Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)

(**NAAC Accredited & ISO 9001:2015 Certified Institution)**

NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

LAB MANUAL

(2025-26)

BIS601 - FULLSTACK DEVELOMENT

VI Semester ISE

Name: ____

USN: ____

Batch: ___________________ Section: ______________

FULL STACK DEVELOPMENT lab

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.

Program 1:

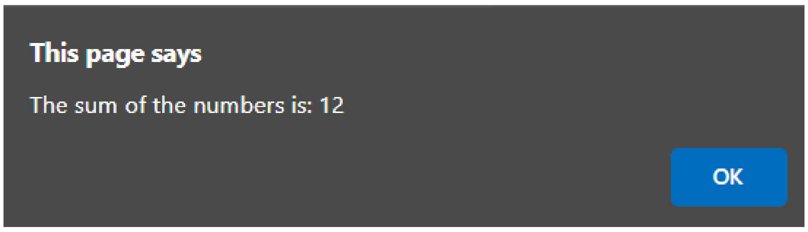
a. Write a script that Logs "Hello, World!" to the console. Create a script that calculates the sum of two numbers and displays the result in an alert box.

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.

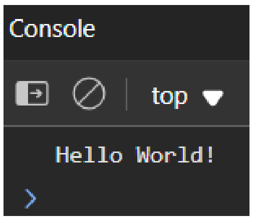
a. Write a script that Logs "Hello, World!" to the console. Create a script that calculates the sum of two numbers and displays the result in an alert box.

```
 <script>
     console.log("Hello World!")
     let number1 = 5;
     let number2 = 7;
     let sum = number1 + number2;
     alert("The sum of the numbers is: " + sum);
     </script>
```

Output:



```
This page says
The sum of the numbers is: 12
```

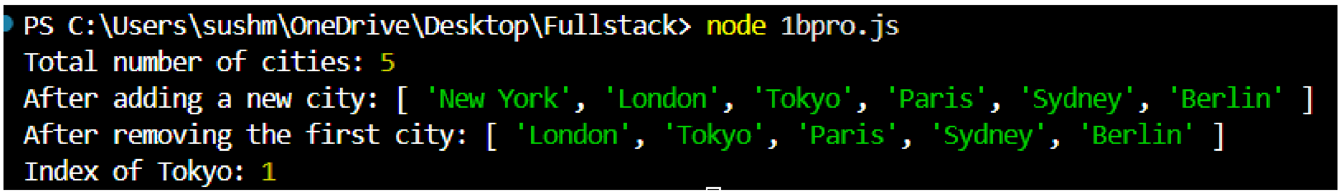


```
Hello World!
```

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.

```
let cities = ["New York", "London", "Tokyo", "Paris",
"Sydney"];
console.log("Total number of cities:", cities.length);
cities.push("Berlin");
console.log("After adding a new city:", cities);
cities.shift();
console.log("After removing the first city:", cities);
let cityToFind = "Tokyo";
let index = cities.indexOf(cityToFind);
console.log(`Index of ${cityToFind}:`, index);
```

Output:



```
PS C:\Users\sushm\OneDrive\Desktop\Fullstack> node 1bpro.js
Total number of cities: 5
After adding a new city: [ 'New York', 'London', 'Tokyo', 'Paris', 'Sydney', 'Berlin' ]
After removing the first city: [ 'London', 'Tokyo', 'Paris', 'Sydney', 'Berlin' ]
Index of Tokyo: 1
```

Program 2

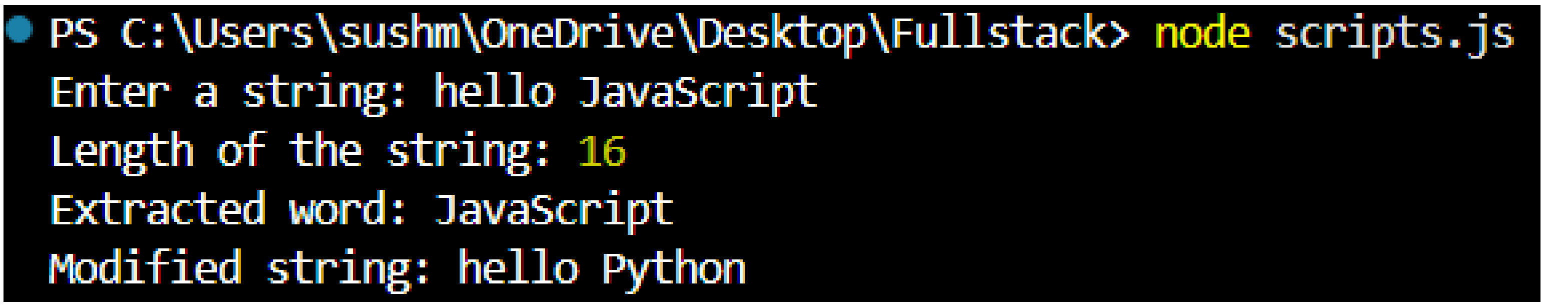
a. Read a string from the user, Find its length. Extract the word "JavaScript" using substring() or slice(). Replace one word with another word and log the new string.

b. Write a function isPalindrome(str) that checks if a given string is a palindrome (reads the same backward).

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.

```
const readline = require("readline");
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});
rl.question("Enter a string: ", function (inputString)
{
    console.log("Length of the string:",
inputString.length);
    let extractedWord =
inputString.includes("JavaScript")
    ?
inputString.substring(inputString.indexOf("JavaScript
"), inputString.indexOf("JavaScript") + 10)
    : "JavaScript not found";
    console.log("Extracted word:", extractedWord);
    let newString = inputString.replace("JavaScript",
"Python");
    console.log("Modified string:", newString);
    rl.close();
});
```

Output:

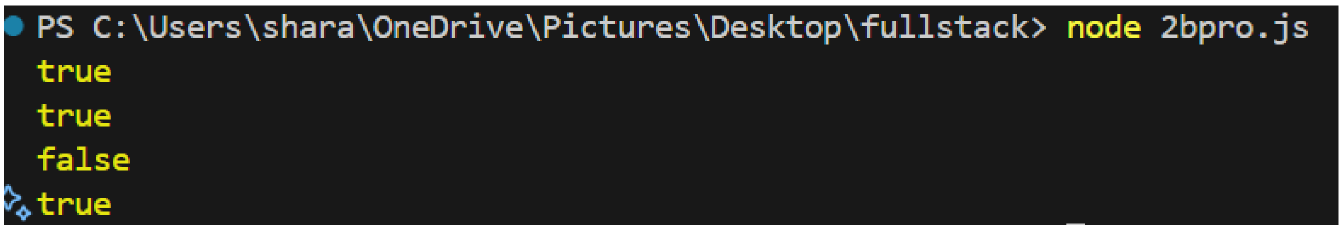


```
PS C:\Users\sushm\OneDrive\Desktop\Fullstack> node scripts.js
Enter a string: hello JavaScript
Length of the string: 16
Extracted word: JavaScript
Modified string: hello Python
```

b. Write a function isPalindrome(str) that checks if a given string is a palindrome (reads the same backward).

```
function isPalindrome(str) {
    const cleanedStr = str.toLowerCase().replace(/[^a-
z0-9]/g, '');
    return cleanedStr ===
cleanedStr.split('').reverse().join('');
}
console.log(isPalindrome("madam"));         // true
console.log(isPalindrome("racecar"));       // true
console.log(isPalindrome("hello"));         // false
console.log(isPalindrome("A man, a plan, a canal:
Panama")); // true
```

Output:



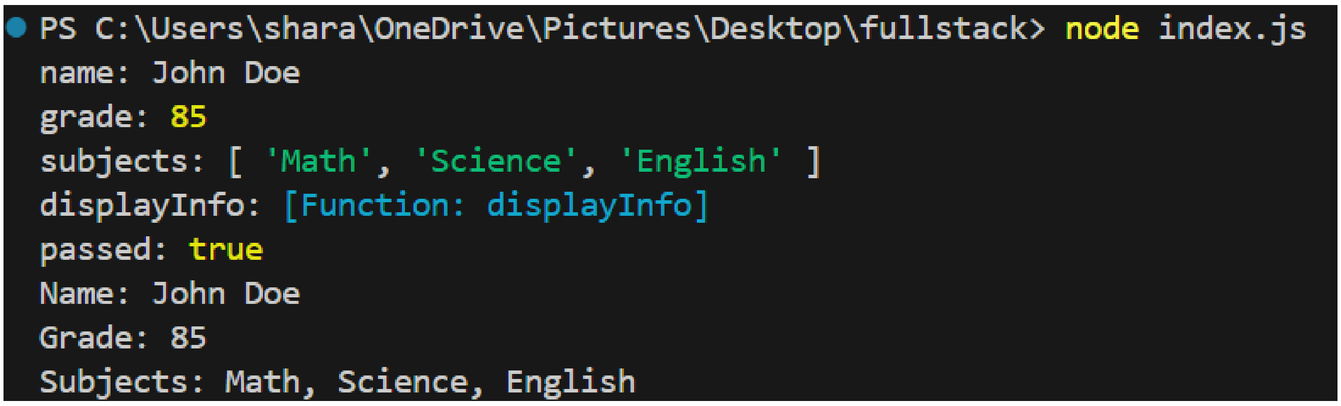
```
PS C:\Users\shara\OneDrive\Pictures\Desktop\fullstack> node 2bpro.js
true
true
false
true
```

Program 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.

```
const student = {
    name: "John Doe",
    grade: 85,
    subjects: ["Math", "Science", "English"],
    displayInfo() {
        console.log(`Name: ${this.name}`);
        console.log(`Grade: ${this.grade}`);
        console.log(`Subjects:
${this.subjects.join(",")}`);
    }
};
student.passed = student.grade ≥ 50;
for (let key in student) {
    console.log(`${key}:`, student[key]);
}
student.displayInfo();
```

Output:



```
PS C:\Users\shara\OneDrive\Pictures\Desktop\fullstack> node index.js
name: John Doe
grade: 85
subjects: [ 'Math', 'Science', 'English' ]
displayInfo: [Function: displayInfo]
passed: true
Name: John Doe
Grade: 85
Subjects: Math, Science, English
```

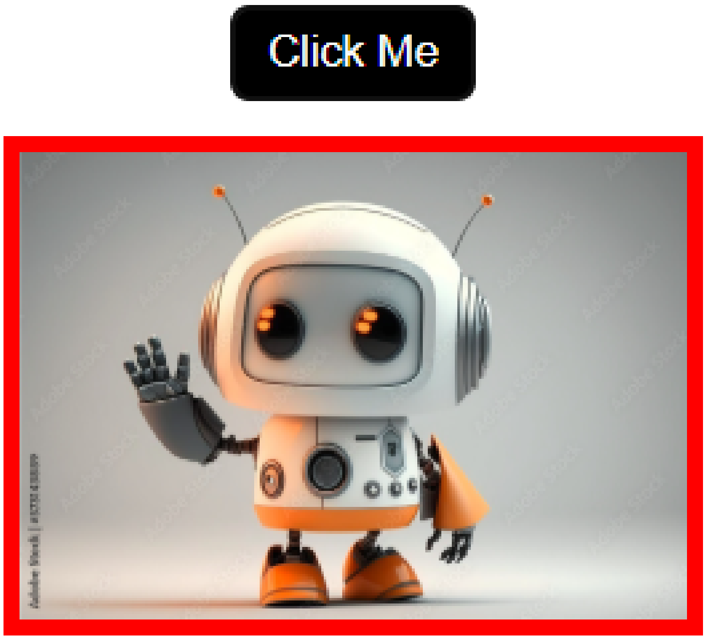
Program 4:

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.

```
c<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Event Listeners Example</title>
  <style>
    body { display: flex; flex-direction: column;
align-items: center; justify-content: center; height:
100vh; margin: 0; }
    img { width: 200px; height: 140px; border: 5px
solid black; margin-top: 10px; }
    button { padding: 5px 10px; background: #000;
color: #fff; border-radius: 5px; cursor: pointer; }
  </style>
</head>
<body>
  <button id="myButton">Click Me</button>
  <img id="myImage" src="https://as1.ftcdn.net/v2/
jpg/05/73/14/38/1000_F_573143889_NVvKlj8AGINKQyT7Pr3tkv
CScXShff0F.jpg">
  <script>
    const img = document.getElementById('myImage');
    document.getElementById('myButton').onclick = () =>
console.log('Button clicked!');
```

```
    img.onmouseover = () => img.style.borderColor =
'red';
    document.onkeydown = (e) => console.log('Key
pressed: ' + e.key);
  </script>
</body>
</html>
```

Output:



Program 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 getting started, make sure you have Node.js and npm (Node Package Manager) installed on your system.

- Node.js is a runtime environment that allows you to run JavaScript outside of a browser.
- npm is a package manager that comes with Node.js and is used to install and manage JavaScript libraries and dependencies.

Installation Instructions

1.Download Node.js from the official website: Node.js Downloads
2.Choose the LTS (Long Term Support) version for better stability.
3.Follow the installation wizard. Ensure that you check the option to install npm alongside Node.js.
4.After installation, open your terminal or command prompt and run the following commands to verify the installation:

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

Step 2: Create a Project Folder

1.Create a new directory for your project using the terminal:

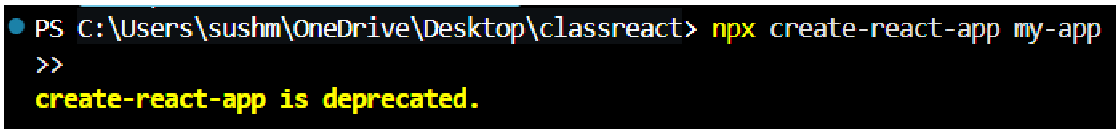
```
mkdir issue-tracker
cd issue-tracker
```

2.Open this folder in Visual Studio Code (VS Code) or your preferred code editor.

Step 3: Initialize a React Application

1.To create a React application, use npx (Node Package eXecute).

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



```
PS C:\Users\sushm\OneDrive\Desktop\classreact> npx create-react-app my-app
>>
create-react-app is deprecated.
```

Navigate to the Project Folder:

```
cd issue-tracker
```

Step 4: Modify the App Component

Now that your project is set up, we'll customize the main App.js component to display issues using static data.

Follow These Steps:

1.Go to the src folder inside your project directory.
2.Open **App.js** and replace its contents with the following code:

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

const issues = [
  { id: 1, title: "Bug in login page", description:
"Error on invalid credentials.", status: "Open" },
  { id: 2, title: "UI glitch on homepage", description:
"Misalignment on small screens.", status: "Closed" },
  { id: 3, title: "Missing translation for settings
page", description: "No Spanish translations.", status:
"Open" },
  { id: 4, title: "Database connection error",
description: "Intermittent issue during peak hours.",
status: "Open" },
];

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

const App = () => (
  <div className="App">
    <h1>Issue Tracker</h1>
    <div className="issue-list">
      {issues.map((issue) => <Issue key={issue.id}
{...issue} />)}
    </div>
  </div>
);

export default App;
```

Step 5: Style the Application

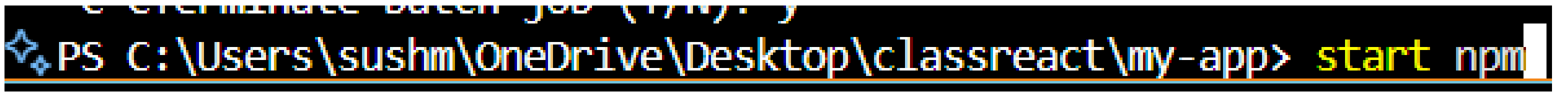
To make the app visually appealing, update the App.css file with the following CSS:

App.css:

```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: #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-bottom: 10px;
  font-size: 1.5em;
}
.issue p {
  margin-bottom: 10px;
  color: #555;
}
.status {
  font-weight: bold;
}
.status.open { color: #e73ce7; }
.status.closed { color: #362ecc; }
```

Step 6: Start the Development Server

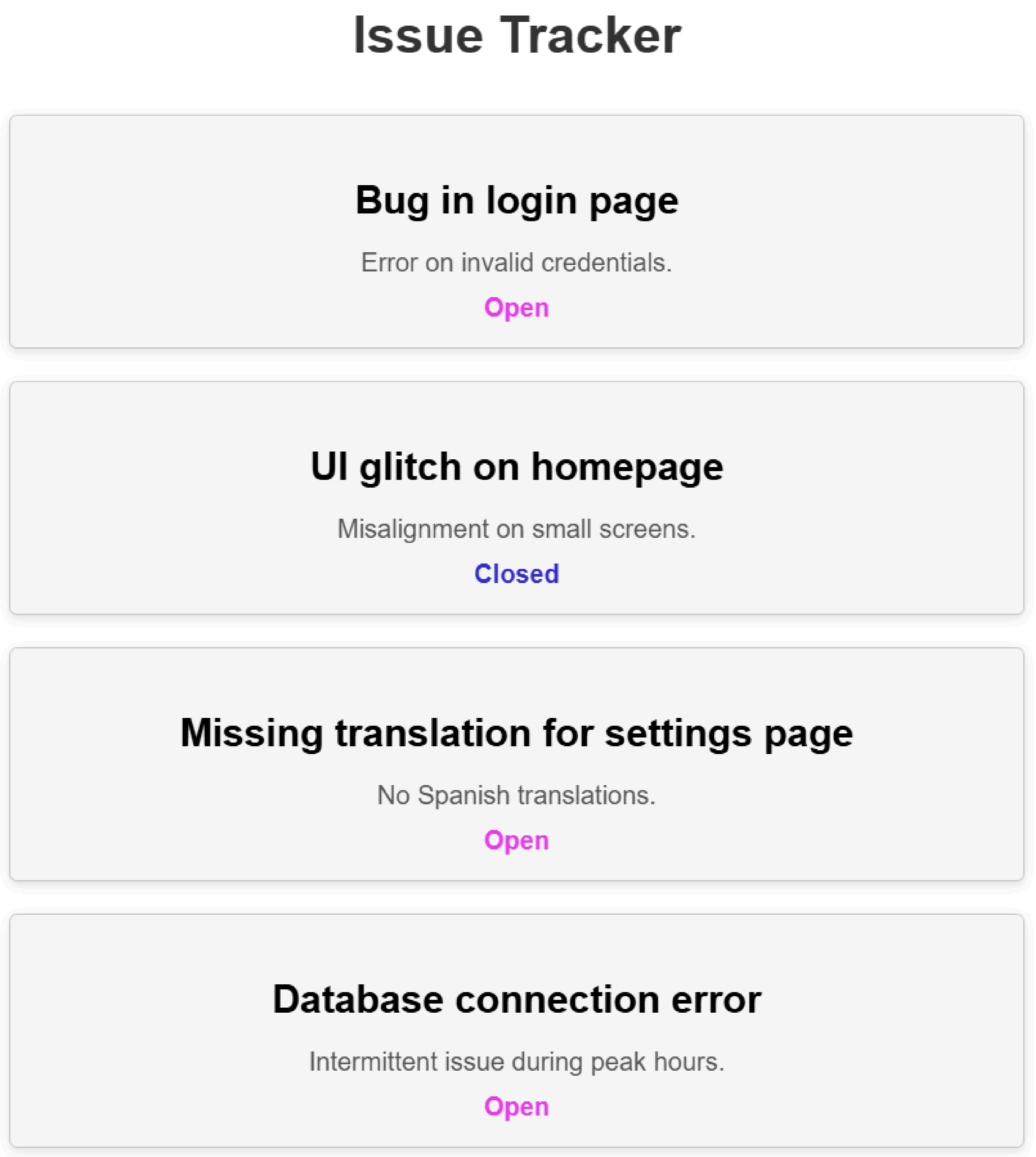
Once everything is set up, you can start the application using the following command:



```
PS C:\Users\sushm\OneDrive\Desktop\classreact\my-app> start npm
```

This will launch a local development server.
By default, your app will be available at http://localhost:3000.
Open your browser and visit that URL to see the issue tracker in action.

Output:



```
Issue Tracker

Bug in login page
Error on invalid credentials.
Open

UI glitch on homepage
Misalignment on small screens.
Closed

Missing translation for settings page
No Spanish translations.
Open

Database connection error
Intermittent issue during peak hours.
Open
```

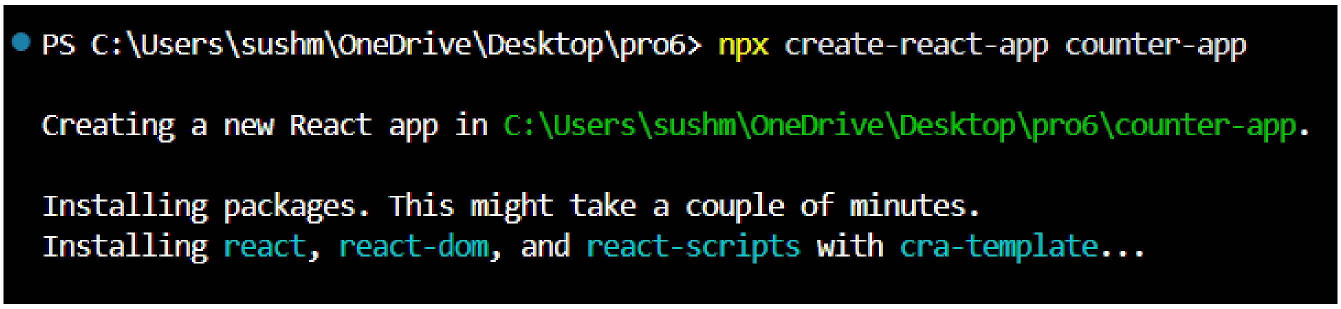
Program 6:

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: Set Up the Project

1.Create a folder and open it in **Visual Studio Code** (VSCode).
2.Open the integrated terminal using **Ctrl +** ` or from the **Terminal menu**.
3.Run the following command to create a **React app**:

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



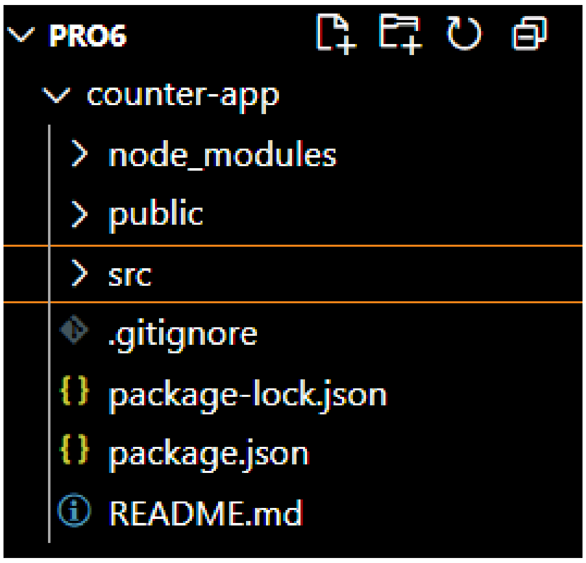
```
PS C:\Users\sushm\OneDrive\Desktop\pro6> npx create-react-app counter-app

Creating a new React app in C:\Users\sushm\OneDrive\Desktop\pro6\counter-app.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...
```

4.After the installation is complete, navigate to your project directory:

```
cd counter-app
```

```
PS C:\Users\sushm\OneDrive\Desktop\pro6> cd counter-app
```

After completing these steps, your project structure will look like this:



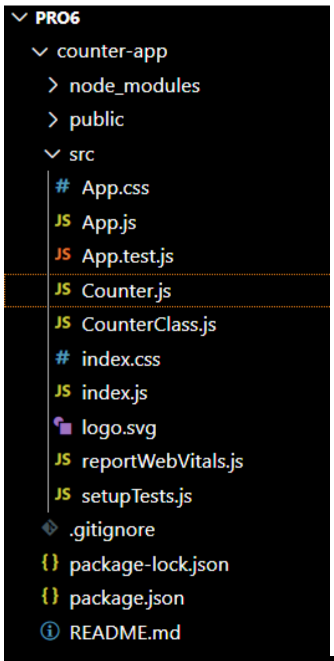
```
PRO6
  counter-app
    > node_modules
    > public
    > src
    .gitignore
    package-lock.json
    package.json
    README.md
```

Step 2: Create and Organize Files

1.Inside the **src** folder, create two files:
- Counter.js
- CounterClass.js

2.Copy the respective code into these files.
3.Modify the App.js file by copying the provided code into it.
4.Similarly, update the App.css file with the given styles.

After completing these steps, your project structure will look like this:



```
PRO6
  counter-app
    node_modules
    public
    src
      App.css
      App.js
      App.test.js
      Counter.js
      CounterClass.js
      index.css
      index.js
      logo.svg
      reportWebVitals.js
      setupTests.js
    .gitignore
    package-lock.json
    package.json
    README.md
```

Counter.js

```
import React, { useState, useEffect } from 'react';
const Counter = () => {
  const [count, setCount] = useState(0);
  useEffect(() => console.log('Fetching initial
data...'), []);
  const actions = [
    { label: 'Increment', operation: (prev) => prev + 1
},
    { label: 'Decrement', operation: (prev) => prev - 1
},
    { label: 'Double', operation: (prev) => prev * 2 },
    { label: 'Reset', operation: () => 0 },
  ];
  return (
    <div>
      <h1>Count: {count}</h1>
      {actions.map(({ label, operation }) => (
        <button key={label} onClick={() =>
setCount(operation)}>{label}</button>
      ))}
    </div>
  );
};
export default Counter;
```

JS CounterClass.js

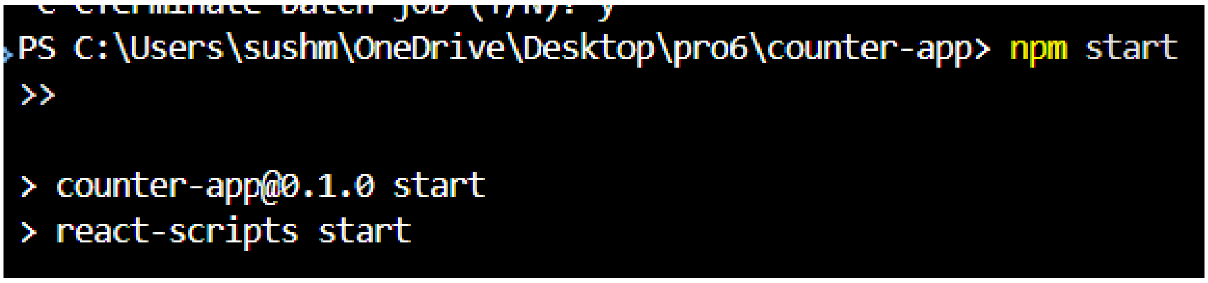
```
import React, { Component } from 'react';
class Counter extends Component {
  state = { count: 0 };
  componentDidMount() {
    console.log('Fetching initial data...');
  }
  handleCount = (action) => {
    this.setState((prevState) => ({
      count:
        action === 'increment' ? prevState.count + 1 :
        action === 'decrement' ? prevState.count - 1 :
        action === 'double' ? prevState.count * 2 : 0
    }));
  };
  render() {
    return (
      <div>
        <h1>Count: {this.state.count}</h1>
        {['Increment', 'Decrement', 'Double', 'Reset'].map(action
=> (
          <button key={action} onClick={() =>
this.handleCount(action.toLowerCase())}>{action}</button>
        ))}
      </div>
    );
  }
}
export default Counter;
```

JS App.js

```
import React from 'react';
import './App.css';
import Counter from './CounterClass';
const App = () => (
  <div className="App">
    <h1 className='head'>Welcome to the Counter App</
h1>
    <Counter />
  </div>
);
export default App;
```

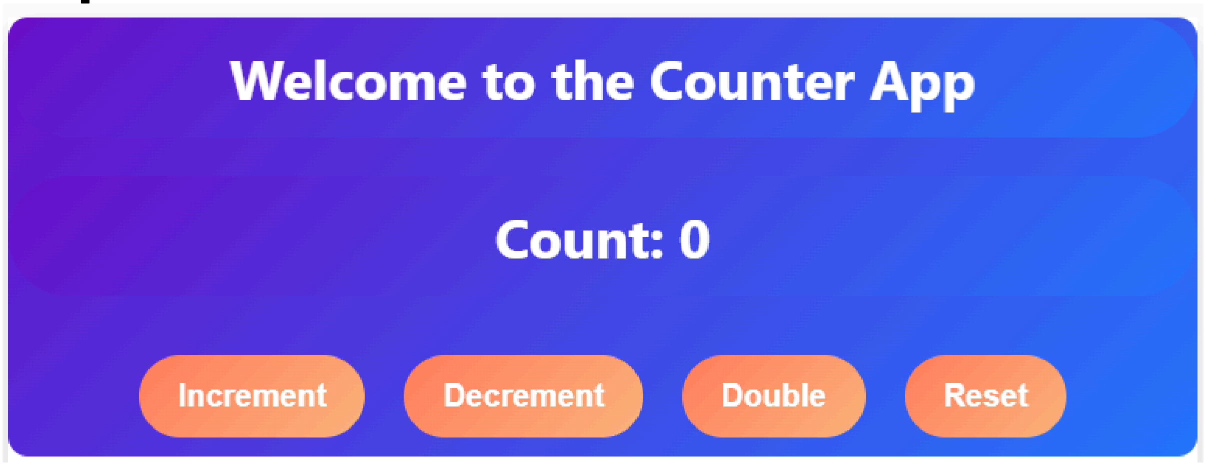
App.css

```css
.App, h1, .head {
  border-radius: 10px;
  background: linear-gradient(135deg, #6a11cb,
#2575fc);
  color: #fff;
  text-align: center;
}

#root {
  margin: 20px auto;
  width: 40%;
  height: 400px;
  box-shadow: 0 8px 20px rgba(0, 0, 0, 0.15);
}

h1, .head {
  font-size: 28px;
  padding: 12px 60px;
  margin: 20px auto;
  border-radius: 50px;
}

.head {
  width: 100%;
  padding: 12px 0;
}

button {
  margin: 10px;
  padding: 12px 20px;
  font-size: 16px;
  font-weight: 600;
  border: none;
  border-radius: 25px;
  cursor: pointer;
  background: linear-gradient(135deg, #ff7e5f,
#feb47b);
color: #fff;
  transition: transform 0.2s, background 0.3s;
}

button:hover {
  transform: translateY(-3px);
  background: linear-gradient(135deg, #feb47b,
#ff7e5f);
}
```



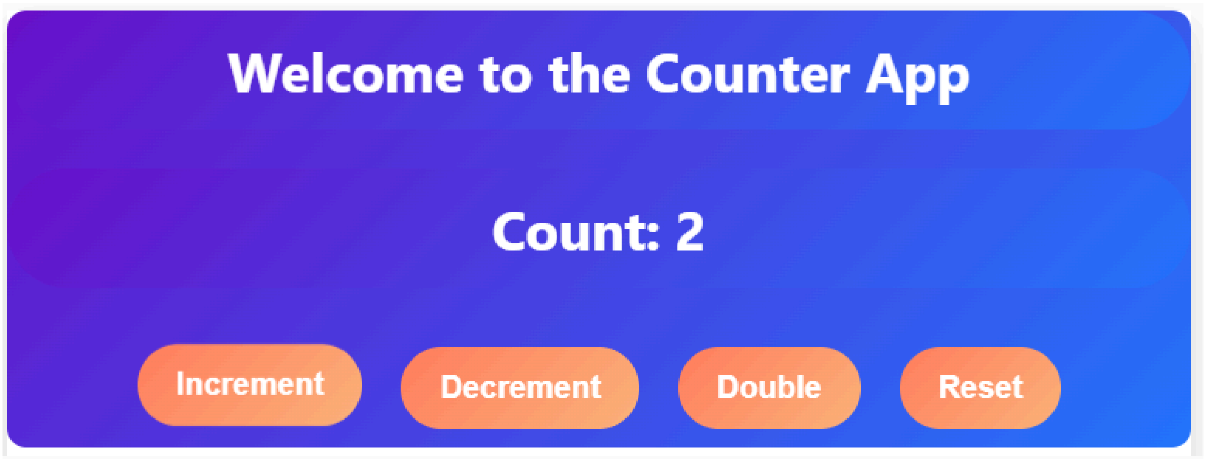
```
^C Terminate batch job (Y/N)? y
PS C:\Users\sushm\OneDrive\Desktop\pro6\counter-app> npm start
>>

> counter-app@0.1.0 start
> react-scripts start
```

Output:



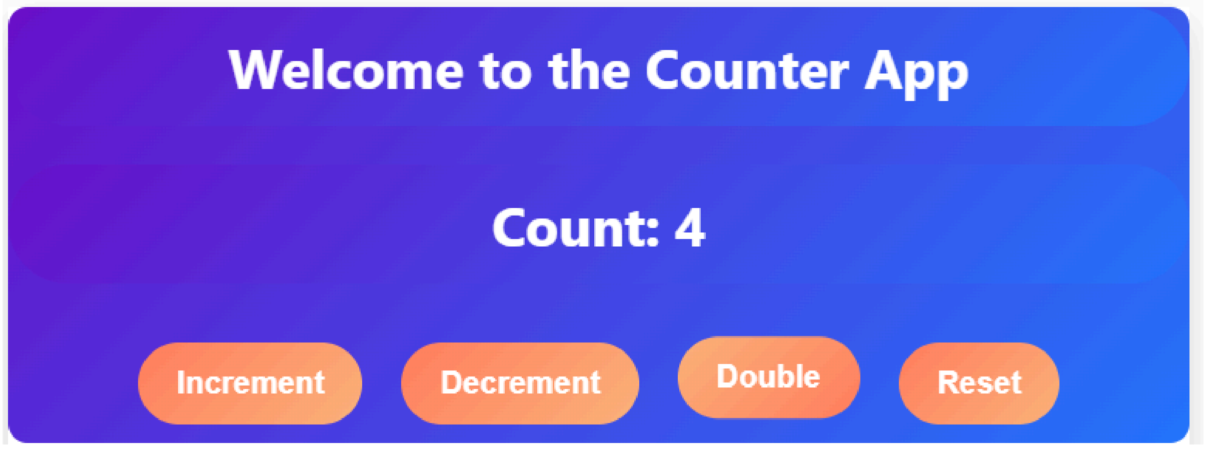
Welcome to the Counter App

Count: 0



Welcome to the Counter App

Count: 2



Welcome to the Counter App

Count: 4

Program 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.

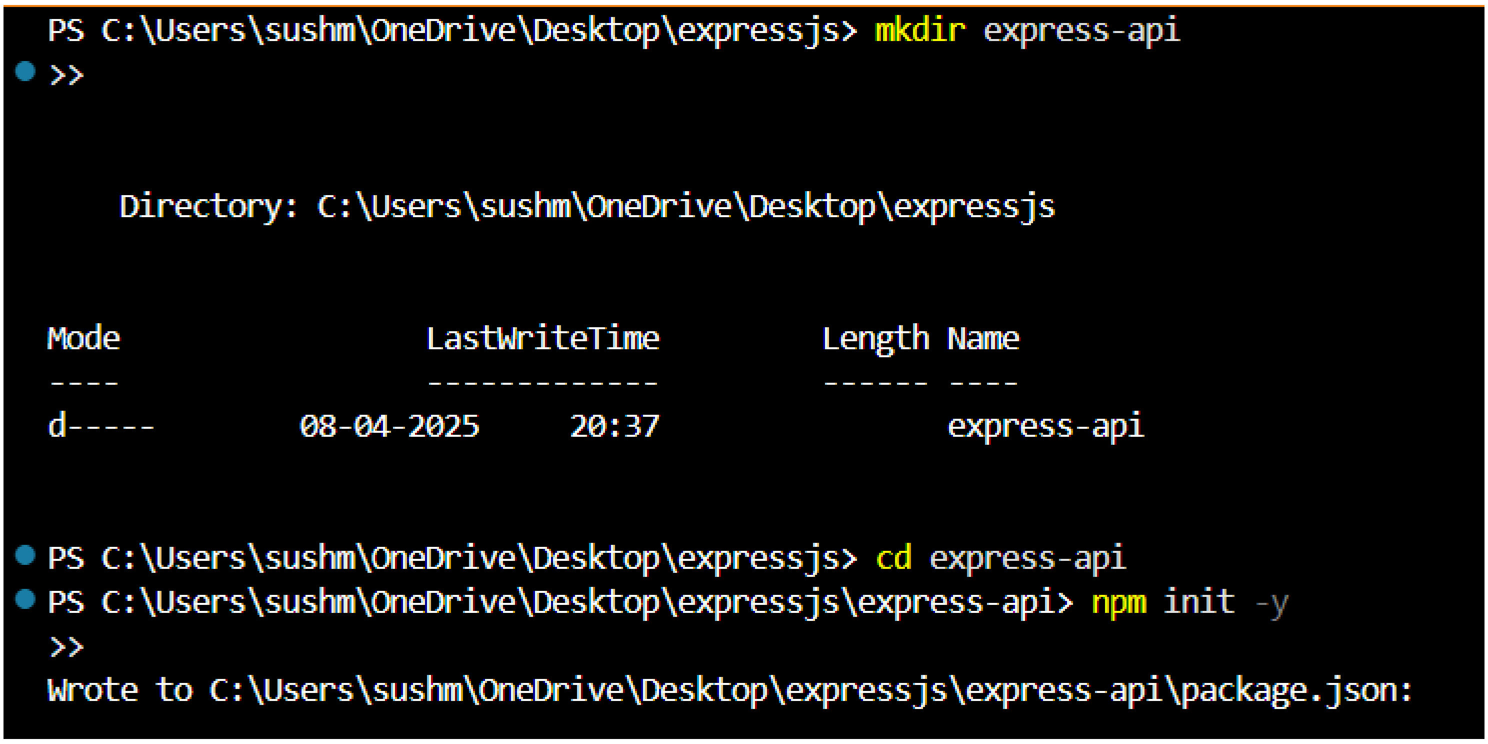
Step 1: Project Initialization

1.Create a new project directory and open it using **Visual Studio Code.**
2.Launch the integrated terminal inside VS Code.

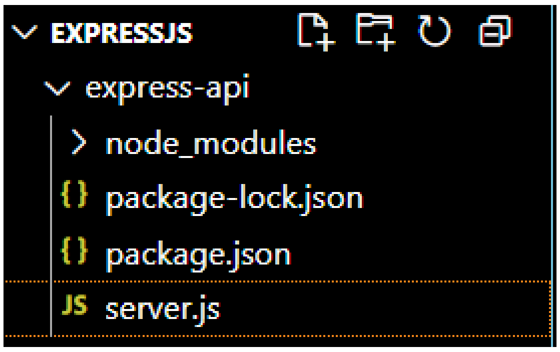
Step 2: Install Express

Execute the following commands one by one in the terminal:

```
mkdir express-api
cd express-api
npm init -y
npm install express
```



```
PS C:\Users\sushm\OneDrive\Desktop\expressjs> mkdir express-api
>>


    Directory: C:\Users\sushm\OneDrive\Desktop\expressjs


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        08-04-2025     20:37                express-api


PS C:\Users\sushm\OneDrive\Desktop\expressjs> cd express-api
PS C:\Users\sushm\OneDrive\Desktop\expressjs\express-api> npm init -y
>>
Wrote to C:\Users\sushm\OneDrive\Desktop\expressjs\express-api\package.json:
```



```
EXPRESSJS
  express-api
    node_modules
    package-lock.json
    package.json
    server.js
```

Step 3: Create the Server File

1.Inside the express-api folder, create a new file named server.js.

2.Paste the following code into server.js and save the file:

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

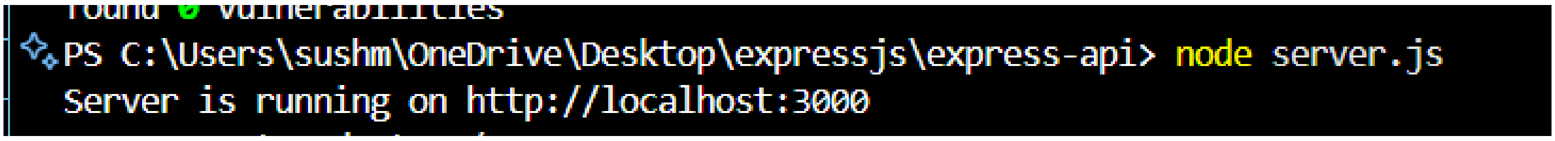
Step 4: Install and Set Up Postman

To test the API endpoints, download and install Postman:

1. Visit the official Postman download page.
2. Download the installer for your operating system.
3. Run the installer and follow the setup instructions.
4. (Optional) Create a free Postman account for syncing your workspace across devices.

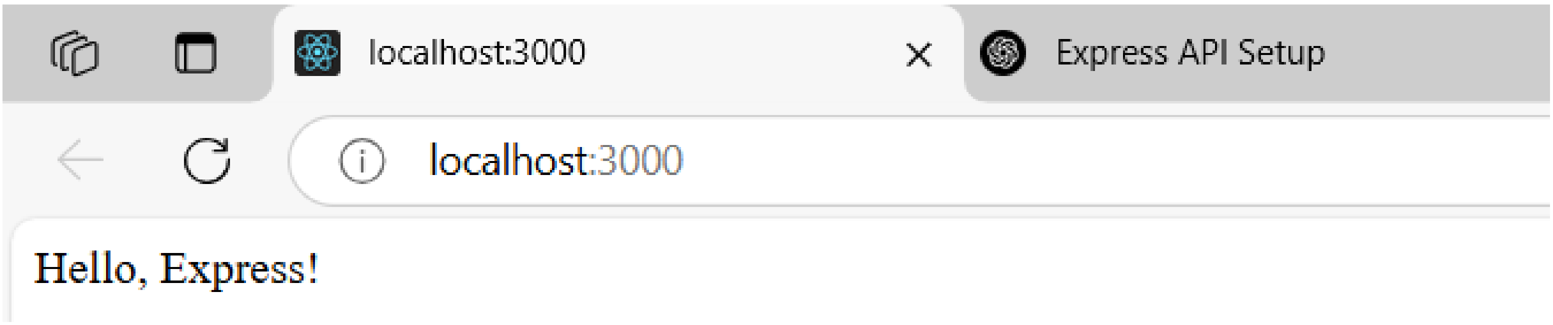
Step 5: Launch the Server

In the terminal, run the following command to start the Express server:



```
found 0 vulnerabilities
PS C:\Users\sushm\OneDrive\Desktop\expressjs\express-api> node server.js
Server is running on http://localhost:3000
```

The server will be available at http://localhost:3000, and you can begin sending requests using Postman.



Hello, Express!

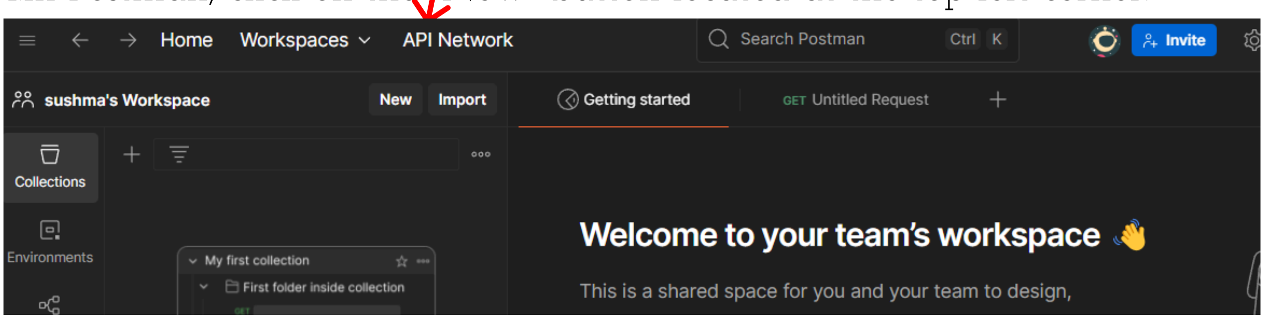
Step 6: Open Postman

After starting your Express server, use Postman to test your API endpoints:

1. Launch the Postman application.
2. Ensure your server (e.g., node server.js) is running in the terminal.

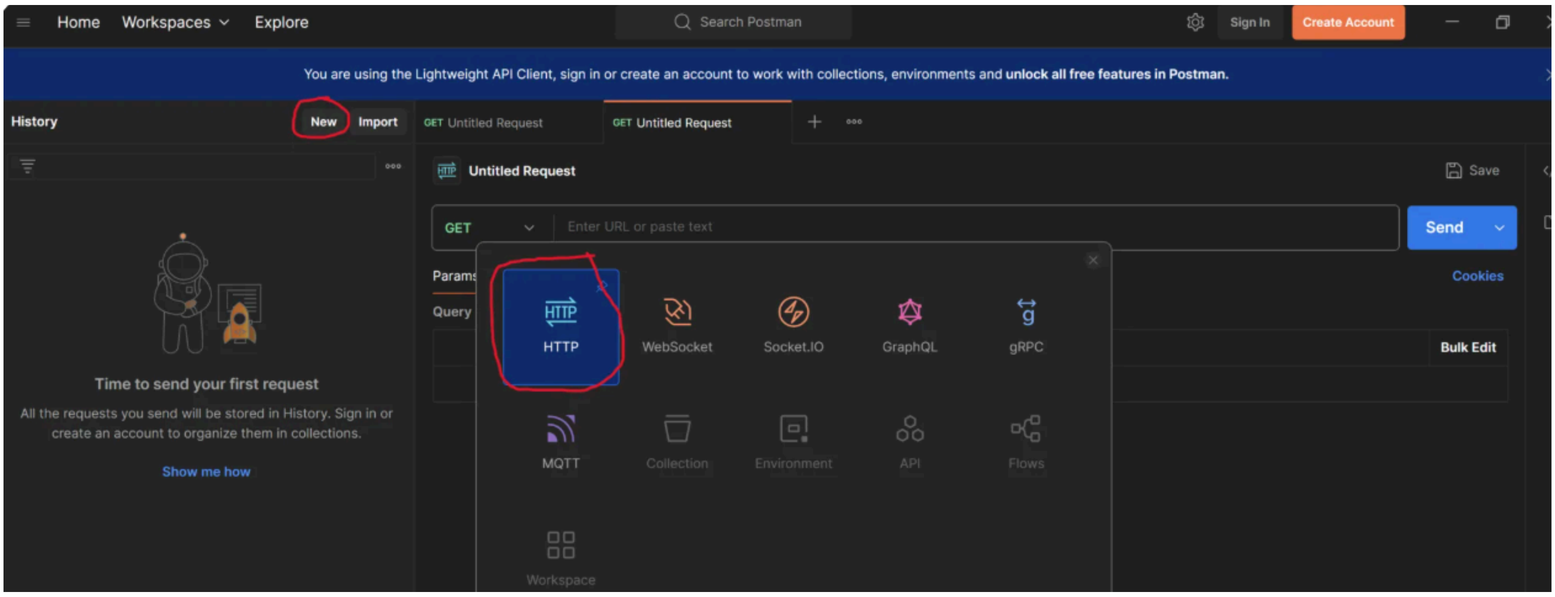
Step 7: Create and Send a Request

1.In Postman, click on the "New" button located at the top-left corner.



Welcome to your team's workspace 👋

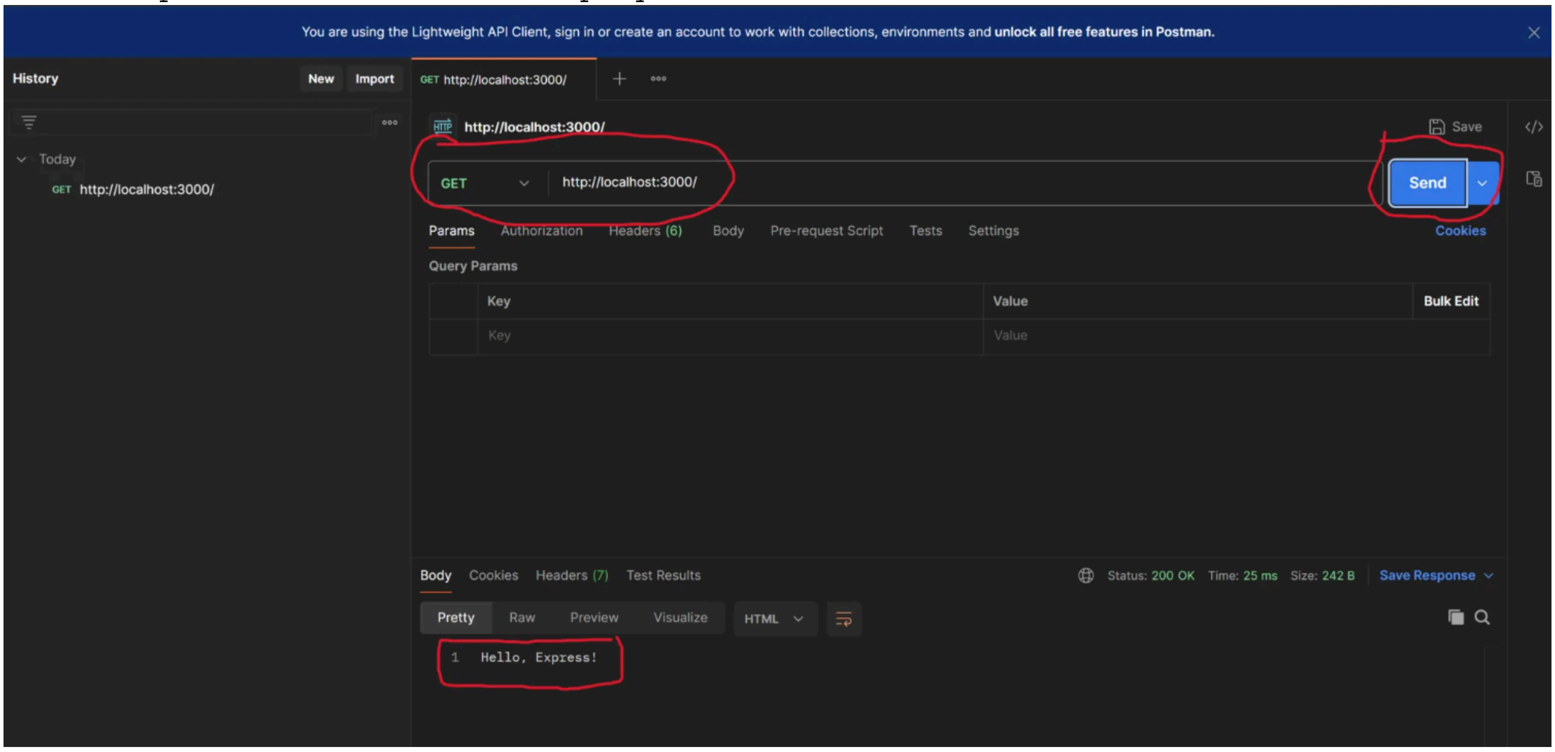
This is a shared space for you and your team to design, 2.Select "HTTP Request" from the options provided.


Home
Workspaces
Explore
Search Postman
Sign In
Create Account
You are using the Lightweight API Client, sign in or create an account to work with collections, environments and unlock all free features in Postman.
History
New
Import
GET Untitled Request
GET Untitled Request
Untitled Request
Save
GET
Enter URL or paste text
Send
Cookies
Bulk Edit
Time to send your first request
All the requests you send will be stored in History. Sign in or create an account to organize them in collections.
Show me how
HTTP
WebSocket
Socket.IO
GraphQL
gRPC
MQTT
Collection
Environment
API
Flows
Workspace

3.In the request window:

- Set the request type to GET.
- Enter the URL http://localhost:3000/ in the address bar (or as shown in the terminal when your server starts).

4.Click the "Send" button.
5.The response window should display:



```
Hello, Express!
```

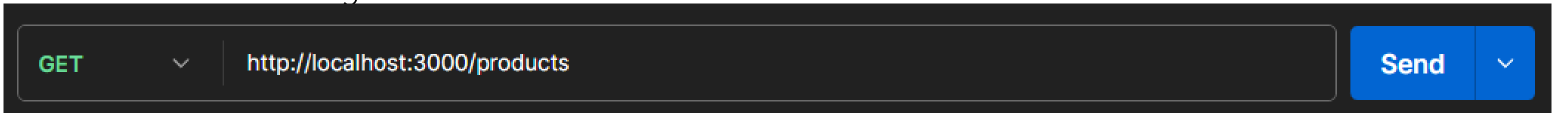
This confirms that your server is working and responding to requests correctly.

Step 8: Test Fetching All Products (GET /products)

To retrieve the list of all available products using the API, follow the steps below:

Steps to Test:

1.Open Postman and create a new request (as explained in Step 7).
2.Set the HTTP method to GET.
3.Enter the following URL in the address bar:



```
http://localhost:3000/products
```

Click on the "Send" button.

You should receive a JSON response containing the list of products. The response will look similar to:

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

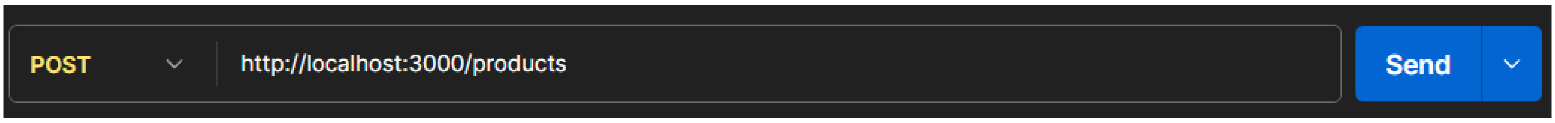
This confirms that the **GET /products** endpoint is functioning as expected and returning the in-memory product list.

Step 9: Test Adding a New Product (POST /products)

This step allows you to test the API's ability to add a new product to the product list using a POST request.

Steps to Test:

1. Open Postman and create a new request (refer to Step 7 if needed).
2. Set the HTTP method to POST.
3. In the URL field, enter:



POST http://localhost:3000/products

4. Click the "Send" button without entering any data in the body.
   - You should receive a response like:

```
{
  "message": "Name and price are required"
}
```

Now to send valid product data:

- *Click on the Body tab.*
- *Select raw.*
- *From the dropdown next to raw, choose JSON.*

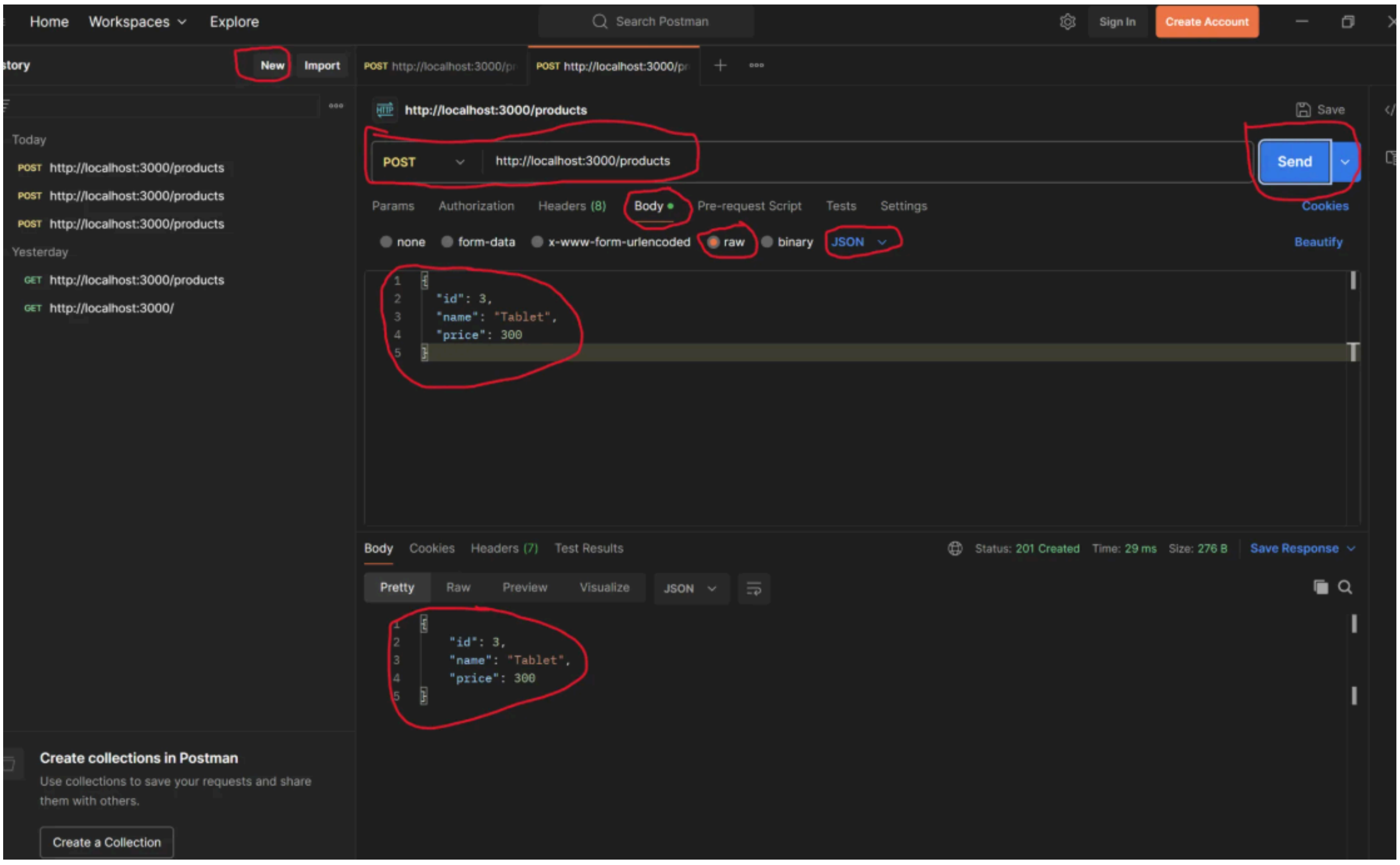
Enter the following JSON data:

```
{
  "name": "Tablet",
  "price": 700
}
```

Click "Send".
You should receive a success response with the new product details:

```
{
  "id": 3,
  "name": "Tablet",
  "price": 700
}
```

This confirms that the POST /products endpoint successfully adds a new product when valid JSON data is provided.



History: Today — POST http://localhost:3000/products; POST http://localhost:3000/products; POST http://localhost:3000/products. Yesterday — GET http://localhost:3000/products; GET http://localhost:3000/

POST http://localhost:3000/products

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

Status: 201 Created Time: 29 ms Size: 276 B

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

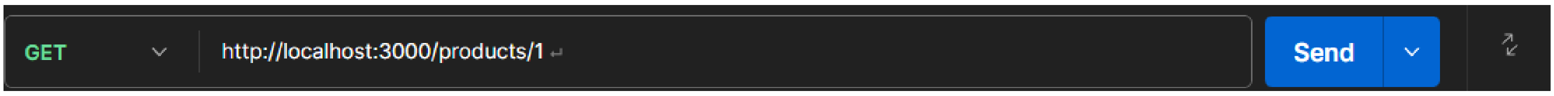
Create collections in Postman
Use collections to save your requests and share them with others.
Create a Collection

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

This step demonstrates how to retrieve the details of a single product using its ID.

Steps to Test:

1.Open Postman and create a new request.
2.Set the HTTP method to GET.
3.In the URL field, enter:



```
http://localhost:3000/products/1
```

Replace 1 with the ID of the product you want to retrieve.

4.Click the "Send" button.
5.If the product with the specified ID exists, you will receive a response similar to:

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

If the product is not found, the response will be:

```
{
  "message": "Product not found"
}
```

This confirms that the GET /products/:id endpoint correctly retrieves individual product data based on the provided ID.

Step 11: Test Updating a Product (PUT /products/:id)

This step demonstrates how to update the details of an existing product using its ID.

Steps to Test:

1.Open Postman and create a new request.
2.Set the HTTP method to PUT.
3.In the URL field, enter:

```bash
http://localhost:3000/products/1
```

Replace `1` with the ID of the product you wish to update.

4.Click on the Body tab.
5.Select raw, then choose JSON from the dropdown menu.
6.Enter the following updated JSON data:

```json
{
  "name": "Gaming Laptop",
  "price": 1500
}
```

7.Click the "Send" button.
8.You should receive a success response showing the updated product:

```
{
```

9. If the product ID does not exist or if any required fields are missing, appropriate error messages such as the following will be returned:

json Copy Edit

```json
{
  "message": "Product not found"
}
```

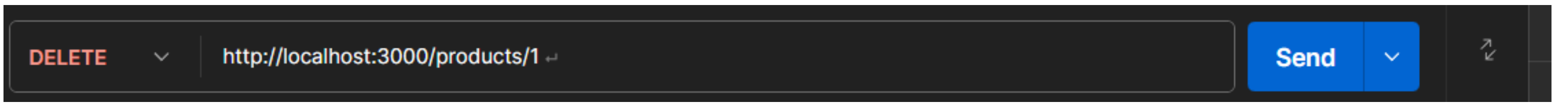
This confirms that the PUT /products/:id endpoint is functioning correctly and updates the specified product.

Step 12: Test Deleting a Product (DELETE /products/:id)

This step verifies the functionality of deleting a specific product using its ID.

Steps to Test:

1.Open Postman and create a new request.
2.Set the HTTP method to DELETE.
3.In the URL field, enter:


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

4.Click the "Send" button.

Expected Response:

- If the product exists and is deleted successfully, the server will return:

yaml Copy Edit

```yaml
Status: 204 No Content
```

> **This status indicates that the product was deleted successfully, and there is no content in the response body.**

Program 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.

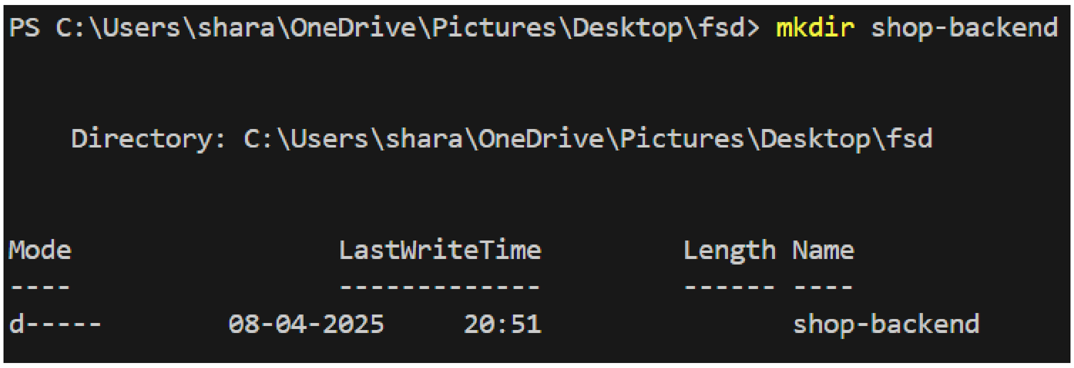
Step 1: Create Your Project Folder

> Make a new folder for your project and open it in VS Code.
> Open the integrated terminal in VS Code.

Step 2: Establish the Backend System

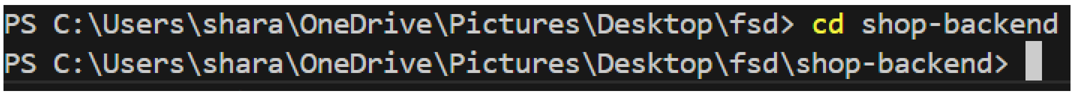
> Execute these commands in your terminal to prepare your project environment.
> Install mongoose and other required packages step by step using separate commands.

```
mkdir shop-backend
```



```
PS C:\Users\shara\OneDrive\Pictures\Desktop\fsd> mkdir shop-backend


    Directory: C:\Users\shara\OneDrive\Pictures\Desktop\fsd


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        08-04-2025     20:51                shop-backend
```

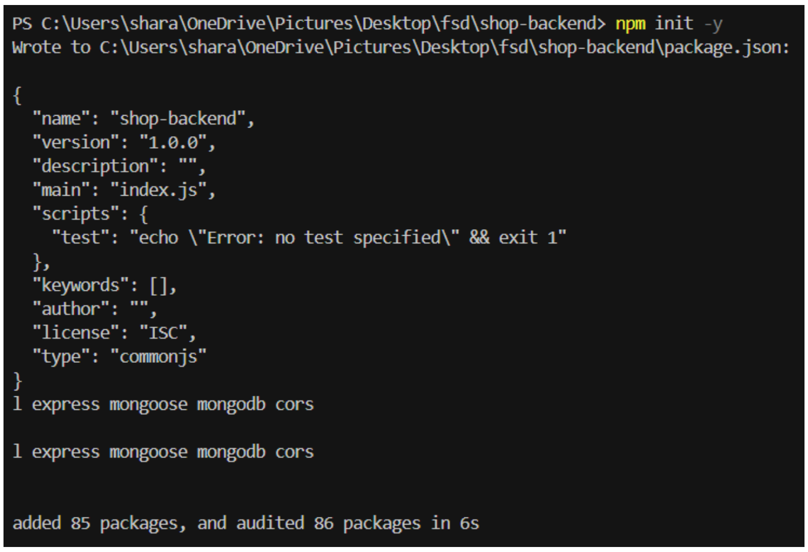
```
cd shop-backend
```



```
PS C:\Users\shara\OneDrive\Pictures\Desktop\fsd> cd shop-backend
PS C:\Users\shara\OneDrive\Pictures\Desktop\fsd\shop-backend>
```

Step 3: Initialize Node.js Project

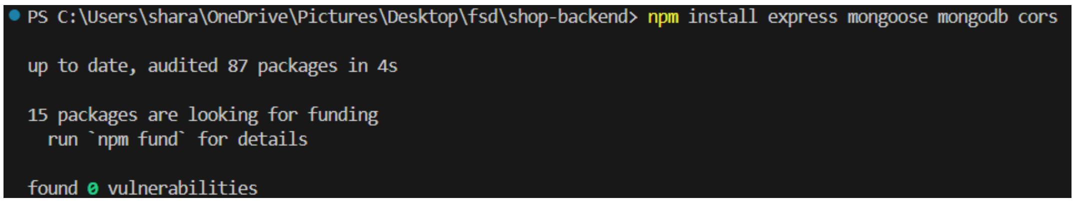
```
npm init -y
```



```
PS C:\Users\shara\OneDrive\Pictures\Desktop\fsd\shop-backend> npm init -y
Wrote to C:\Users\shara\OneDrive\Pictures\Desktop\fsd\shop-backend\package.json:

{
  "name": "shop-backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs"
}
l express mongoose mongodb cors

l express mongoose mongodb cors


added 85 packages, and audited 86 packages in 6s
```

Step 4: Install Required Packages

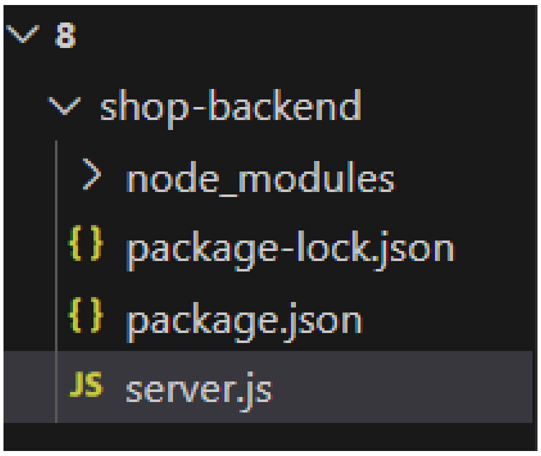
```
npm install express mongoose mongodb cors
```



```
PS C:\Users\shara\OneDrive\Pictures\Desktop\fsd\shop-backend> npm install express mongoose mongodb cors

up to date, audited 87 packages in 4s

15 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Step 5: Create the Server File

> Right-click on the shop-backend folder and create a new file named server.js.
> Paste the provided code into the file and save it.



```
8
  shop-backend
    node_modules
    package-lock.json
    package.json
    server.js
```

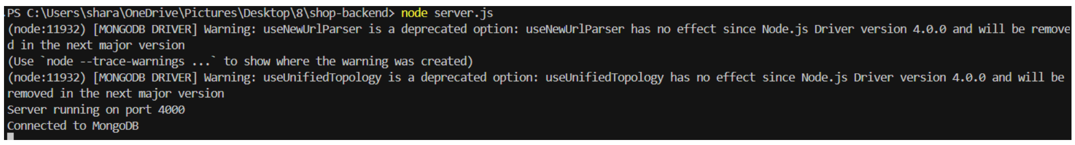
```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');

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

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

const Product = mongoose.model('Product', new mongoose.Schema({
  name: String,
  price: Number,
  description: String
}));
```

```
app.get('/products', async (_, res) ⇒ res.json(await
Product.find()));

app.post('/products', async (req, res) ⇒ {
  const product = new Product(req.body);
  await product.save();
  res.status(201).json(product);
});

app.put('/products/:id', async (req, res) ⇒ {
  const updated = await
Product.findByIdAndUpdate(req.params.id, req.body, {
new: true });
  res.json(updated);
});

app.delete('/products/:id', async (req, res) ⇒ {
  await Product.findByIdAndDelete(req.params.id);
  res.status(204).end();
});

app.listen(4000, () ⇒ console.log('Server running on
port 4000'));
```

Step 6: Launch the Backend Server

Run the following command in the terminal to start your backend server

```
node server.js
```



```
PS C:\Users\shara\OneDrive\Pictures\Desktop\8\shop-backend> node server.js
(node:11932) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser has no effect since Node.js Driver version 4.0.0 and will be remove
d in the next major version
(Use `node --trace-warnings ...` to show where the warning was created)
(node:11932) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 and will be
removed in the next major version
Server running on port 4000
Connected to MongoDB
```

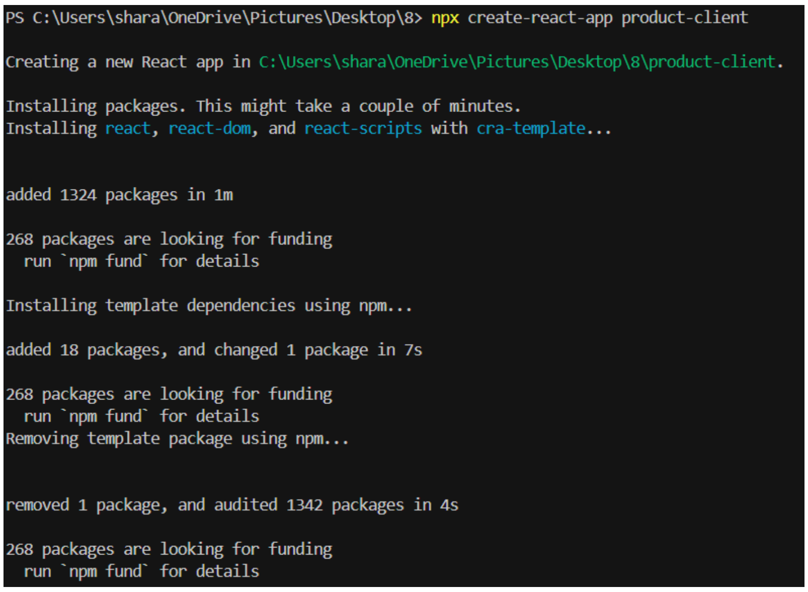
> Once it's running, you should see a message in the terminal confirming that the server is active on port 4000 and successfully connected to MongoDB.

Step 7: Set Up the Frontend

> Open a new terminal window while keeping the backend server running.

> Run the below command to generate a new folder for the frontend.

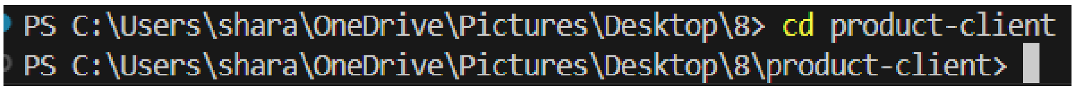
```
npx create-react-app product-client
```



```
PS C:\Users\shara\OneDrive\Pictures\Desktop\8> npx create-react-app product-client

Creating a new React app in C:\Users\shara\OneDrive\Pictures\Desktop\8\product-client.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...


added 1324 packages in 1m

268 packages are looking for funding
  run `npm fund` for details

Installing template dependencies using npm...

added 18 packages, and changed 1 package in 7s

268 packages are looking for funding
  run `npm fund` for details
Removing template package using npm...


removed 1 package, and audited 1342 packages in 4s

268 packages are looking for funding
  run `npm fund` for details
```

> Once the setup is complete, switch to the new directory using the below command:

```
cd product-client
```



```
PS C:\Users\shara\OneDrive\Pictures\Desktop\8> cd product-client
PS C:\Users\shara\OneDrive\Pictures\Desktop\8\product-client>
```

Step 8: Modify App.js and App.css Files

> Replace the content of the **App.js** file with the provided code and save the changes.

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

const API = 'http://localhost:4000/products';

const App = () => {
  const [products, setProducts] = useState([]);
  const [form, setForm] = useState({ name: '', price: '', description: '' });
  const [editingId, setEditingId] = useState(null);
  const [errors, setErrors] = useState({});

  useEffect(() => {
    fetch(API).then(res => res.json()).then(setProducts);
  }, []);

  const handleChange = e => {
    const { name, value } = e.target;
    setForm(f => ({ ...f, [name]: value }));
    setErrors(e => ({ ...e, [name]: '' }));
  };
```

```
 const validate = () => {
   const errs = Object.fromEntries(
     Object.entries(form).filter(([k, v]) => !v).map(([k]) => [k,
`${k[0].toUpperCase() + k.slice(1)} is required.`])
   );
   setErrors(errs);
   return !Object.keys(errs).length;
 };

 const handleSubmit = () => {
   if (!validate()) return;
   fetch(editingId ? `${API}/${editingId}` : API, {
     method: editingId ? 'PUT' : 'POST',
     headers: { 'Content-Type': 'application/json' },
     body: JSON.stringify(form)
   }).then(() => {
     setForm({ name: '', price: '', description: '' });
     setEditingId(null);
     fetch(API).then(res => res.json()).then(setProducts);
   });
 };

 const handleEdit = ({ _id, name, price, description }) => {
   setForm({ name, price, description });
   setEditingId(_id);
 };

 const handleDelete = id => {
   window.confirm('Delete this product?') &&
     fetch(`${API}/${id}`, { method: 'DELETE' }).then(() =>
       fetch(API).then(res => res.json()).then(setProducts)
     );
 };
```

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

      {products.length ? (
        <table className="product-table">
          <thead>
            <tr><th>Name</th><th>Price</th><th>Description</
th><th>Actions</th></tr>
          </thead>
          <tbody>
            {products.map(({ _id, name, price, description }) => (
              <tr key={_id}>
                <td>{name}</td>
                <td>₹{price}</td>
                <td>{description}</td>
                <td className="action-buttons">
                  <button onClick={() => handleEdit({ _id, name, price, description
})} className="edit-button">Edit</button>
                  <button onClick={() => handleDelete(_id)} className="delete-
button">Delete</button>
                </td>
              </tr>
            ))}
          </tbody>
        </table>
      ) : (
        <p className="no-products-message">No products. Please add one.</
p>
      )}
```

```
 <div className="form-section">
        <h2 className="form-title">{editingId ? 'Edit Product' : 'Add
Product'}</h2>
        {['name', 'price', 'description'].map(field => (
          <div className="input-wrapper" key={field}>
            {field === 'description' ? (
              <textarea name={field} rows={3} className="input-field"
placeholder={`Product ${field}`}
                value={form[field]} onChange={handleChange} />
            ) : (
              <input type={field === 'price' ? 'number' : 'text'} name={field}
className="input-field"
                placeholder={`Product ${field}`} value={form[field]}
onChange={handleChange} />
            )}
            {errors[field] && <p className="error-message">{errors[field]}</
p>}
          </div>
        ))}
        <button onClick={handleSubmit} className={editingId ? 'update-
button' : 'add-button'}>
          {editingId ? 'Update' : 'Add'}
        </button>
      </div>
    </div>
  );
};

export default App;
```

> Next, open the **App.css** file, paste the given styling code into it, and save it as well.

```css
/* Container & Title */
.container {
  max-width: 1000px;
  margin: auto;
  padding: 20px;
  font-family: Arial, sans-serif;
  background: #fff;
  border-radius: 8px;
  box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
}

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

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

.product-table thead {
  background: #007bff;
  color: #fff;
  text-align: left;
}
```

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

.product-table tbody tr:nth-child(even) {
  background: #f1f1f1;
}

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

/* Buttons */
button {
  padding: 10px 20px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  transition: background-color 0.2s;
  color: #fff;
  font-weight: 500;
  margin-top: 10px;
}

.add-button    { background: #28a745; }
.add-button:hover { background: #218838; }

.update-button    { background: #ffc107; color: #000; margin-top: 20px; }
.update-button:hover { background: #e0a800; }

.edit-button    { background: #007bff; }
.edit-button:hover { background: #0056b3; }

.delete-button    { background: #dc3545; }
.delete-button:hover { background: #c82333; }
```

```
/* Form */
.form-section {
  margin-top: 40px;
}

.form-title {
  font-size: 1rem;
  background: #0001;
  color: #000;
  padding: 7px 12px;
  border-radius: 7px;
  margin-bottom: 10px;
  width: fit-content;
}

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

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

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

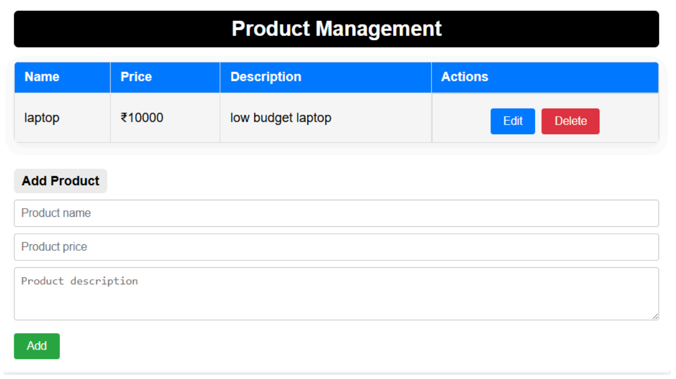
```
.no-products-message {
  text-align: center;
  font-size: 1.2rem;
  color: #666;
  background: #f9f9f9;
  padding: 10px;
  border-radius: 8px;
  margin-top: 20px;
}

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

Step 9: Run the React development server

> Enter the command below to launch the frontend, allowing you to view and interact with your application in the browser.

```
npm start
```

Output



Product Management

Name	Price	Description	Actions
laptop	₹10000	low budget laptop	Edit Delete

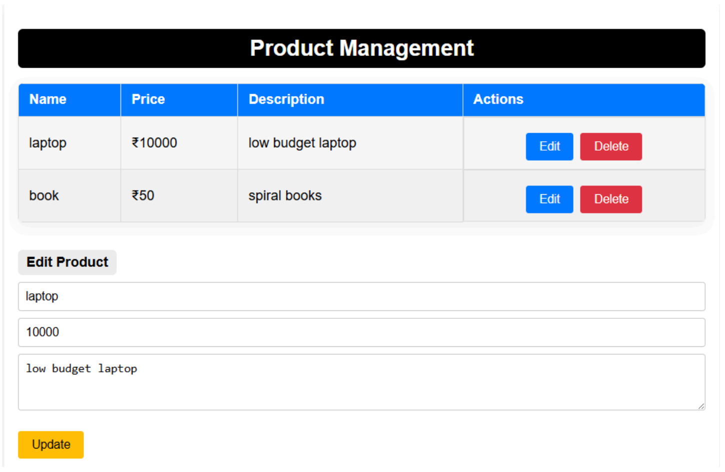
Add Product

Product name

Product price

Product description

Add



Product Management

Name	Price	Description	Actions
laptop	₹10000	low budget laptop	Edit Delete
book	₹50	spiral books	Edit Delete

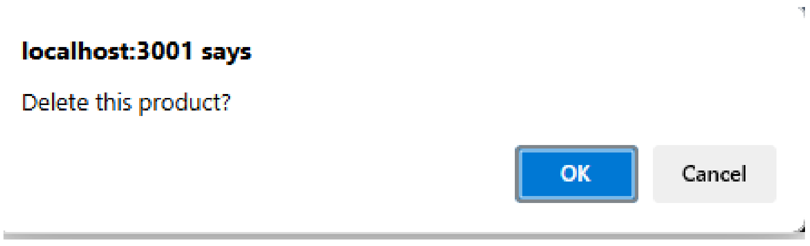
Edit Product

laptop

10000

low budget laptop

Update



localhost:3001 says

Delete this product?

OK Cancel