React Router Hooks

- React Router hooks are a set of React hooks provided by the React Router library that simplify working with routing in functional components.
- React Router provides a set of hooks that allow developers to work seamlessly with routing in React applications.
- They allow developers to access routing-related information and functionalities, such as navigation, route parameters, and query parameters, without needing to rely on class components or higher-order components like withRouter.

Features of React Router Hooks

- **Integration with Functional Components:** React Router hooks are designed to be used in functional components, aligning with React's modern development practices.
- **Improved Developer Experience:** Hooks provide a cleaner and more intuitive API for accessing routing-related functionality, reducing boilerplate code.
- **Dynamic Routing:** Hooks enable easy access to route parameters and navigation functions, making dynamic routing seamless.
- **Modularity:** Each hook serves a specific purpose, allowing developers to pick and use only what is necessary for a given component.

General Use Cases

- **Dynamic Page Rendering:** Hooks make it simple to extract route parameters or query parameters to render content dynamically.
  Example: Rendering a product details page based on a dynamic product ID from the URL.
- **Programmatic Navigation:** They enable navigation between pages programmatically, such as redirecting users after form submission or authentication.
- **Stateful URL Management:** React Router hooks allow managing the application's state via the URL, making it easy to handle search queries or filters.
- **Condition-Based Rendering:** Hooks can check the current route or match patterns to conditionally render components or layouts.

useParams Hook

- The useParams hook is a powerful utility provided by React Router for accessing dynamic parameters defined in the URL.
- It is widely used in modern single-page applications (SPAs) to handle dynamic routing scenarios where parts of the URL are variable.
- The useParams hook is a part of the React Router library that allows you to access route parameters defined in your application's URL.
- It is primarily used to retrieve dynamic segments from the route.

Core Concepts of useParams

- **Dynamic Routing:** Routes in React Router can include placeholders (e.g., /user/:id), representing dynamic segments of the URL. These placeholders allow developers to handle a wide range of use cases, such as user profiles, product pages, or category-based content.
- **Parameter Extraction:** useParams extracts the values of these placeholders and returns them as an object. Each key in the object corresponds to the placeholder name, and the value is the actual value from the URL.
- **Real-Time Updates:** The hook updates automatically if the route changes while the component is still mounted, making it ideal for real-time applications that depend on URL changes.
- **Integration with Functional Components:** Designed specifically for functional components, it aligns with React's modern architecture and avoids the need for HOCs or class components.

How useParams Works

1. **Route Definition:** Dynamic segments are defined in the route path using syntax.
   ```
   <Route path="/user/:id" element={<UserProfile />} />
   ```
2. **Access Parameters:** The useParams hook is used in the component rendered by the route to access the dynamic values.
   ```
   const { id } = useParams();
   ```
3. **Parameter Object:** useParams returns an object with key-value pairs, where:
   - Key: The name of the dynamic segment (e.g., id).
   - Value: The actual value from the URL.

Use Cases

- **Dynamic Data Fetching:** Fetch specific data (e.g., user profile or product details) based on the ID or slug in the URL.
- **Nested and Child Routes:** Use useParams to retrieve parameters in nested routes for hierarchical data display.
- **SEO-Friendly URLs:** Create clean and semantic URLs using parameters like /products/shoes.
- **Conditional Content Rendering:** Render different components or content based on the parameter value.
- **Building Deep Linking:** Enable users to share direct links to specific content by embedding parameters in the URL.

Example: User Profile Page

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

function UserProfile() {
  const { userId } = useParams();
  return <h1>Welcome, User {userId}</h1>;
}

function App() {
  return (
    <Router>
      <Routes>
        { /* Define the route for UserProfile */ }
        <Route path="/user/:userId" element={<UserProfile />} />
      </Routes>
    </Router>
  );
}

export default App;
```

Example: Get the user object on the basis of the route using useParams

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

function UserProfile() {
  const { userId } = useParams();
  const [user, setUser] = useState(null);

  // Mock user data for demonstration
  const mockUsers = {
    1: { name: 'Neha Rathore', age: 22, email: 'neha@test.com' },
    2: { name: 'Tushar Gupta', age: 23, email: 'tushar@test.com' },
    3: { name: 'Garima Singh', age: 22, email: 'garima@test.com' },
  };

  useEffect(() => {
    // Simulate fetching user data
    const fetchedUser = mockUsers[userId];
    setUser(fetchedUser);
  }, [userId]);

  return (
    <div>
      {user ? (
        <div>
          <h1>User Profile</h1>
          <p><strong>Name:</strong> {user.name}</p>
          <p><strong>Age:</strong> {user.age}</p>
          <p><strong>Email:</strong> {user.email}</p>
        </div>
      ) : (
        <h1>User not found!</h1>
      )}
    </div>
  );
}
```

```
function App() {
  return (
    <Router>
      <Routes>
        { /* Define the route for UserProfile */ }
        <Route path="/user/:userId" element={<UserProfile />} />
      </Routes>
    </Router>
  );}
export default App;
```

Example: Simulate fetching blog details dynamically

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

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

  // Mock blog data
  const mockBlogPosts = {
    react: {
      1: { title: 'Understanding React Hooks', content: 'React Hooks simplify functional components.' },
      2: { title: 'Become an expert in React Hooks', content: 'A list of Detailed exercises and problem statements to make you an expert.' },
    },
    node: {
      1: { title: 'Understanding BackEnd', content: 'Learn Node JS to become a Back End Developer.' },
      2: { title: 'Node Js & Express JS', content: 'Create RESTFUL APIs and Manage Back End Operations.' },
    },  };
```

```
  useEffect(() => {
    // Simulate fetching data
    setLoading(true);
    setTimeout(() => {
      const categoryPosts = mockBlogPosts[category];
      if (categoryPosts && categoryPosts[postId]) {
        setPost(categoryPosts[postId]);
      } else {
        setPost(null); // No post found
      }
      setLoading(false);
    }, 500); // Simulate network delay
  }, [category, postId]);

  return (
    <div>    {loading ? (   <h2>Loading...</h2>    ) : post ? (
        <div>
          <h1>Category: {category}</h1>
          <h2>{post.title}</h2>
          <p>{post.content}</p>
        </div>
      ) : (       <h2>Post not found!</h2>      )}
    </div>
  );}

function App() {
  return (
    <Router>
      <Routes>
        {/* Define the route for BlogPost */}
        <Route path="/blog/:category/:postId" element={<BlogPost />} />
      </Routes>
    </Router>
  );
}
export default App;
```

Example: Nested Route Parameters with dynamic data fetching, loading states, and error handling

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

function ProductDetails() {
  const { productId, reviewId } = useParams();
  const [product, setProduct] = useState(null);
  const [review, setReview] = useState(null);
  const [loading, setLoading] = useState(true);

  // Mock product and review data
  const mockData = {
    products: {
      51: {
        name: 'iPhone 16 Pro Max',
        price: 'INR 2,00,000',
        reviews: {
          101: { title: 'Excellent!', content: 'Great product, highly recommend.' },
          102: { title: 'Good Value', content: 'Prices High but justified and works well.'
},
        },
      },
      52: {
        name: 'Apple MacBook Pro 13 Inch',
        price: 'INR 1,80,000',
        reviews: {
          201: { title: 'Superb', content: 'High performance for professionals.' },
          202: { title: 'Optimised', content: 'Great product with high performance
computing.' },
        },
      },
    },
  };
  useEffect(() => {
    setLoading(true);
```

```
  // Simulate fetching data
  setTimeout(() => {
    const fetchedProduct = mockData.products[productId];
    if (fetchedProduct) {
      setProduct(fetchedProduct);
      const fetchedReview = fetchedProduct.reviews[reviewId];
      if (fetchedReview) {
        setReview(fetchedReview);
      } else {
        setReview(null);
      }
    } else {
      setProduct(null);
    }
    setLoading(false);
  }, 500); // Simulate network delay
}, [productId, reviewId]);

return (
  <div>
    {loading ? (
      <h2>Loading...</h2>
    ) : product && review ? (
      <div>
        <h1>Product Details</h1>
        <p><strong>ID:</strong> {productId}</p>
        <p><strong>Name:</strong> {product.name}</p>
        <p><strong>Price:</strong> {product.price}</p>

        <h2>Review Details</h2>
        <p><strong>ID:</strong> {reviewId}</p>
        <p><strong>Title:</strong> {review.title}</p>
        <p><strong>Content:</strong> {review.content}</p>
      </div>
    ) : (
      <h2>{product ? 'Review not found!' : 'Product not found!'}</h2>
    )}
```

```
    </div>
  );
}

function App() {
  return (
    <Router>
      <Routes>
       { /* Route for ProductDetails */ }
        <Route path="/products/:productId/reviews/:reviewId"
element={<ProductDetails />} />
      </Routes>
    </Router>
  );
}
export default App;
```

Best Practices for useParams

- **Validate Parameters:** Always validate parameters, especially if they come from untrusted sources like user-modifiable URLs.
- **Error Handling:** Provide fallback or error handling for invalid or missing parameters.
- **Keep Component Logic Clean:** Avoid mixing parameter extraction and complex logic in the same component. Use helper functions or separate components if needed.
- **Avoid Tight Coupling:** Do not hardcode parameter values or route paths in components. Use route configurations for better maintainability.

Benefits of useParams

- **Declarative API:** Clear and simple syntax for accessing route parameters.
- **Real-Time Reactivity:** Automatically updates as the route changes.
- **Functional Component Friendly:** Works seamlessly with React's hooks-based architecture.
- **Scalable:** Suitable for both simple and complex routing scenarios.

useLocation Hook

- The useLocation hook is a utility provided by React Router that allows you to access the current location object.
- This object contains information about the current URL, including the pathname, search query, and state.
- It is primarily used to track the application's navigation state and manage dynamic rendering based on the current location.

Key Concepts of useLocation

- **Location Object:** The useLocation hook returns a location object with the following properties:
  - pathname: The current URL path (e.g., /home).
  - search: Query string parameters in the URL (e.g., ?user=123).
  - hash: The hash fragment in the URL (e.g., #section1).
  - state: Optional custom state passed during navigation.
- **Tracking Navigation:** Enables components to react to navigation changes by monitoring the location object.
- **Dynamic Rendering:** Use useLocation to render components or content conditionally based on the current URL or query parameters.
- **Deep Integration:** Works seamlessly with navigation libraries, allowing enhanced control over routing and state management.

Syntax

```
const location = useLocation();
```

How useLocation Works

- **Access Current Location:** When called, useLocation fetches the current location object, reflecting the active URL.
- **Reactivity:** The location object updates automatically when the URL changes. This makes useLocation reactive, allowing components to respond dynamically to navigation.
- **Integration with React Router:** It works within a React Router context, so components using useLocation must be rendered within a Router.

Step-by-Step Workflow

- **Define Routes:** Set up your routes using Route components.

```
<Routes>
  <Route path="/about" element={<About />} />
  <Route path="/contact" element={<Contact />} />
</Routes>
```

- **Use useLocation in a Component:** Access the current location in any child component.

```
import { useLocation } from 'react-router-dom';

function CurrentLocation() {
  const location = useLocation();
  return (
    <div>
      <p>Pathname: {location.pathname}</p>
      <p>Search: {location.search}</p>
      <p>Hash: {location.hash}</p>
      <p>State: {JSON.stringify(location.state)}</p>
    </div>
  );
}
```

- **Navigation Updates:** Whenever the user navigates to a new route, the location object updates, and the component re-renders to reflect the new location details.

Example: How useLocation Works

```
import { BrowserRouter as Router, Routes, Route, useLocation } from 'react-router-dom';

function DisplayLocation() {
  const location = useLocation();
  return (
    <div>
      <h2>Current Location</h2>
      <p>Pathname: {location.pathname}</p>
```

```
      <p>Search: {location.search}</p>
      <p>Hash: {location.hash}</p>
    </div>
  );
}

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<h1>Home</h1>} />
        <Route path="/about" element={<DisplayLocation />} />
        <Route path="/contact" element={<DisplayLocation />} />
      </Routes>
    </Router>
  );
}

export default App;
```

Use URL Like,

/about?searchItem=Electronics#Laptops

Output:

Pathname: /about
Search: ?searchItem=Electronics
Hash: #Laptops

Example: Access Query Parameters

```
import { BrowserRouter as Router, Routes, Route, useLocation } from 'react-
router-dom';
function QueryParamsExample() {
 const location = useLocation();
 const queryParams = new URLSearchParams(location.search);
 const user = queryParams.get('user');
 const age = queryParams.get('age'); // fetching an additional query parameter

 return (
  <div>
   {user ? (
    <div>
     <h1>User ID: {user}</h1>
     {age && <h2>Age: {age}</h2>} { /* Display age if provided */ }
    </div>
   ) : (   <h1>No user found in query parameters!</h1>      )}
  </div>
 );
}

function App() {
 return (
  <Router>
   <Routes>
    { /* Route for QueryParamsExample */ }
    <Route path="/search" element={<QueryParamsExample />} />
   </Routes>
  </Router>
 );
}
export default App;
```

Test the behaviour using URL's like:

- URL: /search?user=Garima421
- URL: /search?user=Garima421&age=22
- URL: /search

Example: Highlight Active Navigation Links

```
import { useLocation, NavLink, BrowserRouter as Router, Routes, Route } from 'react-router-dom';

function Navbar() {
 const location = useLocation();

 return (
  <nav>
   <ul style={{ listStyleType: 'none', display: 'flex', gap: '2rem' }}>
    <li>
     <NavLink
      to="/home"
      className={location.pathname === '/home' ? 'active' : ''}
      style={{
       textDecoration: 'none',
       color: location.pathname === '/home' ? 'blue' : 'black',
      }}
     >
      Home
     </NavLink>
    </li>
    <li>
     <NavLink
      to="/about"
      className={location.pathname === '/about' ? 'active' : ''}
      style={{
       textDecoration: 'none',
       color: location.pathname === '/about' ? 'blue' : 'black',
      }}
     >
      About
     </NavLink>
    </li>
    <li>
     <NavLink
      to="/contact"
```

```
            className={location.pathname === '/contact' ? 'active' : ''}
            style={{
              textDecoration: 'none',
              color: location.pathname === '/contact' ? 'blue' : 'black',
            }}
          >
            Contact
          </NavLink>
        </li>
      </ul>
    </nav>
  );
}
function Home() {
  return <h1>Welcome to the Home Page</h1>;
}
function About() {
  return <h1>About Us</h1>;
}
function Contact() {
  return <h1>Contact Us</h1>;
}

function App() {
  return (
    <Router>
      <Navbar />
      <Routes>
        <Route path="/home" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
      </Routes>
    </Router>
  );
}
export default App;
```

Example: Scroll to Section Based on Hash

```
import { useEffect } from 'react';
import { BrowserRouter as Router, Routes, Route, useLocation } from 'react-router-dom';

function ScrollToHash() {
  const location = useLocation();

  useEffect(() => {
    if (location.hash) {
      // Wait for the DOM to fully load before scrolling
      setTimeout(() => {
        const element = document.querySelector(location.hash);
        if (element) {
          element.scrollIntoView({ behavior: 'smooth', block: 'start' });
        }
      }, 100); // Slight delay to account for rendering time
    }
  }, [location]);

  return null;
}

function Home() {
  return (
    <div>
      <h1>Home Page</h1>
      <div style={{ display: 'flex', gap: '1rem', marginBottom: '2rem' }}>
        <a href="#section1">Go to First Section</a>
        <a href="#section2">Go to Second Section</a>
        <a href="#section3">Go to Third Section</a>
      </div>

      <div style={{ marginTop: '100vh', padding: '1rem', backgroundColor: '#f0f0f0' }} id="section1">
        <h2>First Section</h2>
        <p>Hello User, This is First Section.</p>
```

```
      </div>
      <div style={{ marginTop: '100vh', padding: '1rem', backgroundColor: '#d0d0d0' }} id="section2">
        <h2>Second Section</h2>
        <p>Hello User, This is Second Section.</p>
      </div>
      <div style={{ marginTop: '100vh', padding: '1rem', backgroundColor: '#c0c0c0' }} id="section3">
        <h2>Third Section</h2>
        <p>Hello User, This is Third Section.</p>
      </div>
    </div>
  );
}

function App() {
  return (
    <Router>
      <ScrollToHash />
      <Routes>
        <Route path="/home" element={<Home />} />
      </Routes>
    </Router>
  );
}

export default App;
```

Use Cases

- **Display Dynamic Content:** Use the pathname or search parameters to conditionally render components or fetch data.
- **Custom Navigation State:** Access and utilize custom state passed during navigation to manage component behavior.
- **Query Parameters Management:** Extract query parameters to build search or filter functionalities.
- **Scroll or Section Management:** Implement navigation based on hash fragments to scroll to specific sections of a page.
- **Breadcrumbs or Navigation Tracking:** Dynamically generate breadcrumbs or track the user's navigation path.

Best Practices

- **Validate Inputs:** When using search or state, always validate or sanitize inputs to avoid potential issues.
- **Avoid Hardcoding:** Do not hardcode URL paths or query parameters directly in components. Use route configurations or utility functions.
- **Fallback Handling:** Provide default values or graceful fallbacks when expected properties like state or search are missing.
- **Combine with useNavigate:** Use useLocation along with useNavigate for managing complex navigation scenarios.

Benefits of useLocation

- **Real-Time Location Access:** Provides real-time access to the current URL and state without reloading the page.
- **Simplified Query Management:** Makes working with query strings straightforward, enabling powerful search and filtering functionality.
- **Seamless State Passing:** Enables easy transfer of custom state during navigation without relying on global state management tools.
- **Dynamic Rendering:** Allows for highly responsive and adaptable component rendering based on URL changes.