**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

A screenshot of a computer

AI-generated content may be incorrect.**Snapshots:**

Code : About.js

A screenshot of a computer

AI-generated content may be incorrect.

Code : Home.js

A screenshot of a computer program

AI-generated content may be incorrect.

Code : Students.js

A screen shot of a computer

AI-generated content may be incorrect.

Code : App.js

A screen shot of a computer

AI-generated content may be incorrect.

Code : index.js

A screenshot of a computer program

AI-generated content may be incorrect.

Code : Terminal Commands

A screenshot of a computer

AI-generated content may be incorrect.

Output : Home

A screenshot of a computer

AI-generated content may be incorrect.

Output: Students

A screenshot of a computer

AI-generated content may be incorrect.

Output : About