9. React Router DOM

What

React Router DOM is a library used to add routing (navigation between pages) in a React application. Instead of manually managing page changes, React Router makes it easy to define routes and navigate between them.

Why

Routing is essential for creating multi-page applications. With React Router, you can:

- Navigate between different components (pages).

- Create dynamic routes with parameters.

- Implement navigation without reloading the page.

How

We'll set up React Router using the modern `createBrowserRouter` and `RouterProvider` methods.

Step 1: Install React Router DOM

1. Install the React Router DOM package:

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

Step 2: Set Up Routes Using `createBrowserRouter`

1. Create Route Components: Create the following three files in the `src` folder:

- `Home.jsx` (for the homepage)
- `About.jsx` (for the about page)
- `Contact.jsx` (for the contact page)

2. Home.jsx:

```
import React from 'react';
function Home() {
  return <h2>Welcome to the Home Page</h2>;
}
export default Home;
```

3. About.jsx:

```
import React from 'react';
function About() {
  return <h2>About Us</h2>;
}
export default About;
```

4. Contact.jsx:

```
import React from 'react';
function Contact() {
  return <h2>Contact Us</h2>;
}
export default Contact;
```

Step 3: Create Routes and Set Up `RouterProvider`

Open `App.jsx` and update it as follows:

```
import React from 'react';
import { createBrowserRouter, RouterProvider } from 'react-router-dom';
import Home from './Home';
import About from './About';
import Contact from './Contact';
const router = createBrowserRouter([
  {
    path: '/',
    element: <Home />,
  },
  {
    path: '/about',
    element: <About />,
  },
  {
    path: '/contact',
    element: <Contact />,
  },
]);
function App() {
  return <RouterProvider router={router} />;
}

export default App;
```

- `createBrowserRouter`: Defines the routes and maps paths (`/`, `/about`, `/contact`) to their respective components.

- `RouterProvider`: Wraps the app and uses the defined router.

Step 4: Add Navigation Links

1. Update `Home.jsx` to include navigation links:

```
import React from 'react';
import { Link } from 'react-router-dom';
function Home() {
  return (
    <div>
      <h2>Welcome to the Home Page</h2>
      <nav>
        <Link to="/about">About</Link> | <Link to="/contact">Contact</Link>
      </nav>
    </div>
  );
}
export default Home;
```

Now, you can navigate between pages by clicking the links.

10. `useRef` vs `useState`

What

- `useState` is used to manage state in a React component. When the state updates, the component re-renders.

- `useRef` is used to store a mutable value that does not trigger a re-render when it changes. It's often used to access DOM elements directly.

Why

Knowing when to use `useRef` or `useState` helps optimize performance and ensures clean code. Use `useState` for values that need to trigger a UI update, and `useRef` for values that don't.

Example: Timer with `useState` and `useRef`

1. Create Timer.jsx:

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

function Timer() {

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

  const timerRef = useRef(null);


  const startTimer = () => {

    if (!timerRef.current) {

      timerRef.current = setInterval(() => {

        setCount((prev) => prev + 1);

      }, 1000);

    }

  };


  const stopTimer = () => {

    clearInterval(timerRef.current);

    timerRef.current = null;

  };
  return (
    <div>
      <h2>Timer: {count}s</h2>
      <button onClick={startTimer}>Start</button>
      <button onClick={stopTimer}>Stop</button>
    </div>
  );
}

export default Timer;
```

- `useState`: Tracks the count and re-renders the component when the count updates.
- `useRef`: Holds the timer ID without causing re-renders.

11. Data Fetching with Axios

What

Axios is a promise-based library for making HTTP requests. It simplifies fetching data from APIs and handling errors.

Why

Fetching data is essential for dynamic applications. Axios is preferred for its clean syntax, error handling, and the ability to cancel requests.

Example: Fetching Data with Axios

1. Install Axios:

```
npm install axios
```

2. Create FetchData.jsx:

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

import axios from 'axios';


function FetchData() {

  const [users, setUsers] = useState([]);

  const [loading, setLoading] = useState(true);

  const [error, setError] = useState(null);


  useEffect(() => {

    const fetchData = async () => {

      try {

        const response = await axios.get('https://jsonplaceholder.typicode.com/users');

        setUsers(response.data);

        setLoading(false);

      } catch (err) {

        setError('Failed to fetch data');

        setLoading(false)}}

    fetchData();

  }, []);
  if (loading) return <p>Loading...</p>;
  if (error) return <p>{error}</p>;
  return (
    <div>
      <h2>Users</h2>
      <ul>
        {users.map((user) => (
          <li key={user.id}>{user.name}</li>
        ))}
      </ul>
    </div>
  );
}
export default FetchData;
```

- `useEffect`: Runs the fetch operation when the component mounts.
- `axios.get`: Fetches user data from the API.
- State variables (`loading`, `error`, `users`) manage the UI dynamically.

Summary

- React Router DOM: Simplifies navigation between pages using `createBrowserRouter` and `RouterProvider`.

- `useRef` vs `useState`: Use `useState` for rendering updates and `useRef` for mutable values that don't cause re-renders.

- Axios: Makes data fetching cleaner with promises and error handling.