

Ex.No: 1.i	Personal portfolio
Date:	

Aim:

Algorithm:-

Program:-

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>My Portfolio</title>
  <link rel="stylesheet" href="index.css" />
</head>
<body>
  <header>
    <h1>Gokul</h1>
    <nav>
      <a href="#about">About Me</a>
      <a href="#skills">Skills</a>
      <a href="#projects">Projects</a>
      <a href="#contact">Contact</a>
    </nav>
  </header>

  <section id="about">
    <h2>About Me</h2>
    <p>I am a web developer with experience in HTML, CSS, JavaScript,
    and the MERN stack. I enjoy building responsive and user-friendly web
    applications.</p>
  </section>

  <section id="skills">
    <h2>Skills</h2>
    <ul>
      <li>HTML & CSS</li>
      <li>JavaScript</li>
      <li>React</li>
      <li>Node.js</li>
      <li>MongoDB</li>
    </ul>
  </section>

  <section id="projects">
    <h2>Projects</h2>
    <div class="project">
      <h3>Task Manager App</h3>
      <p>A full-stack task manager app built using React and Express.</p>
    </div>
    <div class="project">
      <h3>Weather App</h3>
      <p>A simple weather app using OpenWeatherMap API.</p>
    </div>
  </section>

  <section id="contact">
    <h2>Contact</h2>
    <p>Email: gokul@example.com</p>
    <p>Phone: +91-1234567890</p>
  </section>
</body>
</html>
```

```
</section>
<footer>
  <p>© 2025 Gokul. All rights reserved.</p>
</footer>
</body>
</html>
```

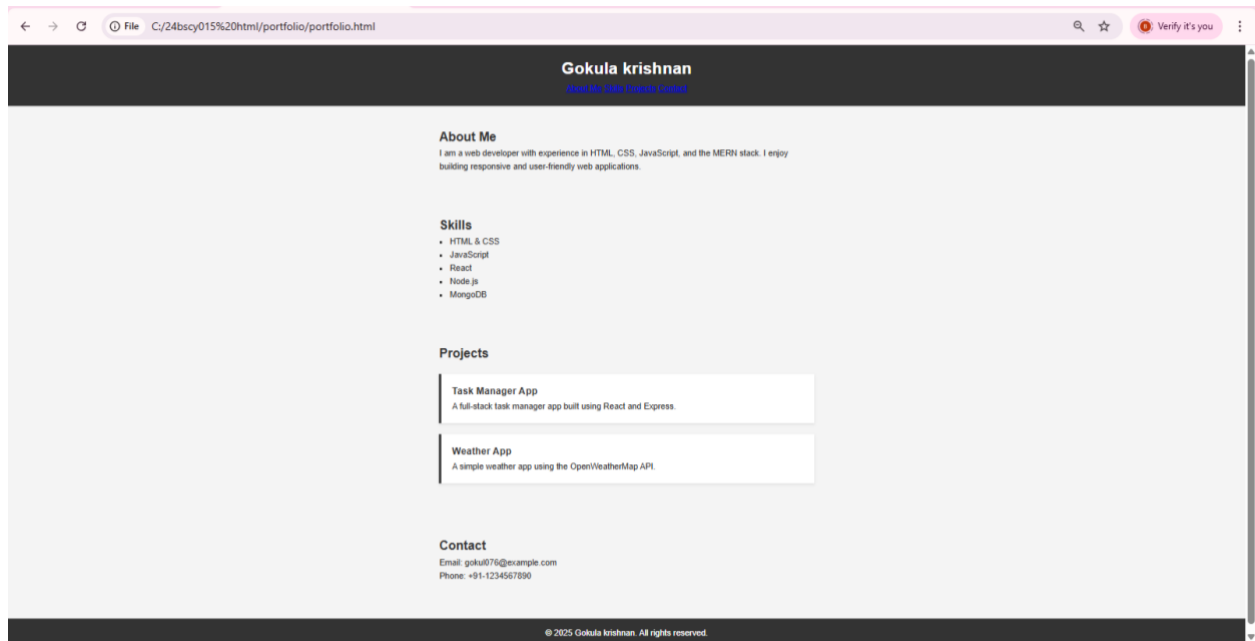
Index.css

```
*{
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
body {
  font-family: Arial, sans-serif;
  line-height: 1.6;
  background: #f4f4f4;
  color: #333;
}
header {
  background: #333;
  color: #fff;
  padding: 20px 0;
  text-align: center;
}
header nav a {
  color: #fff;
  margin: 0 15px;
  text-decoration: none;
}
header nav a:hover {
  text-decoration: underline;
}
section {
  padding: 40px;
  max-width: 800px;
  margin: auto;
}

section img {
  max-width: 150px;
  border-radius: 50%;
  margin-top: 20px;
}
ul {
  list-style-type: square;
  padding-left: 20px;
}
.project {
  background: #fff;
  padding: 20px;
  margin: 20px 0;
  border-left: 5px solid #333;
  box-shadow: 0 2px 5px rgba(0,0,0,0.1);
}
```

```
footer {  
  background: #333;  
  color: #fff;  
  text-align: center;  
  padding: 15px 0;  
  margin-top: 30px;  
}
```

Output:



Result:

The program successfully creates a simple and structured personal portfolio web page using basic HTML and CSS, achieving the desired output.

Ex.No: 1.ii	Create a responsive page with flexbox
Date:	

Aim:

Algorithm:-

Program:-

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Responsive Flexbox Layout</title>
  <link rel="stylesheet" href="index.css" />
</head>
<body>
  <header>
    <h1>My Flexbox Page</h1>
    <nav>
      <a href="#">Home</a>
      <a href="#">About</a>
      <a href="#">Projects</a>
      <a href="#">Contact</a>
    </nav>
  </header>

  <div class="container">
    <main>
      <h2>Main Content</h2>
      <p>This is the main content area. It takes up more space than the sidebar.</p>
    </main>

    <aside>
      <h2>Sidebar</h2>
      <p>This is the sidebar. It contains extra information or links.</p>
    </aside>
  </div>

  <footer>
    <p>© 2025 My Flexbox Layout. All rights reserved.</p>
  </footer>
</body>
</html>
```

index.css

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: 'Segoe UI', sans-serif;
  background: #f0f0f0;
  color: #333;
}
```

```
header {
  background-color: #333;
  color: white;
  padding: 20px;
  text-align: center;
}

nav {
  margin-top: 10px;
}

nav a {
  color: white;
  margin: 0 15px;
  text-decoration: none;
}

nav a:hover {
  text-decoration: underline;
}

.container {
  display: flex;
  padding: 20px;
  gap: 20px;
}

main {
  flex: 3;
  background: white;
  padding: 20px;
  box-shadow: 0 2px 5px rgba(0,0,0,0.1);
}

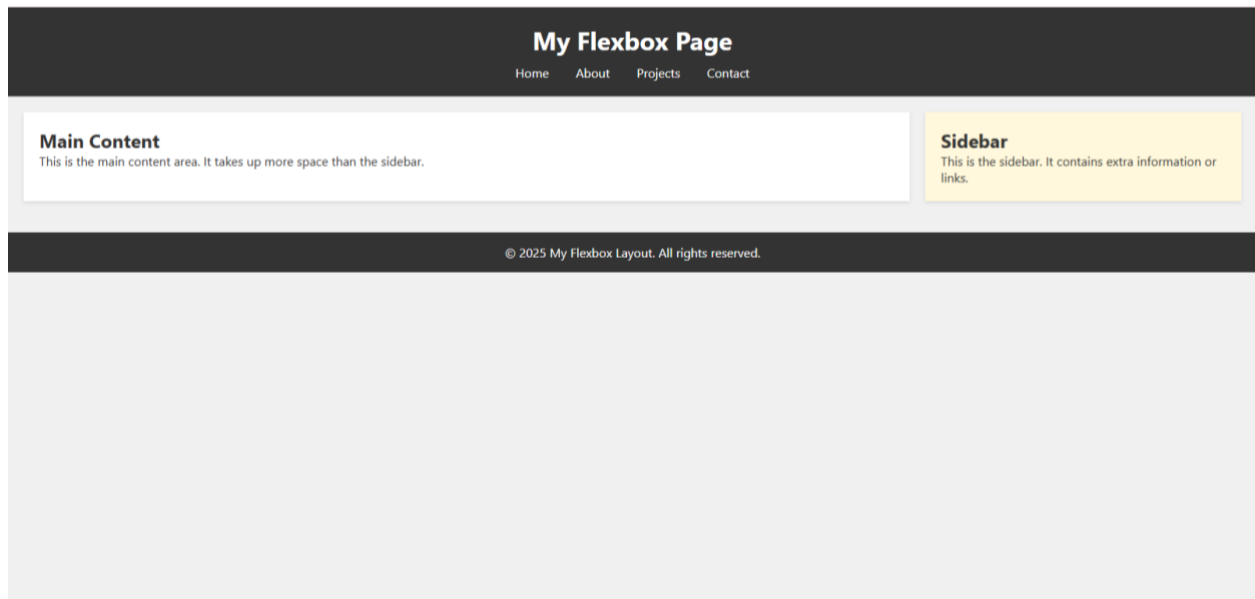
aside {
  flex: 1;
  background: #fff8dc;
  padding: 20px;
  box-shadow: 0 2px 5px rgba(0,0,0,0.1);
}

footer {
  background: #333;
  color: white;
  text-align: center;
  padding: 15px;
  margin-top: 20px;
}

@media (max-width: 768px) {
  .container {
    flex-direction: column;
  }
}
```

```
nav a {  
  display: inline-block;  
  margin: 5px;  
}  
}
```

Output:



Result:

A responsive web page layout is successfully created using CSS Flexbox.

The layout adapts to screen size changes by stacking the sidebar and content vertically on smaller screens.

Ex.No: 1.iii	Create a webpage using multi-column blog layout
Date:	

Aim:

Algorithm:-

Program:-

```
index.html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Multi-Column Blog Layout</title>
  <link rel="stylesheet" href="index.css" />
</head>
<body>
  <header>
    <h1>My Blog</h1>
  </header>
  <section class="blog-container">
    <article class="blog-post">
      <h2>Post One</h2>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque euismod, nisi vel
consectetur.</p>
      <a href="#">Read More</a>
    </article>

    <article class="blog-post">
      <h2>Post Two</h2>
      <p>Aliquam erat volutpat. Duis eget tincidunt augue, in laoreet libero. Nullam at blandit urna.</p>
      <a href="#">Read More</a>
    </article>

    <article class="blog-post">
      <h2>Post Three</h2>
      <p>Curabitur lacinia, nulla ac malesuada posuere, lectus purus mattis ligula, ac porta purus enim.</p>
      <a href="#">Read More</a>
    </article>

    <article class="blog-post">
      <h2>Post Four</h2>
      <p>Donec dignissim diam at lorem convallis, ac rutrum arcu tincidunt. Sed porta sapien vitae
ante.</p>
      <a href="#">Read More</a>
    </article>

    <article class="blog-post">
      <h2>Post Five</h2>
      <p>Sed nec ipsum ac felis tristique pharetra nec nec nulla. Suspendisse potenti.</p>
      <a href="#">Read More</a>
    </article>

    <article class="blog-post">
      <h2>Post Six</h2>
      <p>Phasellus tristique orci quis lacus pulvinar, eget rhoncus nulla porta. In nec tortor id.</p>
      <a href="#">Read More</a>
    </article>
  </section>
```

```
<footer>
  <p>© 2025 My Blog. All rights reserved.</p>
</footer>
</body>
</html>
```

Index.css

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
```

```
body {
  font-family: 'Segoe UI', sans-serif;
  background-color: #f9f9f9;
  color: #333;
}
```

```
header {
  background-color: #333;
  color: white;
  padding: 20px;
  text-align: center;
}
```

```
.blog-container {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
  gap: 20px;
  padding: 30px;
  max-width: 1200px;
  margin: auto;
}
```

```
.blog-post {
  background-color: white;
  padding: 20px;
  border-left: 5px solid #007BFF;
  box-shadow: 0 2px 6px rgba(0, 0, 0, 0.1);
}
```

```
.blog-post h2 {
  margin-bottom: 10px;
  color: #007BFF;
}
```

```
.blog-post p {
  margin-bottom: 10px;
}
```

```
.blog-post a {
  color: #007BFF;
  text-decoration: none;
  font-weight: bold;
}

.blog-post a:hover {
  text-decoration: underline;
}

footer {
  background-color: #333;
  color: white;
  text-align: center;
  padding: 15px;
  margin-top: 30px;
}
```

Output:



Result:

A multi-column blog layout is successfully created. Blog posts are displayed in three columns on large screens, two on medium screens, and stacked on small screens, ensuring responsive readability.

Ex.No: 2.i	Form validation using JavaScript
Date:	

Aim:

Algorithm:-

Program:-

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Form Validation</title>
  <link rel="stylesheet" href="index.css" />
</head>
<body>
  <div class="form-container">
    <h2>Register</h2>
    <form id="registerForm">
      <label for="name">Name:</label>
      <input type="text" id="name" />

      <label for="email">Email:</label>
      <input type="email" id="email" />

      <label for="password">Password:</label>
      <input type="password" id="password" />

      <button type="submit">Submit</button>
      <p id="errorMsg" class="error"></p>
      <p id="successMsg" class="success"></p>
    </form>
  </div>

  <script src="index.js"></script>
</body>
</html>
```

Index.css

```
body {
  font-family: Arial, sans-serif;
  background: #f9f9f9;
  padding: 40px;
}

.form-container {
  max-width: 400px;
  margin: auto;
  background: white;
  padding: 30px;
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
}

input, label, button {
  display: block;
  width: 100%;
  margin-bottom: 15px;
  font-size: 16px;
```

```
}
```

```
input {  
  padding: 8px;  
}
```

```
button {  
  background: #007BFF;  
  color: white;  
  padding: 10px;  
  border: none;  
  cursor: pointer;  
}
```

```
button:hover {  
  background: #0056b3;  
}
```

```
.error {  
  color: red;  
  font-weight: bold;  
}
```

```
.success {  
  color: green;  
  font-weight: bold;  
}
```

Index.js

```
document.getElementById("registerForm").addEventListener("submit", function(e) {  
  e.preventDefault();
```

```
  const name = document.getElementById("name").value.trim();  
  const email = document.getElementById("email").value.trim(); // no validation  
  const password = document.getElementById("password").value.trim();  
  const errorMsg = document.getElementById("errorMsg");  
  const successMsg = document.getElementById("successMsg");
```

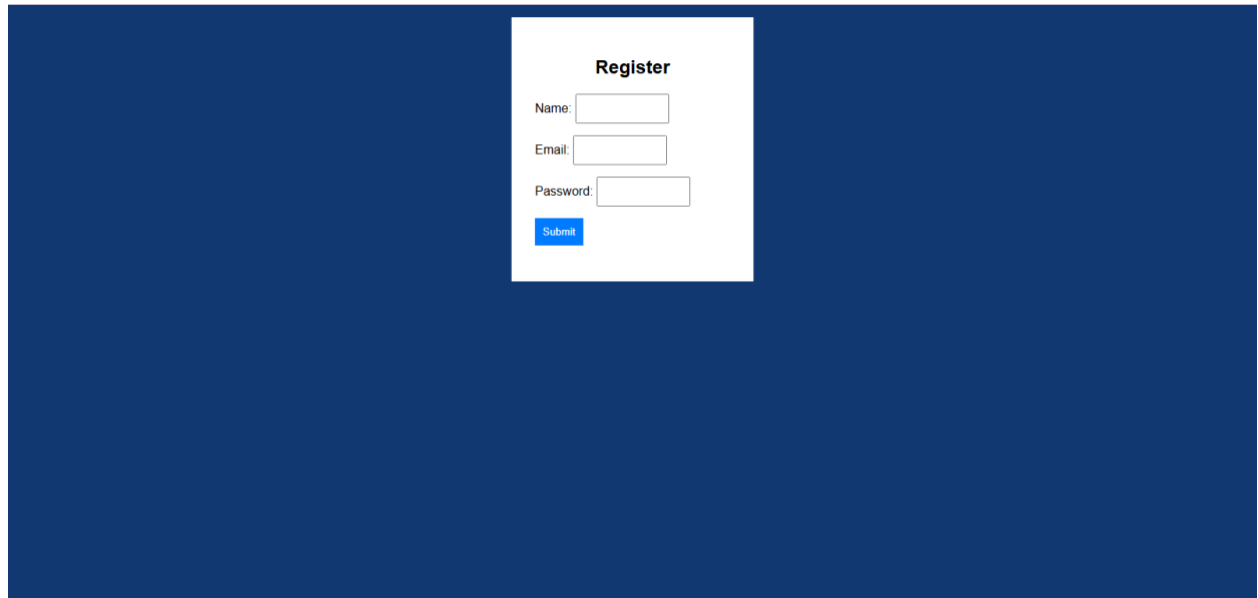
```
  errorMsg.textContent = "";  
  successMsg.textContent = "";
```

```
  if (name === "" || email === "" || password === "") {  
    errorMsg.textContent = "All fields are required.";  
    return;  
  }
```

```
  if (password.length < 8) {  
    errorMsg.textContent = "Password must be at least 8 characters.";  
    return;  
  }
```

```
  successMsg.textContent = "Form submitted successfully!";  
});
```

Output:



Register

Name:

Email:

Password:

Result

A form with fields for Name, Email, and Password is successfully validated using JavaScript.

- Empty fields show an error.
 - Incorrect email formats are rejected.
 - Short passwords are not accepted.
- The user gets real-time feedback before submitting the form.

Ex.No: 2.ii	To-do list application using JavaScript
Date:	

Aim:

Algorithm:-

Program:-

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Simple To-Do List</title>
  <link rel="stylesheet" href="index.css" />
</head>
<body>

  <h2>To-Do List</h2>
  <input type="text" id="taskInput" placeholder="Add a task">
  <button onclick="addTask()">Add</button>

  <ul id="taskList"></ul>

  <script src="index.js"></script>
</body>
</html>
```

Index.css

```
body {
  font-family: sans-serif;
  text-align: center;
  padding: 30px;
}

input, button {
  padding: 10px;
  margin: 5px;
}

ul {
  list-style: none;
  padding: 0;
}

li {
  padding: 10px;
  margin: 5px auto;
  max-width: 300px;
  border: 1px solid #ccc;
  display: flex;
  justify-content: space-between;
  cursor: pointer;
}

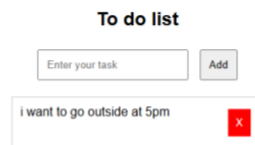
.done {
  text-decoration: line-through;
  color: gray;
}
```

```
.delete {  
  background: red;  
  color: white;  
  border: none;  
  cursor: pointer;  
}
```

Index.js

```
function addTask() {  
  const input = document.getElementById("taskInput");  
  const list = document.getElementById("taskList");  
  
  if (input.value.trim() === "") return;  
  
  const li = document.createElement("li");  
  li.innerHTML = `  
    <span onclick="this.classList.toggle('done')">${input.value}</span>  
    <button class="delete" onclick="this.parentElement.remove()">X</button>  
  `;  
  list.appendChild(li);  
  input.value = "";  
}
```

Output:



Result:

The To-Do List Application allows users to:

- Add new tasks.
 - See them displayed instantly.
 - Remove tasks by clicking the "Delete" button.
- The app uses JavaScript DOM manipulation for interactive behavior.

Ex.No: 2.iii	Image slider using JavaScript
Date:	

Aim:

Algorithm:-

Program:-

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Simple Image Slider</title>
  <link rel="stylesheet" href="index.css">
</head>
<body>

  <h2>Simple Image Slider</h2>
  <div id="slider">
    
    
    
  </div>

  <button class="btn" onclick="prev()">< Prev</button>
  <button class="btn" onclick="next()">Next ></button>

  <script src="index.js"></script>
</body>
</html>
```

Index.css

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

#slider {
  width: 400px;
  height: 250px;
  margin: 40px auto;
  position: relative;
}

#slider img {
  width: 100%;
  height: 100%;
  display: none;
}

#slider img.active {
  display: block;
}

.btn {
  margin: 10px;
  padding: 5px 15px;
  font-size: 16px;
}
```

Index.js

```
let slides = document.querySelectorAll("#slider img");
let index = 0;
function showSlide(i) {
  slides.forEach(img => img.classList.remove("active"));
  slides[i].classList.add("active");
}
```

```
function next() {
  index = (index + 1) % slides.length;
  showSlide(index);
}
```

```
function prev() {
  index = (index - 1 + slides.length) % slides.length;
  showSlide(index);
}
setInterval(next, 3000);
```

Output:



Result:

An interactive image slider is created using JavaScript.

- Clicking Next cycles forward through the images.
- Clicking Previous cycles backward.
- Images loop back when the beginning or end is reached.

Ex.No: 2.iv	Calculator using JavaScript
Date:	

Aim:

Algorithm:-

Program:-

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Simple Calculator</title>
  <link rel="stylesheet" href="index.css">
</head>
<body>

  <div class="calculator">
    <input type="text" id="display" disabled>
    <div class="buttons">
      <button onclick="clearDisplay()">C</button>
      <button onclick="appendValue('/')">/</button>
      <button onclick="appendValue('*')">*</button>
      <button onclick="appendValue('-')">-</button>

      <button onclick="appendValue('7')">7</button>
      <button onclick="appendValue('8')">8</button>
      <button onclick="appendValue('9')">9</button>
      <button onclick="appendValue('+')">+</button>

      <button onclick="appendValue('4')">4</button>
      <button onclick="appendValue('5')">5</button>
      <button onclick="appendValue('6')">6</button>
      <button onclick="calculate()">=</button>

      <button onclick="appendValue('1')">1</button>
      <button onclick="appendValue('2')">2</button>
      <button onclick="appendValue('3')">3</button>
      <button onclick="appendValue('0')">0</button>

      <button onclick="appendValue('.')">.</button>
    </div>
  </div>

  <script src="index.js"></script>
</body>
</html>
```

Index.css

```
body {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  background: #f5f5f5;
  font-family: Arial, sans-serif;
}
.calculator {
  border: 1px solid #ccc;
  padding: 20px;
```



```
background: white;
border-radius: 10px;
box-shadow: 2px 2px 10px rgba(0,0,0,0.1);
}
```

```
#display {
width: 100%;
height: 40px;
font-size: 20px;
margin-bottom: 10px;
text-align: right;
padding: 5px;
}
```

```
.buttons {
display: grid;
grid-template-columns: repeat(4, 50px);
gap: 10px;
justify-content: center;
}
```

```
button {
padding: 10px;
font-size: 18px;
cursor: pointer;
}
```

```
Index.js
function appendValue(value) {
document.getElementById("display").value += value;
}
```

```
function clearDisplay() {
document.getElementById("display").value = "";
}
```

```
function calculate() {
try {
const result = eval(document.getElementById("display").value);
document.getElementById("display").value = result;
} catch {
document.getElementById("display").value = "Error";
}
}
```

Output:



Result:

A fully functional calculator application is created using JavaScript.

It supports:

- Number input and basic arithmetic operations.
- Displaying real-time expression and result.
- Clearing/resetting input using the Clear (C) button.

Ex.No: 3.i	Weather app
Date:	

Aim:

Algorithm:-

Program:-

```
import { useState } from "react";
import axios from "axios";

export default function Weather() {
  const [city, setCity] = useState("");
  const [weather, setWeather] = useState(null);
  const [error, setError] = useState(null);

  const API_KEY = "2753a19d7ce511e1626297e0651d625e";
  const getWeather = async () => {
    try {
      setError(null);
      const response = await axios.get(
        https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${API_KEY}&units=me
tric
      );
      setWeather(response.data);
    } catch (err) {
      setWeather(null);
      setError("City not found");
    }
  };

  return (
    <div className="flex flex-col items-center justify-center min-h-screen bg-blue-100 p-6">
      <h1 className="text-3xl font-bold mb-4">🌤 Weather App</h1>

      <div className="flex gap-2 mb-4">
        <input
          type="text"
          value={city}
          onChange={(e) => setCity(e.target.value)}
          placeholder="Enter city"
          className="p-2 rounded-xl border border-gray-400"
        />
        <button
          onClick={getWeather}
          className="px-4 py-2 bg-blue-500 text-white rounded-xl hover:bg-blue-600"
        >
          Search
        </button>
      </div>

      {error && <p className="text-red-500">{error}</p>}

      {weather && (
        <div className="bg-white p-6 rounded-2xl shadow-md text-center">
```

```
<h2 className="text-2xl font-semibold">{weather.name}</h2>
  <p className="text-lg">{weather.weather[0].description}</p>
  <p className="text-3xl font-bold">{weather.main.temp}°C</p>
  <p>Humidity: {weather.main.humidity}%</p>
  <p>Wind: {weather.wind.speed} m/s</p>
</div>
  )}
</div>
);
}
```

Output:

Result

- The Weather App is successfully created using React and Axios.
- The user can type a city name, and the app will fetch live weather data (temperature, humidity, wind speed, etc.) from the OpenWeatherMap API.
- If the city name is invalid, an error message **“City not found”** is displayed.

Ex.No:3.ii	Blog reader using
Date:	

Aim:

Algorithm:-

Program:-

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Simple Blog Reader</title>
    <style>
      body{font-family:system-ui, -apple-system, "Segoe UI", Roboto, Arial;max-width:900px;margin:24px
auto;padding:0 16px;color:#222}
      h1 {margin:0 0 12px}
      .post{border-bottom:1px solid #eee;padding:12px 0}
      .post h3 {margin:0}
      .meta {color:#666;font-size:.9rem;margin-bottom:8px}
      button {margin-top:8px;padding:6px 10px}
      .detail {background:#fafafa;padding:12px;border:1px solid #eee;margin-top:12px}
    </style>
  </head>
  <body>
    <h1>Tiny Blog Reader</h1>
    <div id="root"></div>

    <!-- React -->
    <script src="https://unpkg.com/react@18/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>

    <script>
      const { useState } = React;

      // sample posts embedded so no server needed
      const POSTS = [
        { id: '1', title: 'Welcome to Tiny Blog', author: 'Mohan', date: '2025-10-13', body: 'This is a tiny blog
built for teaching React. Click a post to read the full content.The command blog.html was not found, but
does exist in the current location. Windows PowerShell does not load commands from the current location
by default. If you trust this command, instead typePS E:\MERN\react\rblog> cd blog.html' },
        { id: '2', title: 'React Basics', author: 'Mohan', date: '2025-10-12', body: 'React uses components and
state. Here we keep things very small and simple.' },
        { id: '3', title: 'Fetch Later', author: 'Mohan', date: '2025-10-11', body: 'Later you can replace the
embedded posts with fetch() to an API or static JSON file.' }
      ];

      function PostList({ onOpen }) {
        return React.createElement('div', null,
          POSTS.map(p =>
            React.createElement('div', { key: p.id, className: 'post' },
              React.createElement('h3', null, p.title),
              React.createElement('div', { className: 'meta' }, `${p.author} • ${p.date}`),
              React.createElement('p', null, p.body.length > 120 ? p.body.slice(0,120) + '...' : p.body),
              React.createElement('button', { onClick: () => onOpen(p) }, 'Read')
            )
          )
        );
      }

      function PostDetail({ post, onBack }) {
        if (!post) return null;
        return React.createElement('div', { className: 'detail' },
```

```

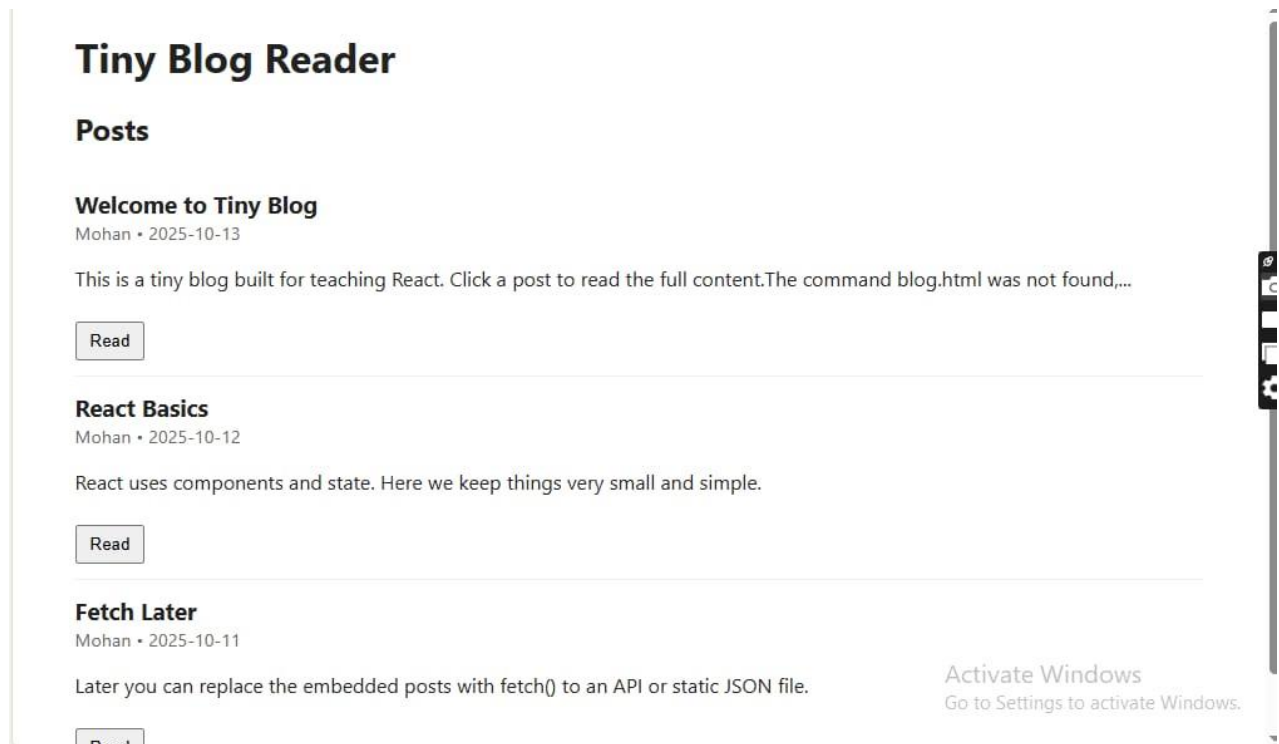
React.createElement('h2', null, post.title),
  React.createElement('div', { className: 'meta' }, `${post.author} • ${post.date}`),
  React.createElement('div', { style: { marginTop: 12 } }, post.body),
  React.createElement('div', null, React.createElement('button', { onClick: onBack }, '← Back to list'))
);
}

function App() {
  const [open, setOpen] = useState(null);
  return React.createElement('div', null,
    React.createElement('div', null,
      React.createElement('h2', null, 'Posts'),
      React.createElement(PostList, { onOpen: setOpen })
    ),
    React.createElement(PostDetail, { post: open, onBack: () => setOpen(null) })
  );
}

ReactDOM.createRoot(document.getElementById('root')).render(React.createElement(App));
</script>
</body>
</html>

```

Output :



RESULT :

The program displays blog titles and allows the reader to open and read each post.

Ex.No: 3.iii	React router
Date:	

Aim:

Algorithm:-

Program:-

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Simple React Hash Router (No react-router)</title>
  <meta name="viewport" content="width=device-width,initial-scale=1" />
  <style>
    body{font-family:Arial,Helvetica,sans-serif;margin:20px;background:#f7f7f7}
    .container{max-width:820px;margin:0 auto;background:#fff;padding:20px;border-radius:8px;box-
shadow:0 2px 6px rgba(0,0,0,0.06)}
    header h1 {margin:0 0 8px 0;font-size:22px}
    nav {margin-top:10px}
    nav a {margin-right:12px;text-decoration:none;color:#0077cc;padding:6px 8px;border-radius:4px}
    nav a.active {background:#0077cc;color:#fff}
    .page {margin-top:18px;padding:14px;border:1px solid #e6e6e6;border-radius:6px;background:#fff}
    footer {margin-top:18px;font-size:12px;color:#666}
  </style>
</head>
<body>
  <div class="container">
    <header>
      <h1>SIMPLE REACT HASH ROUTER</h1>
      <div>Works without react-router. Use links to navigate (hash-based).</div>
    </header>

    <div id="root"></div>

    <footer>
      Tip: Links use the hash (e.g. <code>#/about</code>) so this works from the file system or a server.
    </footer>
  </div>

  <!-- React UMD -->
  <script src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>

  <script>
    (function () {
      const e = React.createElement;
      const { useState, useEffect } = React;

      // Helper: read current route from location.hash
      function getRoute() {
        const h = window.location.hash || '#/';
        // remove leading '#' and ensure leading '/'
        const path = h.slice(1);
        return path === " ? '/' : path;
      }

      function Nav(props) {
        const route = props.route;
        function link(name, path) {
          const isActive = route === path;
```

```

const style = { textDecoration: 'none', color: isActive ? '#fff' : '#0077cc', background: isActive ? '#0077cc'
: 'transparent', padding: '6px 8px', borderRadius: '4px' };
    return e('a', { href: '#' + path, key: path, style }, name);
  }
  return e('nav', null, link('Home', '/'), link('About', '/about'), link('Contact', '/contact'));
}

function Home() {
  return e('div', { className: 'page' },
    e('h2', null, 'Home Page'),
    e('p', null, 'Welcome. This is the Home page in the simple hash-router demo.'),
    e('p', null, 'You can open this file directly in Chrome; the links use #/ routes.')
  );
}

function About() {
  return e('div', { className: 'page' },
    e('h2', null, 'About Page'),
    e('p', null, 'This example demonstrates routing implemented with hashchange and React.'),
    e('p', null, 'No bundler or extra installs required.')
  );
}

function Contact() {
  return e('div', { className: 'page' },
    e('h2', null, 'Contact Page'),
    e('p', null, 'Email: student@example.com'),
    e('p', null, 'Phone: +91-XXXXXXXXXXXX')
  );
}

function NotFound() {
  return e('div', { className: 'page' },
    e('h2', null, '404 — Page not found'),
    e('p', null, 'The requested page was not found. Use the navigation above.')
  );
}

function App() {
  const [route, setRoute] = useState(getRoute());

  useEffect(() => {
    function onHashChange() {
      setRoute(getRoute());
    }
    window.addEventListener('hashchange', onHashChange);
    // ensure route is synced if file opened w/o hash
    if (!window.location.hash) {
      window.location.hash = '#/';
    } else {
      setRoute(getRoute());
    }
    return () => window.removeEventListener('hashchange', onHashChange);
  }, []);

  let pageElem;

```

```

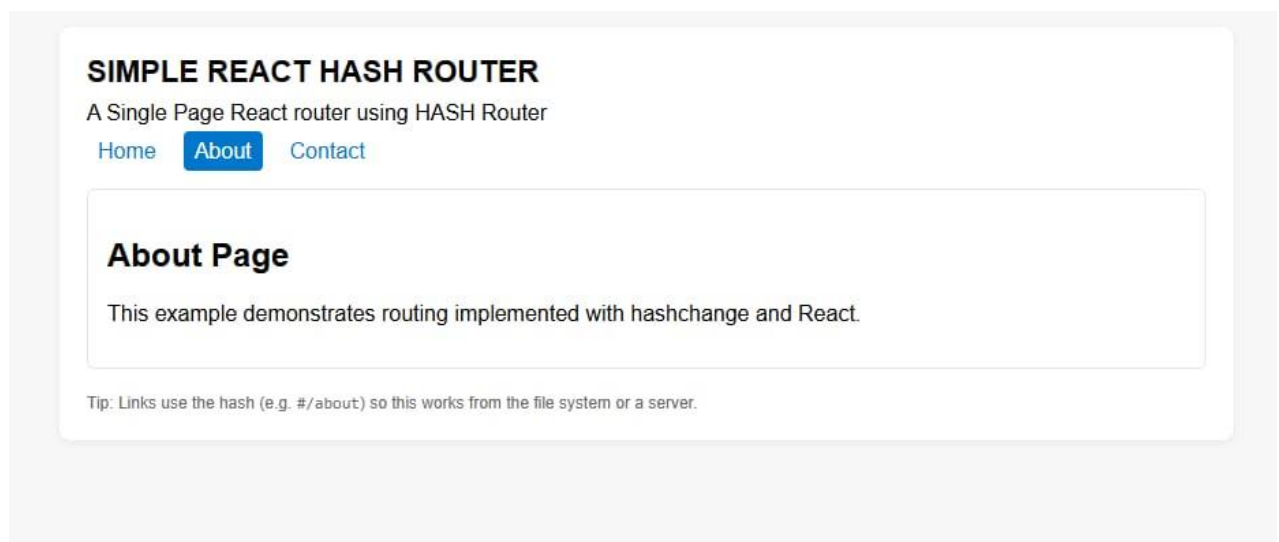
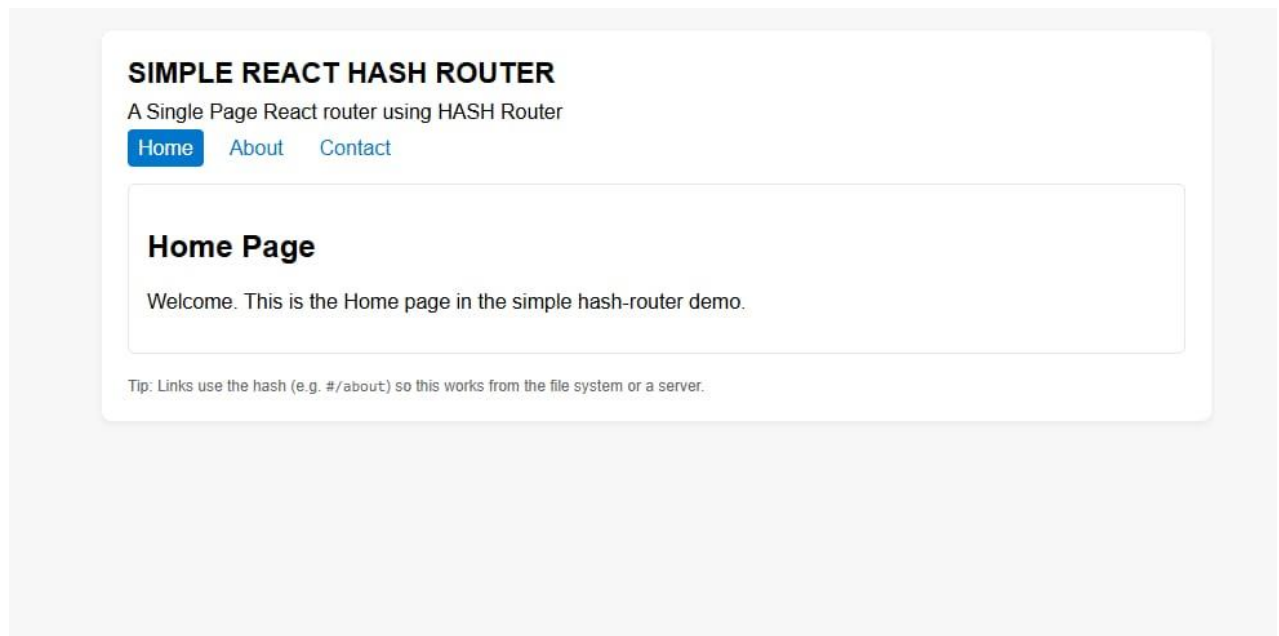
if (route === '/') pageElem = e(Home);
else if (route === '/about') pageElem = e(About);
else if (route === '/contact') pageElem = e(Contact);
else pageElem = e(NotFound);

return e('div', null, e(Nav, { route }), pageElem);
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(e(App));
})();
</script>
</body>
</html>

```

Output :



SIMPLE REACT HASH ROUTER

A Single Page React router using HASH Router

[Home](#)

[About](#)

[Contact](#)

Contact Page

Email: student@example.com

Phone: +91-XXXXXXXXXX

Tip: Links use the hash (e.g. #/about) so this works from the file system or a server.

RESULT :

Links navigate between Home, About, and Contact without page reload.

Ex.No: 4.i	Bookstore API using node.js
Date:	

Aim:

Algorithm:-

Program:-

```
const express = require('express');
const app = express();
app.use(express.json());

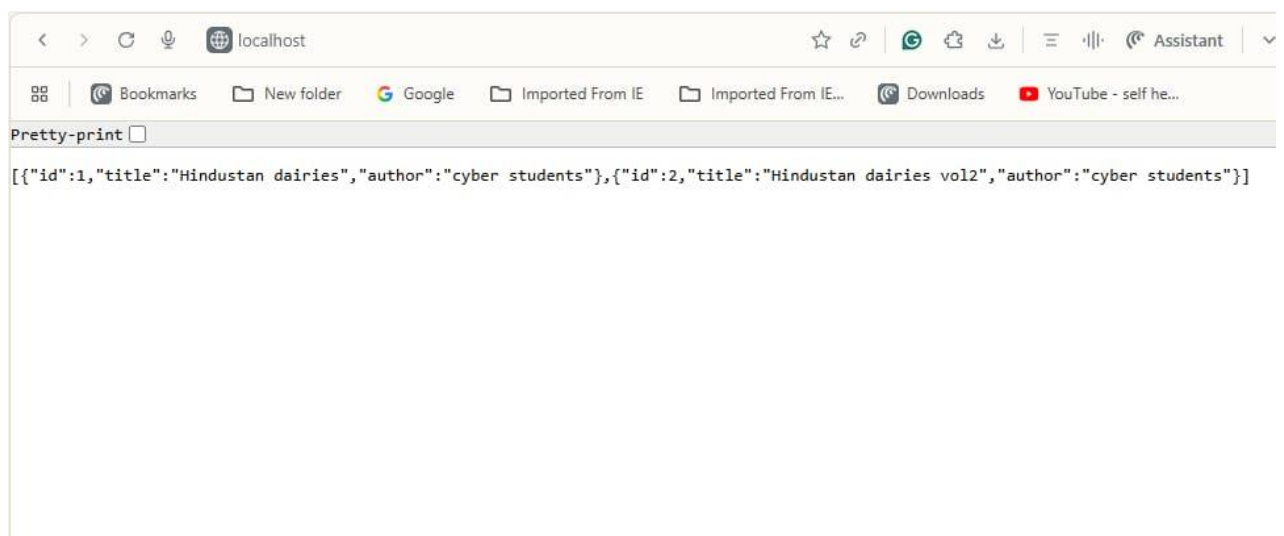
// temporary data (acts as our "database")
let books = [
  { id: 1, title: "Hindustan dairies", author: "cyber students" }
];

// show all books
app.get('/books', (req, res) => {
  res.json(books);
});

// add a new book
app.post('/books', (req, res) => {
  const newBook = {
    id: books.length + 1,
    title: req.body.title,
    author: req.body.author
  };
  books.push(newBook);
  res.status(201).json(newBook);
});

// start the server
app.listen(3000, () => {
  console.log('Bookstore API running at http://localhost:3000');
});
```

Output :



RESULT :

The API successfully allows users to view and add books using GET and POST requests.

Ex.No: 4.ii	Connecting node.js to MongoDB using mongoose
Date:	

Aim:

Algorithm:-

Program:-

```
// server.js — Smallest MongoDB connection example
const mongoose = require('mongoose');

// 1 Connect to local MongoDB (make sure mongod is running)
mongoose.connect('mongodb://127.0.0.1:27017/testdb2')
  .then(() => console.log('Connected to MongoDB successfully!'))
  .catch(err => console.error(' Connection error:', err));

// 2 Define a simple schema (like a table)
const userSchema = new mongoose.Schema({
  name: String,
  age: Number
});

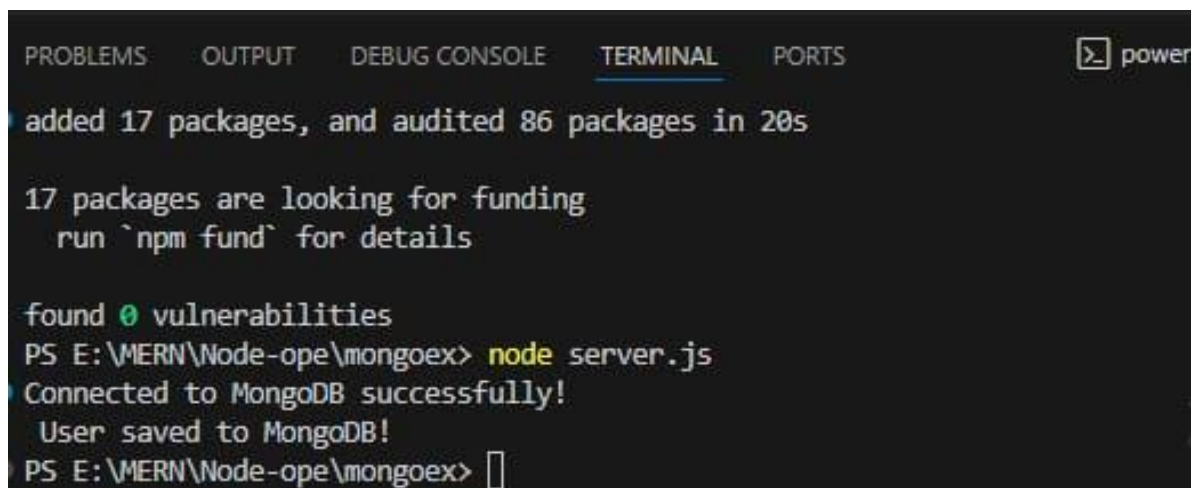
// 3 Create a model
const User = mongoose.model('User', userSchema);

// 4 Insert one sample record
async function runDemo() {
  const user = new User({ name: 'Cyber2', age: 25 });
  await user.save();
  console.log(' User saved to MongoDB!');
  mongoose.connection.close();
}

runDemo();

commands : npm init -y
npm install express
npm install mongoose
npm server.js
```

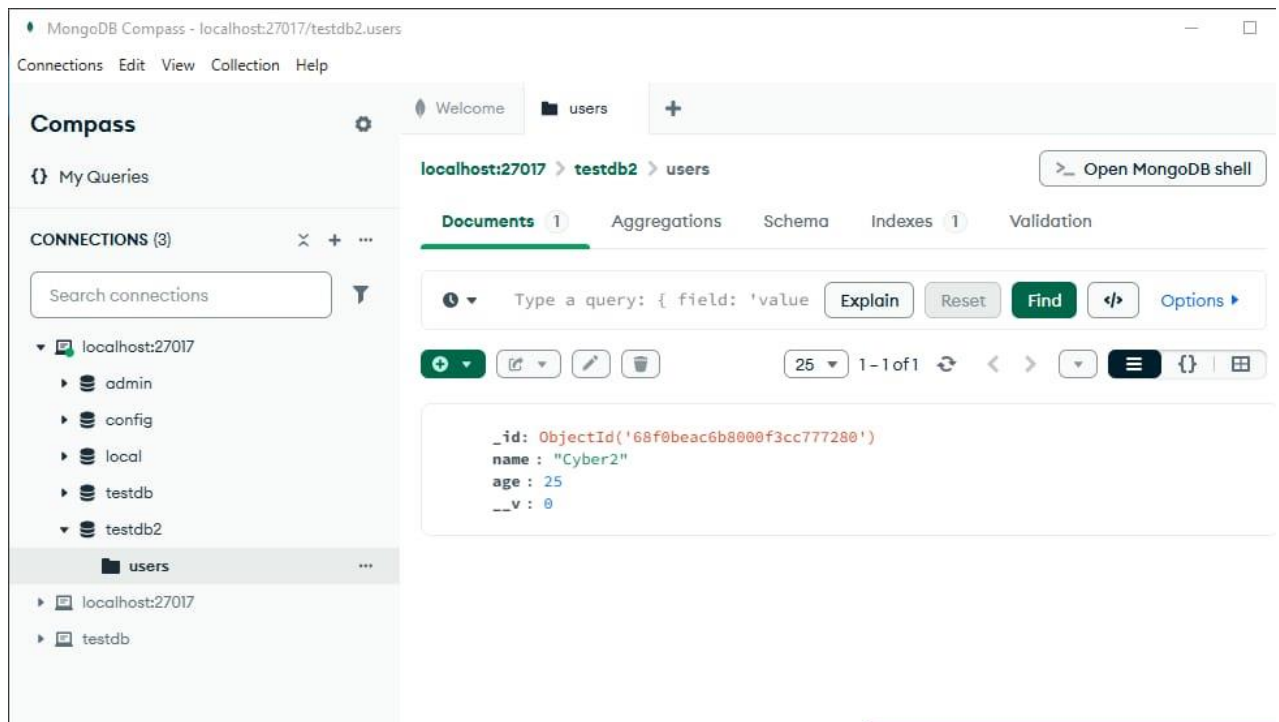
Output :



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  power
added 17 packages, and audited 86 packages in 20s

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

found 0 vulnerabilities
PS E:\MERN\Node-ope\mongoex> node server.js
Connected to MongoDB successfully!
  User saved to MongoDB!
PS E:\MERN\Node-ope\mongoex> 
```



RESULT:

Node.js connects to MongoDB successfully, and users can insert and retrieve data using Postman or MongoDB Compass.

Ex.No: 4.iii	JWT authentication using node.js
Date:	

Aim:

Algorithm:-

Program:-

```
// server.js — Simplest JWT Authentication Example
const express = require('express');
const jwt = require('jsonwebtoken');
const app = express();
app.use(express.json());

const SECRET_KEY = 'my_secret_key'; // only for demo
const USER = { username: 'admin', password: '1234' }; // hardcoded user

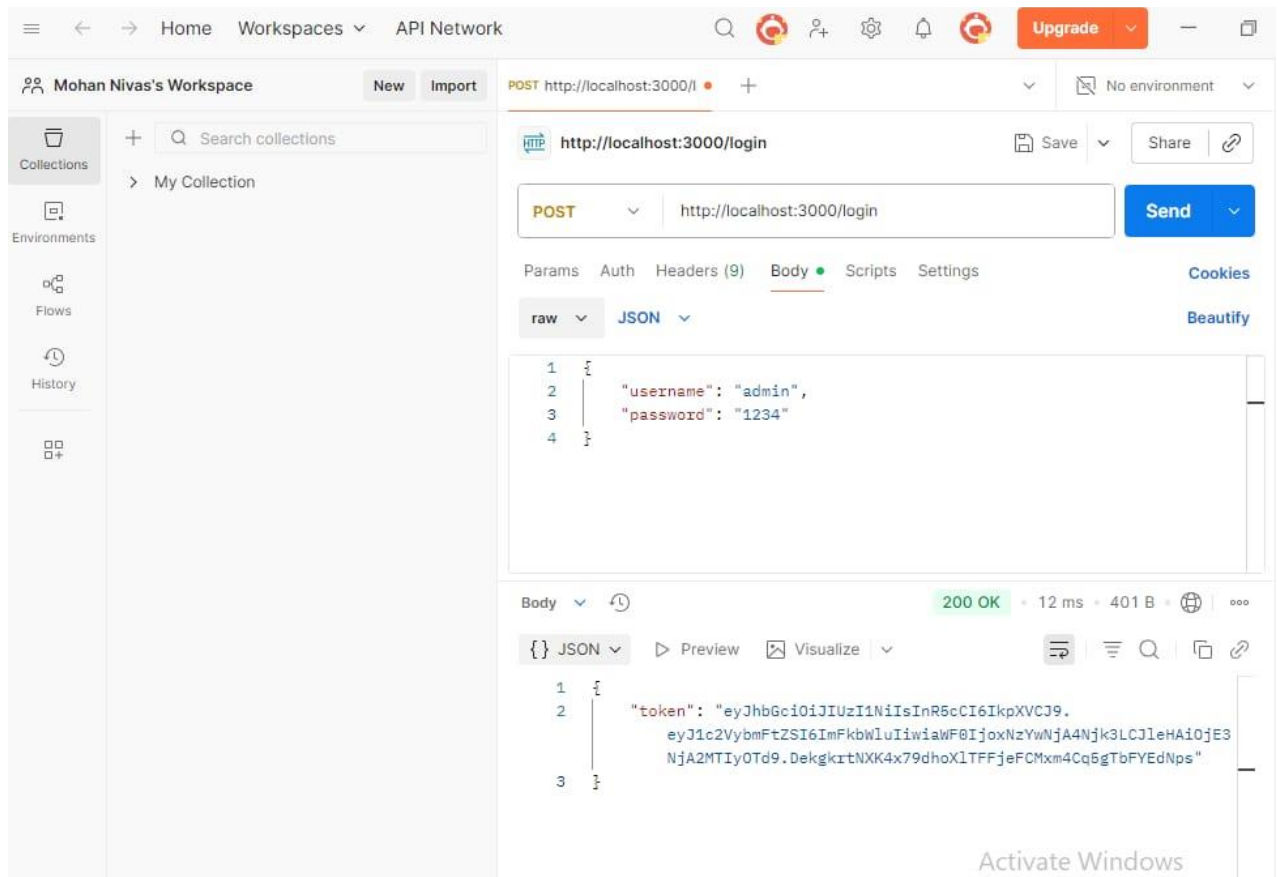
// 🟢 Login route
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  if (username === USER.username && password === USER.password) {
    const token = jwt.sign({ username }, SECRET_KEY, { expiresIn: '1h' });
    return res.json({ token });
  }
  res.status(401).json({ message: 'Invalid credentials' });
});

// 🟡 Protected route
app.get('/protected', (req, res) => {
  const authHeader = req.headers.authorization;
  if (!authHeader) return res.status(401).json({ message: 'Missing token' });

  const token = authHeader.split(' ')[1];
  try {
    const decoded = jwt.verify(token, SECRET_KEY);
    res.json({ message: `Welcome ${decoded.username}, you accessed a protected route!` });
  } catch {
    res.status(403).json({ message: 'Invalid or expired token' });
  }
});

app.listen(3000, () => console.log(' JWT demo running at http://localhost:3000'));
```

Output :



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

• added 13 packages, and audited 82 packages in 6s

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

found 0 vulnerabilities
PS E:\MIERN\Node-ope\jwt_new> node server.js
○ JWT demo running at http://localhost:3000
□
```

RESULT:

JWT authentication implemented successfully, allowing secure access to protected routes.

Ex.No: 5.i	React+ Express Integration
Date:	

Aim:

Algorithm:-

Program:-

```
server.js
// server.js — serve index.html in the same folder
const express = require('express');
const path = require('path');
const app = express();
const PORT = 3000;

// API example
app.get('/api/message', (req, res) => res.json({ message: 'Hello from Express API' }));

// serve index.html for the root and any unmatched routes
app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'index.html'));
});
app.use((req, res) => {
  res.sendFile(path.join(__dirname, 'index.html'));
});

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

index.html
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Part 1 — React + Express</title>
  </head>
  <body>
    <div id="root"></div>

    <!-- React from CDN -->
    <script src="https://unpkg.com/react@18/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>

    <script>
      const e = React.createElement;

      function App() {
        const [msg, setMsg] = React.useState('Loading...');
        React.useEffect(() => {
          fetch('/api/message')
            .then(r => r.json())
            .then(j => setMsg(j.message))
            .catch(() => setMsg('Error fetching message'));
        }, []);

        return e('div', { style: { fontFamily: 'sans-serif', padding: 20 } },
          e('h2', null, 'Part 1 — React + Express Integration'),
          e('p', null, 'API says:'),
          e('pre', null, msg)
        );
      }

      ReactDOM.createRoot(document.getElementById('root')).render(e(App));
    </script>
  </body>
```

</html>

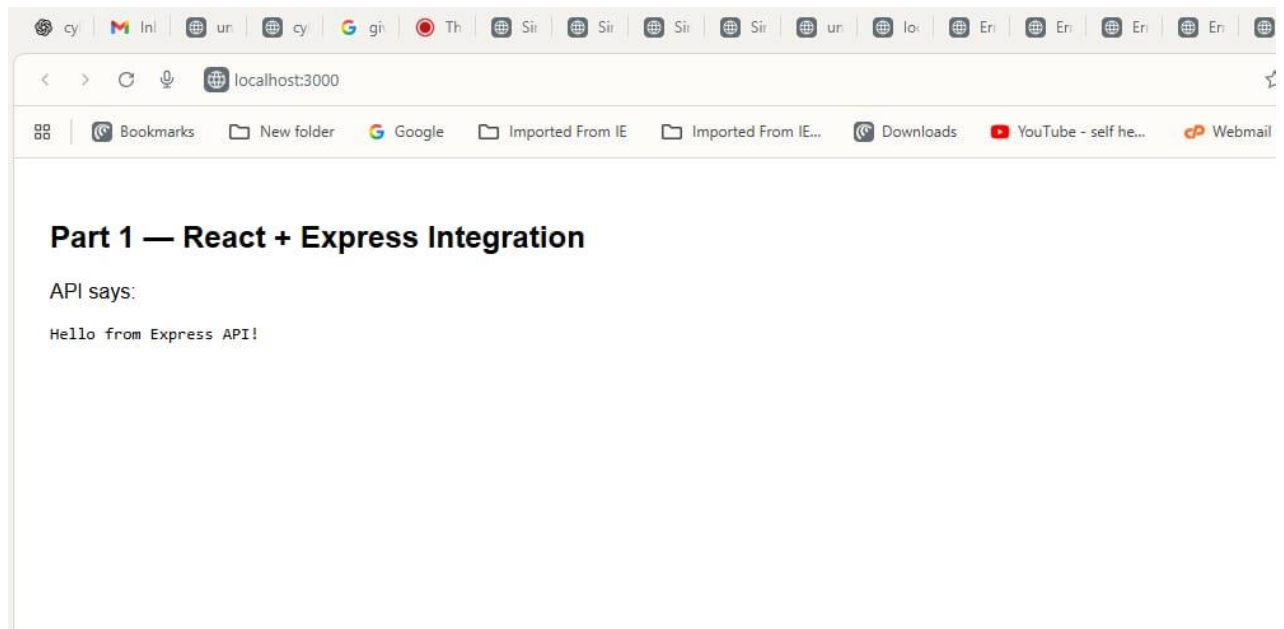
Output :

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

• added 1 package, and audited 70 packages in 5s

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

found 0 vulnerabilities
PS E:\MERN\Node-ope\fulls_int_reactexp\fls> node server.js
Server running: http://localhost:3000
█
```



RESULT:

The React frontend displays the message received from the Express backend.

Ex.No: 5.ii	Full Stack Task Manager
Date:	

Aim:

Algorithm:-

Program:-

Index.html

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Part 1 — React + Express</title>
  </head>
  <body>
    <div id="root"></div>

    <!-- React from CDN -->
    <script src="https://unpkg.com/react@18/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>

    <script>
      const e = React.createElement;

      function App() {
        const [msg, setMsg] = React.useState('Loading...');
        React.useEffect(() => {
          fetch('/api/message')
            .then(r => r.json())
            .then(j => setMsg(j.message))
            .catch(() => setMsg('Error fetching message'));
        }, []);

        return e('div', { style: { fontFamily: 'sans-serif', padding: 20 } },
          e('h2', null, 'Part 1 — React + Express Integration'),
          e('p', null, 'API says:'),
          e('pre', null, msg)
        );
      }

      ReactDOM.createRoot(document.getElementById('root')).render(e(App));
    </script>
  </body>
</html>
```

Server.js

```
// server.js — serve index.html in the same folder
const express = require('express');
const path = require('path');
const app = express();
const PORT = 3000;

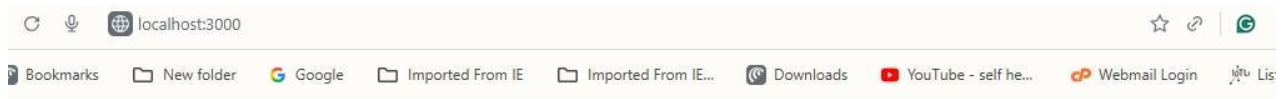
// API example
app.get('/api/message', (req, res) => res.json({ message: 'Hello from Express API' }));

// serve index.html for the root and any unmatched routes
app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'index.html'));
});
app.use((req, res) => {
  res.sendFile(path.join(__dirname, 'index.html'));
});
```

```
});
```

```
app.listen(PORT, () => console.log(`Server running: http://localhost:${PORT}`))
```

Output :



Simple Task Manager

- ☐ Example task



Simple Task Manager

- ☒ sample task 1
- ☒ sample task 2
- ☐ sample task 3

RESULT :

The Task Manager UI loads tasks, allows adding new tasks, toggling completion, and deleting tasks via the backend API.

Ex.No: 5.iii	Development Demo (GitHub+ Render.com)
Date:	

Aim:

Algorithm:-

Program:-

Server.js

```
const express = require('express');
const path = require('path');
const app = express();
const PORT = process.env.PORT || 3000;

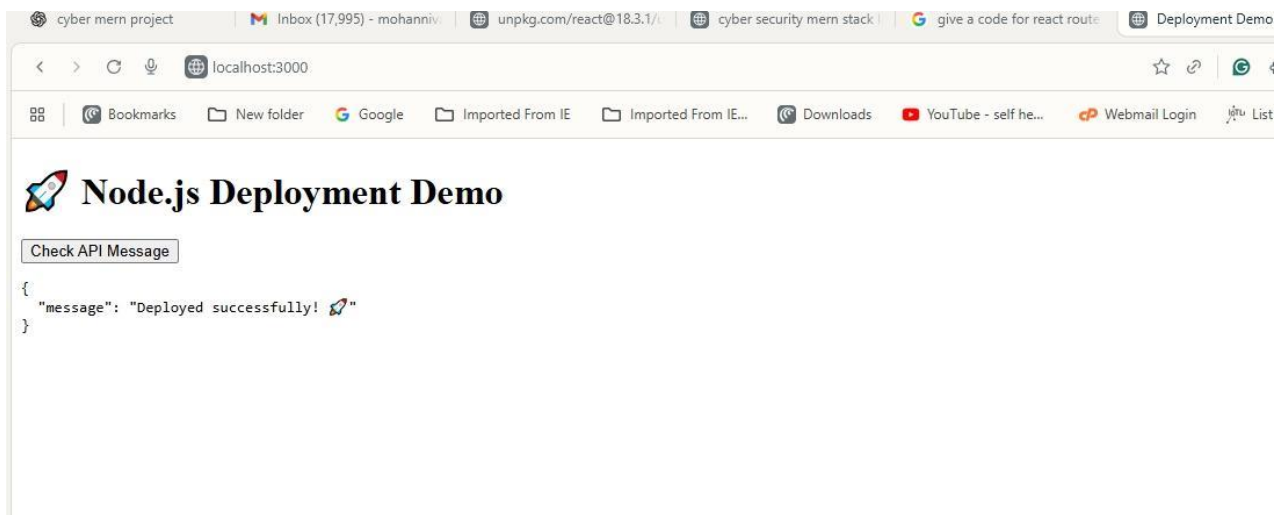
app.get('/', (req, res) => res.sendFile(path.join(__dirname, 'index.html')));
app.get('/api/message', (req, res) => res.json({ message: 'Deployed successfully! 🚀' }));

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

index.html

```
<!doctype html><html><head><meta charset="utf-8"/><title>Deployment Demo</title></head>
<body>
  <h1>🚀 Node.js Deployment Demo</h1>
  <button id="btn">Check API Message</button>
  <pre id="output"></pre>
  <script>
    document.getElementById("btn").onclick = async () => {
      const res = await fetch("/api/message");
      const data = await res.json();
      document.getElementById("output").textContent = JSON.stringify(data, null, 2);
    };
  </script>
</body></html>
```

Output :



RESULT :

After deploying, the Render URL should load the page and the Check API button should fetch JSON from `/api/message`.

Ex.No: 6	Task Manager Development
Date:	

Aim:

Algorithm:-

Program:-

Index.html

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Part 1 — React + Express</title>
  </head>
  <body>
    <div id="root"></div>

    <!-- React from CDN -->
    <script src="https://unpkg.com/react@18/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>

    <script>
      const e = React.createElement;

      function App() {
        const [msg, setMsg] = React.useState('Loading...');
        React.useEffect(() => {
          fetch('/api/message')
            .then(r => r.json())
            .then(j => setMsg(j.message))
            .catch(() => setMsg('Error fetching message'));
        }, []);

        return e('div', { style: { fontFamily: 'sans-serif', padding: 20 } },
          e('h2', null, 'Part 1 — React + Express Integration'),
          e('p', null, 'API says:'),
          e('pre', null, msg)
        );
      }

      ReactDOM.createRoot(document.getElementById('root')).render(e(App));
    </script>
  </body>
</html>
```

Server.js

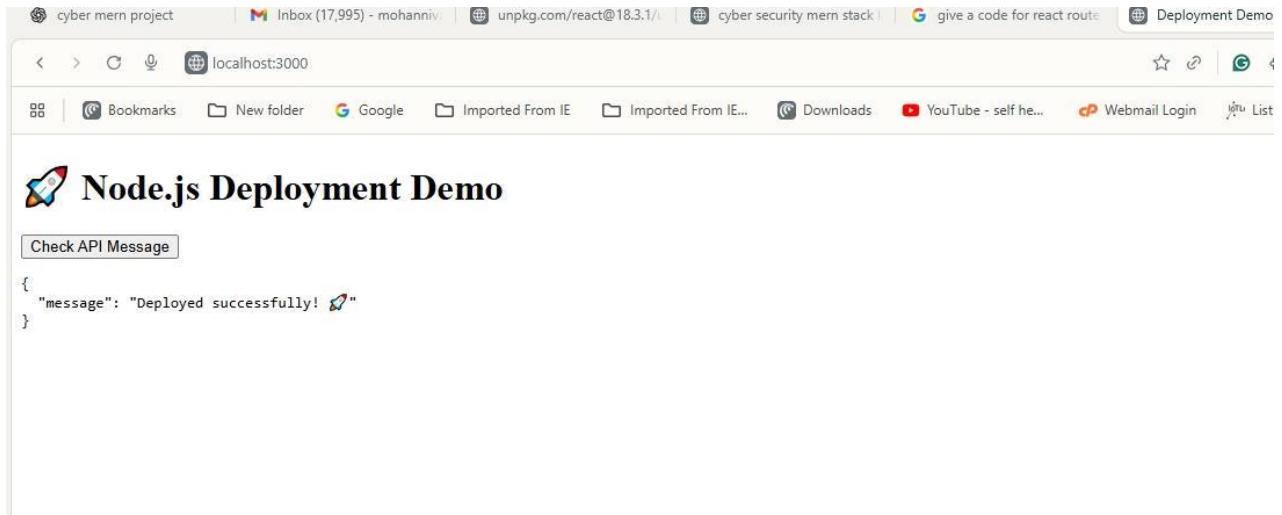
```
// server.js — serve index.html in the same folder
const express = require('express');
const path = require('path');
const app = express();
const PORT = 3000;

// API example
app.get('/api/message', (req, res) => res.json({ message: 'Hello from Express API' }));

// serve index.html for the root and any unmatched routes
app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'index.html'));
});
app.use((req, res) => {
  res.sendFile(path.join(__dirname, 'index.html'));
});
```

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

Output :



RESULT :

The Task Manager should run on Render with a live URL. All CRUD operations on tasks should work as they did locally.

