

## Day 5 – React Hooks, Form Handling, Routing

Date: 27-10-2025

**Objective:** Learn how to manage data and side effects using React Hooks (useState, useEffect), navigate between pages using React Router, handle forms dynamically, and connect React with APIs using fetch.

### 1. REACT HOOKS OVERVIEW

Hooks are special functions that allow functional components to use state and lifecycle features that were previously available only in class components.

#### Commonly Used Hooks:

Hook	Purpose
useState()	Manage component state
useEffect()	Handle side effects (API calls, data loading)
useContext()	Manage global data
useRef()	Access or modify DOM elements directly

### 2. USESTATE HOOK

useState is used to store and update dynamic data in a component.

#### Example:

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

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

  return (
    <div>
      <h2>Count: {count}</h2>
      <button onClick={() => setCount(count + 1)}>Increase</button>
      <button onClick={() => setCount(count - 1)}>Decrease</button>
    </div>
  );
}
```

#### Explanation:

- count is a state variable.

- `setCount()` updates its value.
- Changing the state automatically re-renders the component.

### 3. USEEFFECT HOOK

`useEffect` is used to perform **side effects** like:

- Fetching data from APIs
- Setting up timers
- Updating the document title

#### Example:

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

```
function UserList() {  
  const [users, setUsers] = useState([]);  
  
  useEffect(() => {  
    fetch("https://jsonplaceholder.typicode.com/users")  
      .then(response => response.json())  
      .then(data => setUsers(data));  
  }, []); // Empty dependency → runs only once after render  
  
  return (  
    <div>  
      <h2>User List</h2>  
      <ul>  
        {users.slice(0, 5).map(user => (  
          <li key={user.id}>{user.name}</li>  
        ))}  
      </ul>  
    </div>  
  )  
}
```

```
);  
}
```

**Explanation:**

- The effect runs once when the component loads.
- Fetches data from an external API.
- Stores the result in users state and displays it.

#### 4. REACT ROUTER (NAVIGATION BETWEEN PAGES)

**React Router** enables navigation between multiple views or components within a single-page app (SPA).

**Installation**

```
npm install react-router-dom
```

**Basic Setup**

```
// App.js  
  
import { BrowserRouter as Router, Routes, Route, Link } from "react-router-dom";  
  
import Home from "./Home";  
  
import About from "./About";  
  
import Contact from "./Contact";  
  
  
function App() {  
  return (  
    <Router>  
      <nav>  
        <Link to="/">Home</Link> |  
        <Link to="/about">About</Link> |  
        <Link to="/contact">Contact</Link>  
      </nav>  
  
      <Routes>  
        <Route path="/" element={<Home />} />  

```

```

    <Route path="/about" element={<About />} />

    <Route path="/contact" element={<Contact />} />

  </Routes>

</Router>

);
}

```

#### Explanation:

- <Router> wraps the whole app.
- <Routes> defines different page paths.
- <Link> enables navigation without reloading the page.

## 5. REACT FORM HANDLING

React manages form data through **controlled components**, where the form elements are linked to component state.

#### Example:

```

import React, { useState } from "react";

function StudentForm() {

  const [formData, setFormData] = useState({ name: "", course: "" });

  const handleChange = (e) => {

    setFormData({ ...formData, [e.target.name]: e.target.value });

  };

  const handleSubmit = (e) => {

    e.preventDefault();

    alert(` Student: ${formData.name}, Course: ${formData.course} `);

  };

  return (

```

```

<form onSubmit={handleSubmit}>
  <input
    type="text"
    name="name"
    placeholder="Enter Name"
    value={formData.name}
    onChange={handleChange}
  />
  <input
    type="text"
    name="course"
    placeholder="Enter Course"
    value={formData.course}
    onChange={handleChange}
  />
  <button type="submit">Submit</button>
</form>
);
}

```

### Key Points:

- value attribute binds input to state.
- onChange updates state dynamically.
- onSubmit processes the form data.

## 6. CALLING APIS IN REACT USING FETCH

React can send HTTP requests to backend APIs (for example, an ASP.NET Core Web API).

### GET Request Example:

```

fetch("https://jsonplaceholder.typicode.com/posts")
  .then(res => res.json())

```

```
.then(data => console.log(data))  
.catch(err => console.error("Error fetching data:", err));
```

### **POST Request Example (Sending Data):**

```
fetch("https://jsonplaceholder.typicode.com/posts", {  
  method: "POST",  
  headers: { "Content-Type": "application/json" },  
  body: JSON.stringify({ title: "Hello", body: "React + .NET API", userId: 1 })  
})  
  
.then(res => res.json())  
.then(data => console.log("Data submitted:", data))  
.catch(err => console.error("Error:", err));
```

### **When using with .NET APIs:**

- Replace URL with your backend endpoint.
- Set correct headers and handle authentication if required.

## **7. EXAMPLE: REACT COMPONENT WITH API + FORM + USEEFFECT**

```
import React, { useState, useEffect } from "react";  
  
function Students() {  
  const [students, setStudents] = useState([]);  
  const [newStudent, setNewStudent] = useState("");  
  
  useEffect(() => {  
    fetch("https://jsonplaceholder.typicode.com/users")  
      .then(res => res.json())  
      .then(data => setStudents(data.slice(0, 5)));  
  }, []);  
  
  const addStudent = () => {
```

```

    if (newStudent.trim() === "") return;
    setStudents([...students, { name: newStudent }]);
    setNewStudent("");
  };

  return (
    <div>
      <h2>Student List</h2>
      <ul>
        {students.map((s, i) => (
          <li key={i}>{s.name}</li>
        ))}
      </ul>

      <input
        type="text"
        placeholder="Add new student"
        value={newStudent}
        onChange={(e) => setNewStudent(e.target.value)}
      />

      <button onClick={addStudent}>Add</button>
    </div>
  );
}

export default Students;

```

## BEST PRACTICES

- Keep API URLs in a separate config file.
- Use `useEffect` carefully to prevent infinite loops.

- Validate form input before sending.
- Use async/await for cleaner asynchronous code.
- Use React Router for organized navigation.

## **MINI PRACTICE TASK**

### **Objective:**

Integrate React Hooks, forms, and API communication.

### **Tasks:**

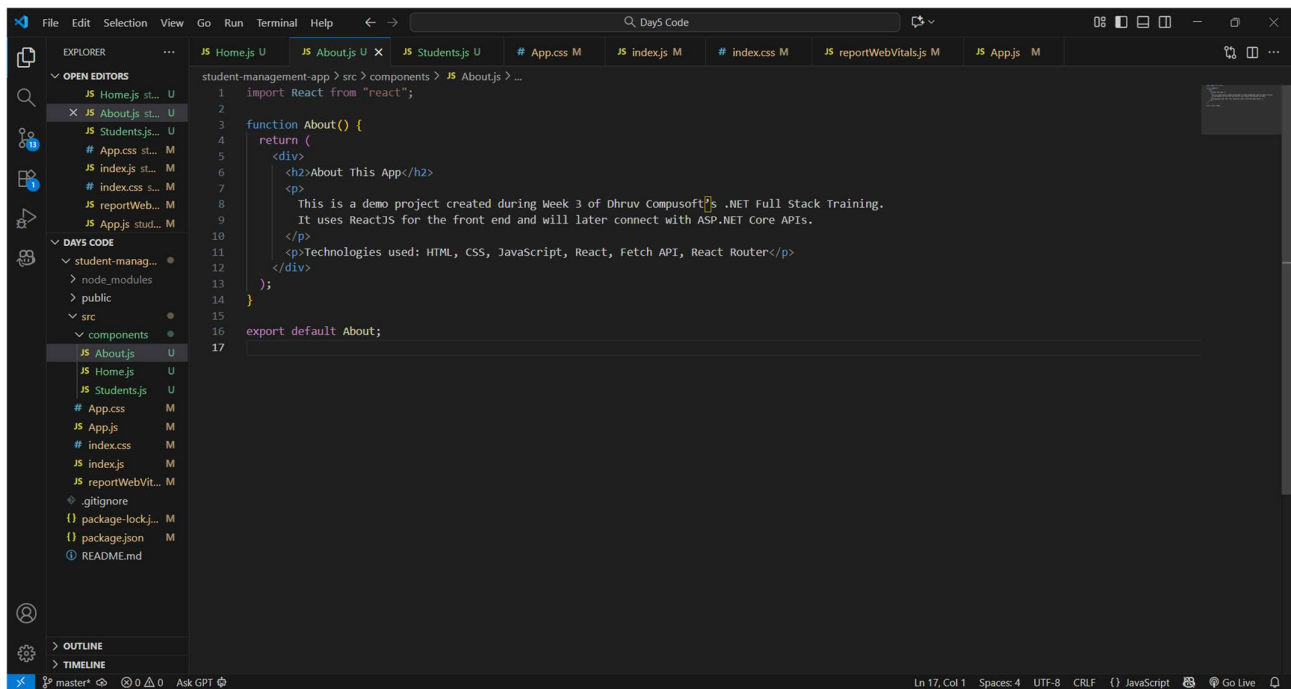
1. Create a multi-page app using React Router (Home, Students, About).
2. In Students page:
  - Use a form to add new students.
  - Fetch student data using API (or local JSON).
  - Display students dynamically.
3. Use useState for form and list management.
4. Use useEffect for fetching data when component loads.

### **Files:**

- App.js
- Students.js
- About.js
- Home.js



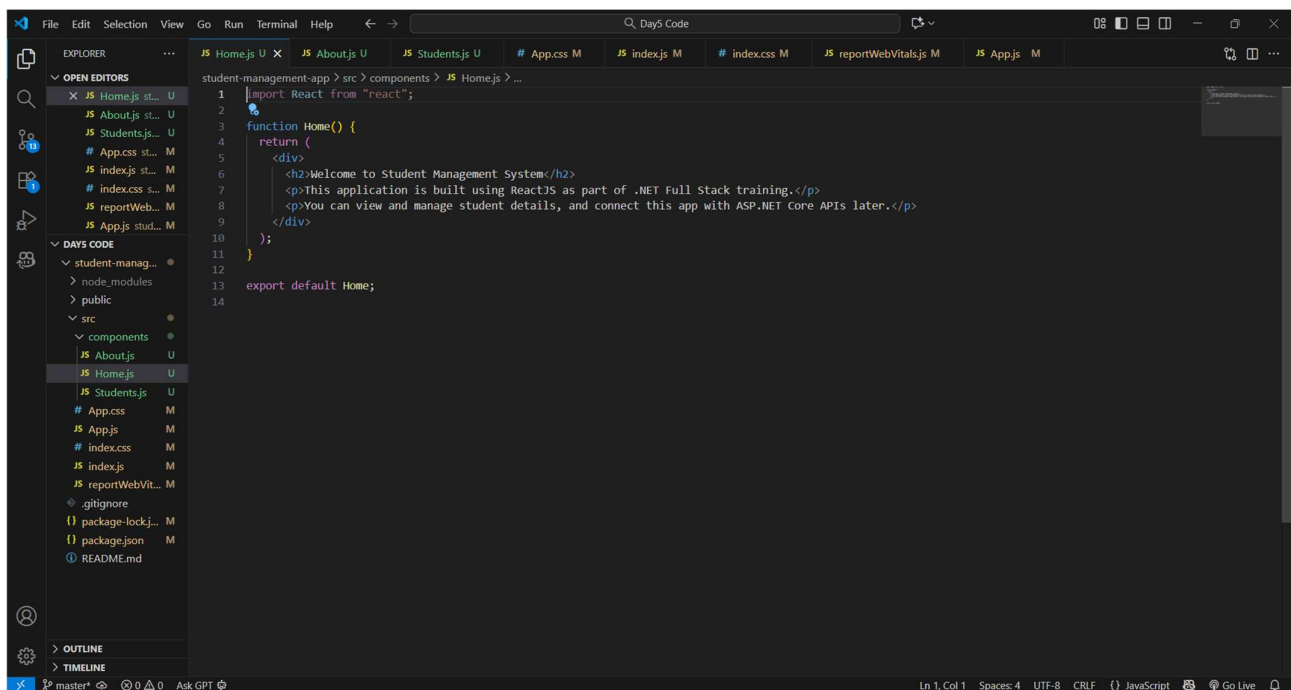
## Snapshots:



The screenshot shows the Visual Studio Code editor with the 'About.js' file open. The Explorer sidebar on the left shows the project structure, including 'src' > 'components' > 'About.js'. The main editor area displays the following code:

```
1 import React from "react";
2
3 function About() {
4   return (
5     <div>
6       <h2>About This App</h2>
7       <p>
8         This is a demo project created during Week 3 of Dhruv Compusoft's .NET Full Stack Training.
9         It uses ReactJS for the front end and will later connect with ASP.NET Core APIs.
10      </p>
11      <p>Technologies used: HTML, CSS, JavaScript, React, Fetch API, React Router</p>
12    </div>
13  );
14}
15
16 export default About;
```

Code : About.js



The screenshot shows the Visual Studio Code editor with the 'Home.js' file open. The Explorer sidebar on the left shows the project structure, including 'src' > 'components' > 'Home.js'. The main editor area displays the following code:

```
1 import React from "react";
2
3 function Home() {
4   return (
5     <div>
6       <h2>Welcome to Student Management System</h2>
7       <p>This application is built using ReactJS as part of .NET Full Stack training.</p>
8       <p>You can view and manage student details, and connect this app with ASP.NET Core APIs later.</p>
9     </div>
10  );
11}
12
13 export default Home;
```

Code : Home.js

```
1 import React, { useState, useEffect } from "react";
2
3 function Students() {
4   const [students, setStudents] = useState([]);
5   const [newStudent, setNewStudent] = useState("");
6   const [course, setCourse] = useState("");
7
8   useEffect(() => {
9     // Fetching dummy data (you can replace with your ASP.NET API)
10    fetch("https://jsonplaceholder.typicode.com/users")
11      .then((res) => res.json())
12      .then((data) => setStudents(data.slice(0, 5)))
13      .catch((err) => console.error("Error fetching data:", err));
14  }, []);
15
16  const handleAddStudent = (e) => {
17    e.preventDefault();
18    if (!newStudent || !course) return alert("Please fill all field");
19
20    const newEntry = { id: students.length + 1, name: newStudent, course };
21    setStudents([...students, newEntry]);
22    setNewStudent("");
23    setCourse("");
24  };
25
26  return (
27    <div>
28      <h2>Student List</h2>
29
30      <form onSubmit={handleAddStudent} className="student-form">
31        <input
32          type="text"
33          placeholder="Enter Student Name"
34          value={newStudent}
35          onChange={(e) => setNewStudent(e.target.value)}
36        />
37        <input
38          type="text"
39          placeholder="Enter Course"
40          value={course}
41          onChange={(e) => setCourse(e.target.value)}
42        />
43        <button type="submit">Add Student</button>
44      </form>
45
46      <ul className="student-list">
47        {students.map((student) => (
48          <li key={student.id}>
49            <strong>{student.name}</strong> <span>{student.course || "C"}</span>
50          </li>
51        ))}
52      </ul>
53    </div>
54  );
55
56 export default Students;
```

Code : Students.js

```
1 import React from "react";
2 import { BrowserRouter as Router, Routes, Route, Link } from "react-router-dom";
3 import Home from "./components/Home";
4 import About from "./components/About";
5 import Students from "./components/Students";
6 import "./App.css";
7
8 function App() {
9   return (
10    <Router>
11      <div className="container">
12        <nav className="navbar">
13          <h2>Student Management System</h2>
14          <ul>
15            <li><Link to="/">Home</Link></li>
16            <li><Link to="/students">Students</Link></li>
17            <li><Link to="/about">About</Link></li>
18          </ul>
19        </nav>
20
21        <div className="content">
22          <Routes>
23            <Route path="/" element={<Home />} />
24            <Route path="/students" element={<Students />} />
25            <Route path="/about" element={<About />} />
26          </Routes>
27        </div>
28      </div>
29    </Router>
30  );
31
32 export default App;
```

Code : App.js

The screenshot shows the VS Code editor interface. The Explorer panel on the left displays the project structure, including files like Home.js, About.js, Students.js, App.css, App.js, index.css, and index.js. The index.js file is selected and open in the editor. The code in index.js is as follows:

```
1 import React from "react";
2 import ReactDOM from "react-dom/client";
3 import "../index.css";
4 import App from "../App";
5 import reportWebVitals from "../reportWebVitals";
6
7 const root = ReactDOM.createRoot(document.getElementById("root"));
8 root.render(
9   <React.StrictMode>
10     <App />
11   </React.StrictMode>
12 );
13
14 reportWebVitals();
```

Code : index.js

The screenshot shows the VS Code editor interface with the terminal panel open at the bottom. The terminal displays the commands and output for running the application. The commands entered are:

```
PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty_NET_Training\Week3_Frontend\Day-5 (27-10-2025)\Day5 Code> cd student-management-app
PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty_NET_Training\Week3_Frontend\Day-5 (27-10-2025)\Day5 Code\student-management-app> npm start
```

The output shows the following messages:

```
> student-management-app@0.1.0 start
> react-scripts start

(node:29476) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
(Use 'node --trace-deprecation ...' to show where the warning was created)
(node:29476) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
Starting the development server...
Compiled successfully!

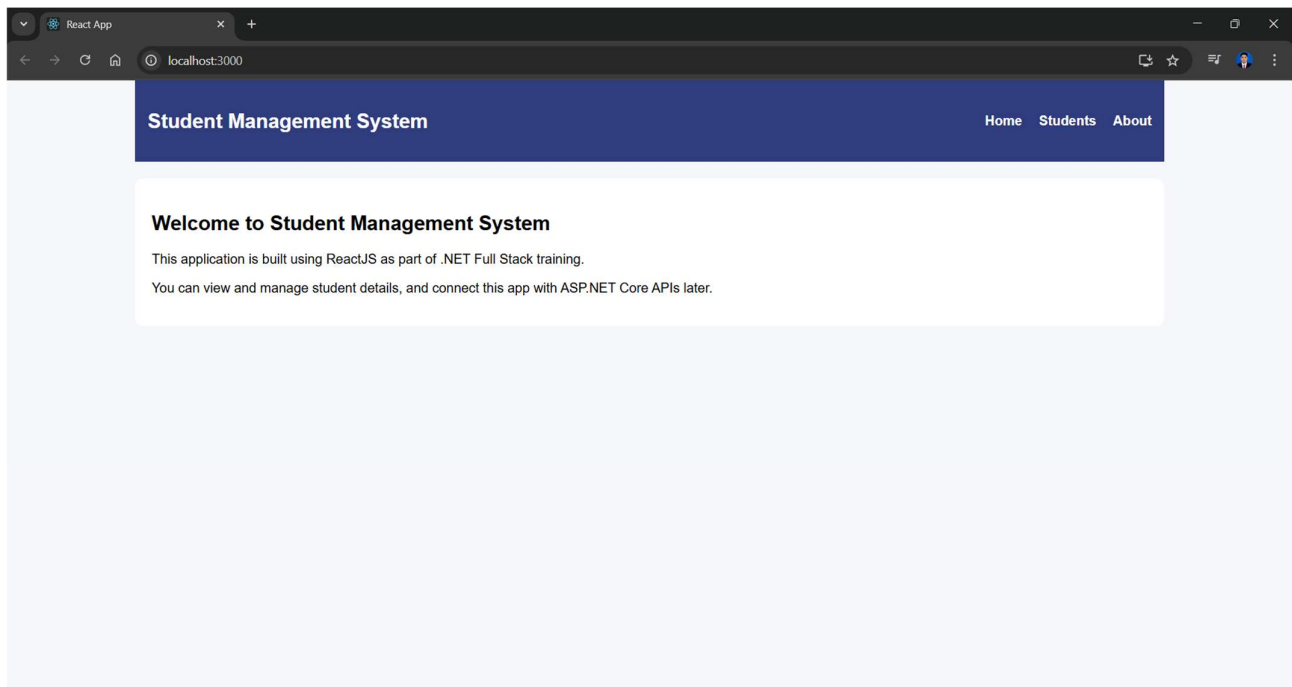
You can now view student-management-app in the browser.

Local:      http://localhost:3000
On Your Network: http://10.193.12.199:3000

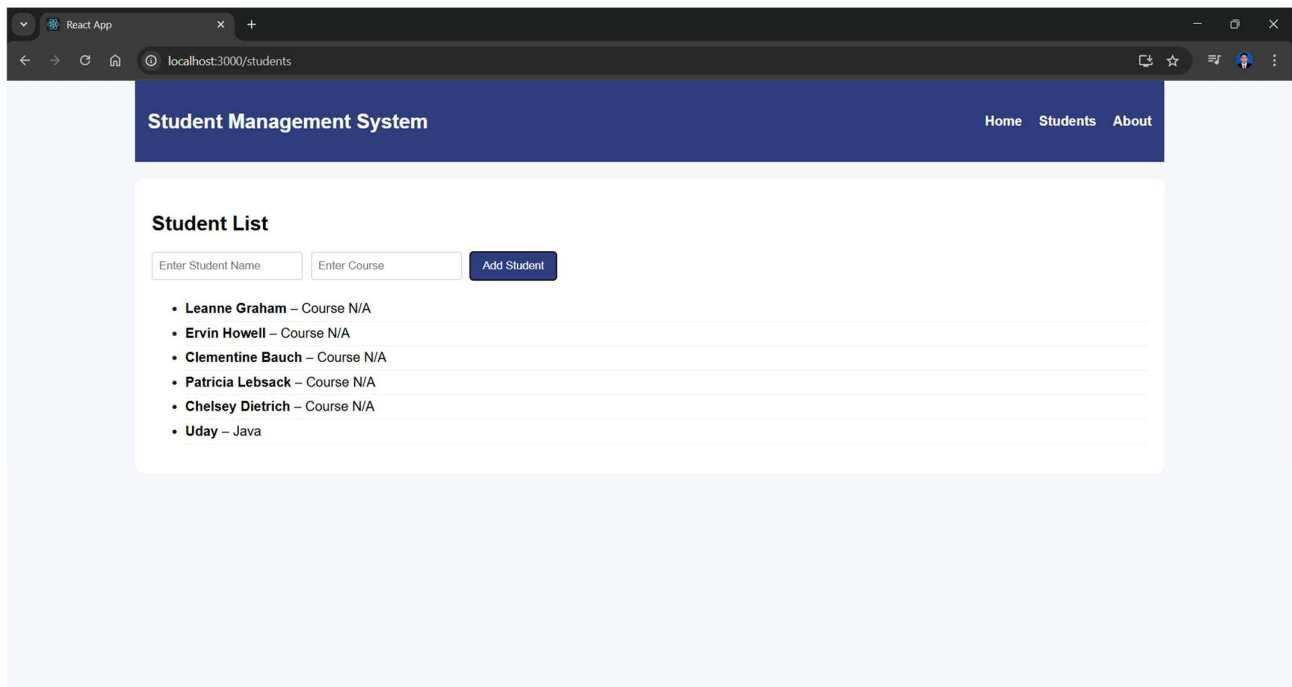
Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

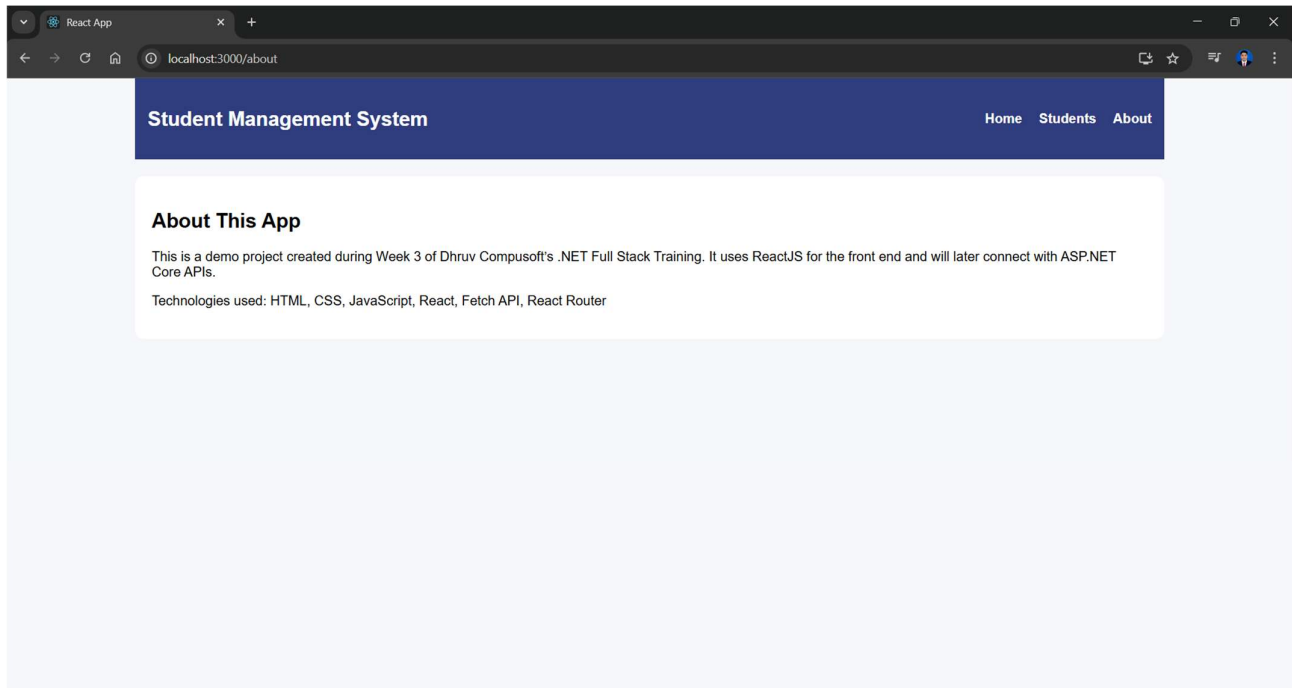
Code : Terminal Commands



Output : Home



Output: Students



Output : About