB://localhost:27017 (default for local installations).
3. **Select the shop Database**:
    - In the left-hand menu, select the shop database.
4. **View the products Collection**:
    - Click on the products collection, and all the documents (data) will be displayed in a tabular format.

Sample Screen: