

# React Router 6.22.3

---

RUJUAN XING



# What is Routing?

---

Routing in reactJS is the mechanism by which we navigate through a website or web-application. Routing can be server-side routing or client-side routing. However, React Router is a part of client-side routing.

Server-side routing results in every request being a full page refresh loading unnecessary and repeated data every time causing a bad UX and delay in loading.

With client-side routing, it could be the rendering of a new component where routing is handled internally by JavaScript. The component is rendered only without the server being reloaded.

# Why React Router?

---

React Routing without any knowledge of it can be manually implemented using useState and JSX for conditioning.

```
const [page, setpage] = useState("products")
const routeTo = (newpage) => {
  setpage(newpage)
}
<button onClick={() = routeTo("cart")}> Cart />
{page === "cart" && ( <Cart cart={cart} setcart={setcart}/>)}
```

But large-scale application like e-commerce?

# What is React Router?

---

React Router is a fully-featured routing solution for React apps.

- It offers pre-developed components, Hooks, and utility functions to create modern routing strategies.
- It offers various router mechanisms for the web version, such as URL path-based routing, URL-hash routing, and invisible routing (known as memory routing).
- `npm install react-router-dom`

# Different Routers

---

The react-router-dom package offers three higher-level, ready-to-use router components:

- **BrowserRouter:** It is the most common type and most used type of router that uses **HTML5 history API** (pushState, replaceState, and the popstate event) making your UI in sync with the URL.
- **HashRouter:** This type of router uses the hash portion of the URL keeping your UI in sync with the URL (i.e., index.html#/profile)
- **MemoryRouter:** A router that keeps the history of your “URL” in memory does not like to put it in the address bar. Often used but useful in testing and non-browser environments like React Native.

# React Router Components

---

**Routes:** It includes features like relative routing and linking, automatic route ranking, nested routes, and layouts.

**Route:** It is responsible for rendering the UI of a React component. It has a prop called `path` which always matches the current URL of the application. The second required prop is called `element` that tells the `Route` component when a current URL is encountered and which React component to be rendered.

**Link:** The link component is used to create links to different routes and implement navigation around the application. It works like an HTML anchor tag.

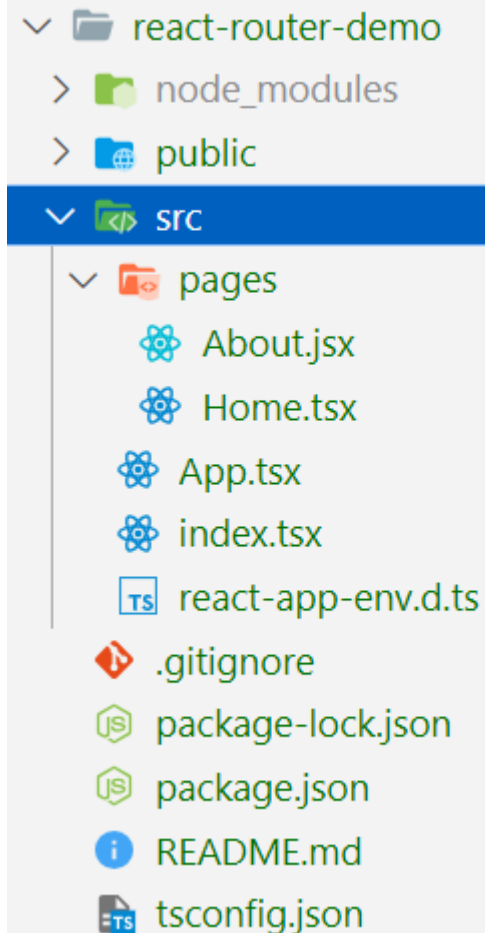
**NavLink:** A `<NavLink>` is a special kind of `<Link>` that knows whether or not it is "active", "pending", or "transitioning". This is useful in a few different scenarios:

- When building a navigation menu, such as a breadcrumb or a set of tabs where you'd like to show which of them is currently selected
- It provides useful context for assistive technology like screen readers

# Demo

```
root.render(<BrowserRouter><App /></BrowserRouter>);
```

```
export default function App() {
  return (
    <div className="container">
      <div className="col-3">
        <div className="d-flex flex-column flex-shrink-0 p-3 border">
          <ul className="nav nav-pills flex-column mb-auto">
            <li className="nav-item">
              <NavLink to="/home" className="nav-link" >
                Home
              </NavLink>
            </li>
            <li>
              <NavLink to="/about" className="nav-link">
                About
              </NavLink>
            </li>
          </ul>
        </div>
      </div>
      <div className="col-9">
        <Routes>
          <Route path="/home" element={<Home />} />
          <Route path="/about" element={<About />} />
        </Routes>
      </div>
    </div>
  )
}
```

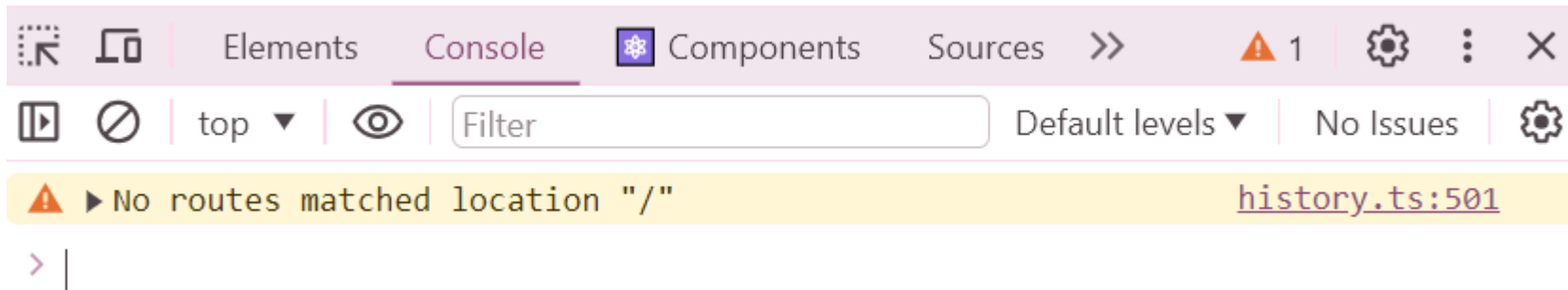


```

react-router-demo
├── node_modules
├── public
├── src
│   ├── pages
│   │   ├── About.jsx
│   │   ├── Home.tsx
│   │   ├── App.tsx
│   │   ├── index.tsx
│   │   └── react-app-env.d.ts
│   ├── .gitignore
│   ├── package-lock.json
│   ├── package.json
│   ├── README.md
│   └── tsconfig.json
```

# <Navigate>

A `<Navigate>` element changes the current location when it is rendered. It's a component wrapper around `useNavigate`, and accepts all the same arguments as props.



```
<Routes>
  <Route path="/home" element={<Home />} />
  <Route path="/about" element={<About />} />
  <Route path="/" element={<Navigate to='/home'/>} />
</Routes>
```



# Active Link

---

By default, the highlighted link has a “active” class.

If we want to change to customized class name?

```
<NavLink to="/home" className={({isActive}) => classNames('nav-link', {highlight: isActive})} >  
  Home  
</NavLink>
```

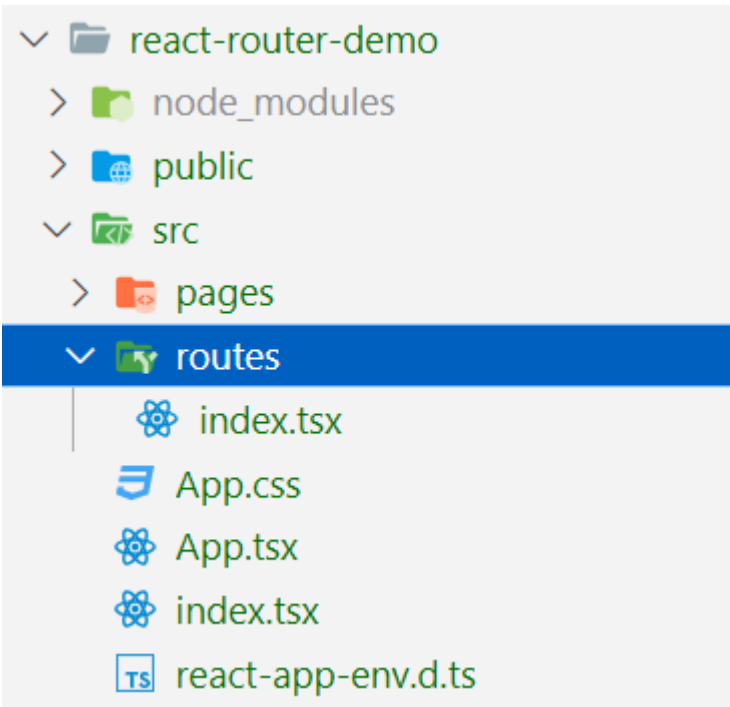
# useRoutes

---

The `useRoutes` hook is the functional equivalent of `<Routes>`, but it uses JavaScript objects instead of `<Route>` elements to define your routes. These objects have the same properties as normal `<Route>` elements, but they don't require JSX.

The return value of `useRoutes` is either a valid React element you can use to render the route tree, or null if nothing matched.

# Demo



```
export default [  
  {  
    path: '/home',  
    element: <Home />  
  },  
  {  
    path: '/about',  
    element: <About />  
  },  
  {  
    path: '/',  
    element: <Navigate to='/home' />  
  }  
]
```

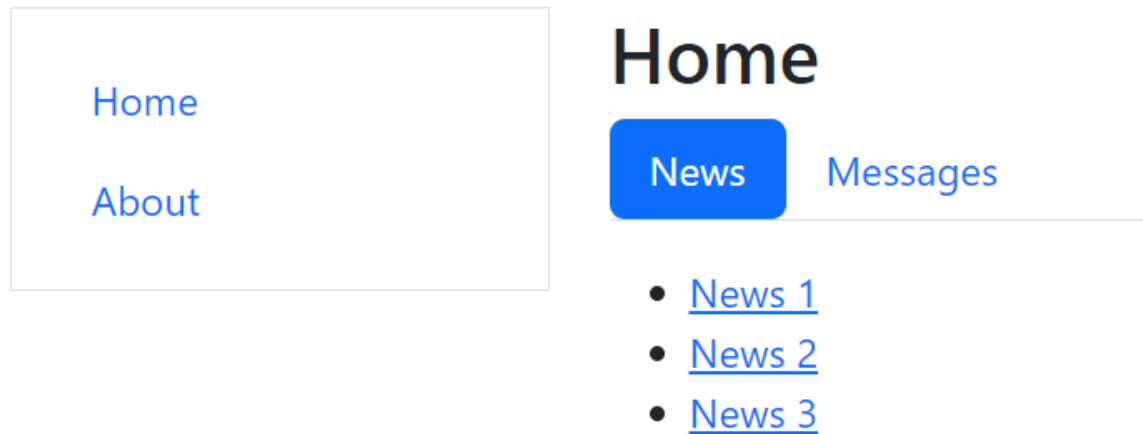
```
export default function App() {  
  const element = useRoutes(routes);  
  return (  
    ...  
    <div className="col-9">  
      {element}  
    </div>  
  )  
}
```

# Nested Routes

---

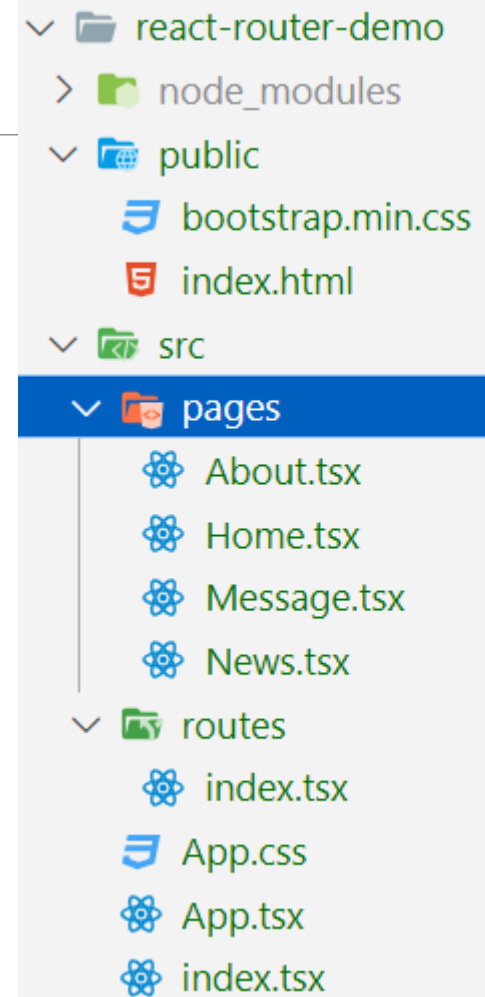
Nested Routing is the general idea of coupling segments of the URL to component hierarchy and data.

For example, on home page one gets presented multiple tabs (e.g. News, Messages) to navigate. By clicking these tabs, the URL in the browser will change, but instead of replacing the whole page, only the content of the tab gets replaced.



# Demo

```
export default function Home() {  
  return (  
    <div>  
      <h2>Home</h2>  
      <ul className="nav nav-pills border-bottom mb-3">  
        <li className="nav-item">  
          <NavLink className="nav-link" to="news">News</NavLink>  
        </li>  
        <li className="nav-item">  
          <NavLink className="nav-link" to="messages">Messages</NavLink>  
        </li>  
      </ul>  
      <h3>???    </div>  
  )  
}
```



# <Outlet>

---

An `<Outlet>` should be used in parent route elements to render their child route elements. This allows nested UI to show up when child routes are rendered. If the parent route matched exactly, it will render a child index route or nothing if there is no index route.

```
export default function Home() {  
  return (  
    <div>  
      <h2>Home</h2>  
      <ul className="nav nav-pills border-bottom mb-3">  
        <li className="nav-item">  
          <NavLink className="nav-link" to="news">News</NavLink>  
        </li>  
        <li className="nav-item">  
          <NavLink className="nav-link" to="messages">Messages</NavLink>  
        </li>  
      </ul>  
      <Outlet />  
    </div>  
  )  
}
```

# useParams

---

The `useParams` hook returns an object of key/value pairs of the dynamic params from the current URL that were matched by the `<Route path>`.

Returns an object that maps the names of URL Parameters to their values in the current URL.

```
export default [
  {
    path: '/home',
    element: <Home />,
    children: [
      {
        path: 'messages',
        element: <Message />,
        children: [
          {
            path: 'detail/:id/:title/:content',
            element: <Detail />
          }
        ]
      }
    ]
  }
]
```

# Demo – Message.tsx

---

```
export default function Message() {  
  
  //messages state here  
  
  return (  
    <div>  
      <ul>  
        {messages.map(msg => (  
          <li>  
            <Link to={`detail/${msg.id}/${msg.title}/${msg.content}`}>{msg.title}</Link>  
          </li>  
        )  
      )}  
      </ul>  
      <hr />  
      <Outlet />  
    </div>  
  )  
}
```



# Demo – Detail.tsx

---

```
export default function Detail() {  
  const { id, title, content } = useParams();  
  return (  
    <div>  
      <h4>Detail</h4>  
      <ul>  
        <li>Message ID: {id}</li>  
        <li>Message Title: {title}</li>  
        <li>Message content: {content}</li>  
      </ul>  
    </div>  
  )  
}
```

# useSearchParams

The `useSearchParams` hook is used to read and modify the query string in the URL for the current location. Like React's own `useState` hook, `useSearchParams` returns an array of two values: the current location's search params and a function that may be used to update them. Just as React's `useState` hook, `setSearchParams` also supports functional updates. Therefore, you may provide a function that takes a `searchParams` and returns an updated version.

```
export default [
  {
    path: '/home',
    element: <Home />,
    children: [
      {
        path: 'messages',
        element: <Message />,
        children: [
          {
            path: 'detail',
            element: <Detail />
          }
        ]
      }
    ]
  }
]
```

# Demo – Message.tsx

```
export default function Message() {  
  
  return (  
    <div>  
      <ul>  
        {messages.map(msg => (  
          <li>  
            <Link  
to={`detail?id=${msg.id}&title=${msg.title}&content=${msg.content}`}>{msg.title}</Link>  
          </li>  
        )  
      )}  
      </ul>  
      <hr />  
      <Outlet />  
    </div>  
  )  
}
```

# Demo – Detail.tsx

---

```
export default function Detail() {  
  const [search, setSearch] = useSearchParams();  
  return (  
    <div>  
      <h4>Detail</h4>  
      <ul>  
        <li>Message ID: {search.get('id')}</li>  
        <li>Message Title: {search.get('title')}</li>  
        <li>Message content: {search.get('content')}</li>  
      </ul>  
    </div>  
  )  
}
```