React Router and Hooks

React Router is a powerful library that allows you to implement dynamic routing in a React application. It enables you to define different routes in your application, making it possible to navigate between different components or pages. Hooks like `useState` and `useEffect` are essential tools in functional components, providing state management and side effects, respectively. In this guide, we'll cover how to use React Router for routing and how to incorporate hooks like `useState` and `useEffect` in a functional component.

1. Setting Up React Router

To use React Router, you need to install the `react-router-dom` package:

bash

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

a. Basic Setup with React Router

To set up routing, you typically wrap your app with the `BrowserRouter` component, and then use `Route`, `Switch`, and `Link` components to define routes and navigation.

Example: Basic Routing Setup

jsx

```
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter as Router, Route, Switch, Link } from
'react-router-dom';

// Home component
function Home() {
  return <h2>Home Page</h2>;
}
```

```
// About component
function About() {
  return <h2>About Page</h2>;
}

// Contact component
function Contact() {
  return <h2>Contact Page</h2>;
}

function App() {
  return (
    <Router>
      <div>
        {/* Navigation Links */}
        <nav>
          <ul>
            <li><Link to="/">Home</Link></li>
            <li><Link to="/about">About</Link></li>
            <li><Link to="/contact">Contact</Link></li>
          </ul>
        </nav>

        {/* Route Definitions */}
        <Switch>
          <Route exact path="/" component={Home} />
          <Route path="/about" component={About} />
          <Route path="/contact" component={Contact} />
        </Switch>
      </div>
    </Router>
  );
}
```

```
ReactDOM.render(<App />, document.getElementById('root'));
```

- **`BrowserRouter` (alias `Router`)**: Wraps your entire application, enabling the use of routing components.
- **`Route`**: Defines the component that should be rendered when the path matches the current URL.
- **`Switch`**: Ensures that only the first matching `Route` is rendered.
- **`Link`**: Provides navigation links that update the URL without refreshing the page, enabling smooth transitions between routes.

b. Dynamic Routing with URL Parameters

You can create dynamic routes that capture parameters from the URL.

Example: Dynamic Routing

jsx

```
import React from 'react';
import { BrowserRouter as Router, Route, Switch, Link, useParams
} from 'react-router-dom';

function UserProfile() {
  // useParams hook to get URL parameters
  let { username } = useParams();
  return <h2>User Profile: {username}</h2>;
}

function App() {
  return (
    <Router>
      <div>
        <nav>
          <ul>
            <li><Link to="/user/johndoe">John Doe</Link></li>
            <li><Link to="/user/janedoe">Jane Doe</Link></li>
          </ul>
```

```
        </nav>

        <Switch>
          <Route path="/user/:username" component={UserProfile}
/>
        </Switch>
      </div>
    </Router>
  );
}

export default App;
```

- **`useParams`**: A hook that retrieves the parameters from the current route. In this example, `username` is extracted from the URL.

2. Using Hooks in Functional Components

React hooks like `useState` and `useEffect` allow functional components to manage state and handle side effects.

a. Managing State with `useState`

`useState` is a hook that lets you add state to functional components.

Example: Counter with `useState`

jsx

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

function Counter() {
  // Declare a state variable `count` initialized to 0
  const [count, setCount] = useState(0);

  return (
    <div>
```

```
      <p>Count: {count}</p>
      <button onClick={() => setCount(count +
1)}>Increment</button>
    </div>
  );
}

export default Counter;
```

- **`useState(0)`**: Initializes the `count` state variable to `0`.
- **`setCount`**: A function to update the `count` state. Each time the button is clicked, `count` is incremented by 1, causing the component to re-render.

b. Handling Side Effects with `useEffect`

`useEffect` is a hook that allows you to perform side effects in functional components, such as fetching data, directly manipulating the DOM, or subscribing to services.

Example: Fetching Data with `useEffect`

jsx

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

function DataFetcher() {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    // Fetch data from an API
    fetch('https://jsonplaceholder.typicode.com/posts/1')
      .then(response => response.json())
      .then(data => {
        setData(data);
        setLoading(false);
      });
```

```
  }, []); // Empty dependency array means this effect runs once
when the component mounts

  if (loading) {
    return <p>Loading...</p>;
  }

  return (
    <div>
      <h2>{data.title}</h2>
      <p>{data.body}</p>
    </div>
  );
}

export default DataFetcher;
```

- **`useEffect()`**: Runs after every render by default. You can control when it runs by passing a dependency array as the second argument.
    - **`[]` (Empty Array)**: The effect runs only once after the initial render (componentDidMount equivalent).
    - **`[dependency1, dependency2]`**: The effect runs when any of the dependencies change.

3. Combining React Router and Hooks

You can combine React Router with hooks like `useState` and `useEffect` to create dynamic and interactive components.

Example: Dynamic Data Fetching Based on Route Parameter

jsx

```
import React, { useState, useEffect } from 'react';
import { BrowserRouter as Router, Route, Switch, Link, useParams
} from 'react-router-dom';
```

```
function Post() {
  const { postId } = useParams();
  const [post, setPost] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    // Fetch post data based on the postId route parameter

fetch(`https://jsonplaceholder.typicode.com/posts/${postId}`)
      .then(response => response.json())
      .then(data => {
        setPost(data);
        setLoading(false);
      });
  }, [postId]); // Effect runs when postId changes

  if (loading) {
    return <p>Loading...</p>;
  }

  return (
    <div>
      <h2>{post.title}</h2>
      <p>{post.body}</p>
    </div>
  );
}

function App() {
  return (
    <Router>
      <div>
        <nav>
          <ul>
            <li><Link to="/post/1">Post 1</Link></li>
```

```
            <li><Link to="/post/2">Post 2</Link></li>
            <li><Link to="/post/3">Post 3</Link></li>
          </ul>
        </nav>

        <Switch>
          <Route path="/post/:postId" component={Post} />
        </Switch>
      </div>
    </Router>
  );
}

export default App;
```

- **`useParams()`**: Retrieves the `postId` from the URL, triggering the `useEffect` hook whenever `postId` changes.
- **`useEffect`**: Fetches data for the specific post every time the route changes, ensuring the correct data is displayed.

Summary

- **React Router**: Provides routing capabilities in React applications, allowing you to define routes, navigate between components, and capture URL parameters.
- **`useState`**: A hook that enables state management in functional components.
- **`useEffect`**: A hook that handles side effects in functional components, such as fetching data, subscriptions, or manually changing the DOM.
- **Combining Router and Hooks**: React Router can be seamlessly integrated with hooks like `useState` and `useEffect` to create dynamic, responsive components that react to route changes and manage state effectively.