## ****React Routing****

## ****1. Introduction to Routing in React****

### What is Routing?

In React apps, **routing** allows users to navigate between different components/pages via URL paths, **without reloading the page**. This is essential for single-page applications (SPAs), where all content is loaded on one page and rendered dynamically.

### How Routing Works in React

React doesn't come with built-in routing. Instead, we use a library like **React Router** to:

* Map **URLs to components**
* Navigate between **views/pages**
* Support **parameters**, **nested routes**, **private routes**, and more

### What is React Router?

**React Router** is the **most popular** routing library for React.

It provides:

* Declarative routing using <Routes> and <Route>
* Nested routes with layouts
* URL parameters and search params
* Lazy loading, protected routes, and data loading (v6.4+)

### Key React Router Concepts

| **Concept** | **Purpose** |
| --- | --- |
| <BrowserRouter> | Wraps your app to enable routing |
| <Routes> | Holds all route definitions |
| <Route path="" /> | Defines a route and what to render |
| <Link> / <NavLink> | Navigation elements |
| useNavigate() | Programmatic navigation |
| useParams() | Access dynamic route segments |
| <Outlet> | Nested route rendering |

### React Router with Vite vs Create React App (CRA)

Vite uses main.jsx instead of index.js, and doesn’t pre-install React Router like CRA sometimes does. But React Router works **identically** once set up.

### Step 1: Create a New Vite + React App

npm create vite@latest my-routing-app -- --template react

cd my-routing-app

npm install

### Step 2: Install React Router DOM

npm install react-router-dom

This installs the **client-side routing** package used with React.

### Step 3: Understand the Vite Folder Structure

When you open the project, you’ll typically see:

src/

├── App.jsx

├── main.jsx

├── assets/

* main.jsx – Entry point of the app
* App.jsx – Main component (often the root layout or wrapper)
* No index.js like in CRA

### Step 4: Set Up BrowserRouter in main.jsx

Update your main.jsx to include <BrowserRouter>:

// src/main.jsx

import React from 'react';

import ReactDOM from 'react-dom/client';

import { BrowserRouter } from 'react-router-dom';

import App from './App';

import './index.css';

ReactDOM.createRoot(document.getElementById('root')).render(

<React.StrictMode>

<BrowserRouter>

<App />

</BrowserRouter>

</React.StrictMode>

);

### Step 5: Create Basic Pages

Inside src/, create a pages/ directory and add some basic components:

**Home.jsx**:

export default function Home() {

return <h1>Home Page</h1>;

}

**About.jsx**:

export default function About() {

return <h1>About Page</h1>;

}

**Contact.jsx**:

export default function Contact() {

return <h1>Contact Page</h1>;

}

### Step 6: Set Up Routes in App.jsx

Now configure routing in your App.jsx:

// src/App.jsx

import { Routes, Route, Link } from 'react-router-dom';

import Home from './pages/Home';

import About from './pages/About';

import Contact from './pages/Contact';

function App() {

return (

<div>

<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>

</div>

);

}

export default App;

### Step 7: Run the App

Start the dev server:

npm run dev

Navigate to:

* / → Home
* /about → About
* /contact → Contact

Navigation is **instant** with no page reload.

## ****3. Basic Routing****

### 3.1 Setup Recap

Before proceeding, ensure:

* You have BrowserRouter in main.jsx
* App.jsx contains a <Routes> block
* You have at least Home, About, and Contact components

### 3.2 Define More Static Routes

Let’s add a Services.jsx page.

**src/pages/Services.jsx**:

export default function Services() {

return <h1>Services Page</h1>;

}

**Update App.jsx**:

import Services from './pages/Services';

<Route path="/services" element={<Services />} />

### 3.3 Add Navigation with <Link> and <NavLink>

**Link** navigates without reloads.

<Link to="/about">About</Link>

**NavLink** adds an active class automatically:

<NavLink to="/about" style={({ isActive }) => ({

color: isActive ? 'red' : 'blue',

})}>

About

</NavLink>

Use NavLink for nav menus where you want active link styling.

You **don't need to define the style prop for every NavLink** if you're repeating the same active style logic. Instead, you can clean things up in a few ways:

**Option 1: Define a reusable function**

Create a function once (e.g., in App.jsx or a utils file), and reuse it for all your NavLinks:

const navLinkStyle = ({ isActive }) => ({

color: isActive ? 'red' : 'blue',

});

Then use it like this:

<NavLink to="/about" style={navLinkStyle}>About</NavLink>

<NavLink to="/contact" style={navLinkStyle}>Contact</NavLink>

**Option 2: Use className instead**

If you're using CSS (which is cleaner and more scalable), go with className and define styles in a CSS file:

**In your CSS:**

.nav-link {

color: blue;

text-decoration: none;

}

.nav-link.active {

color: red;

}

**In JSX:**

<NavLink to="/about" className={({ isActive }) => isActive ? 'nav-link active' : 'nav-link'}>

About

</NavLink>

### 3.4 Programmatic Navigation with useNavigate

React Router lets you navigate with a function.

**Example:**

import { useNavigate } from 'react-router-dom';

function Contact() {

const navigate = useNavigate();

const handleSubmit = () => {

// Do something then redirect

navigate('/');

};

return (

<div>

<h1>Contact</h1>

<button onClick={handleSubmit}>Go Home</button>

</div>

);

}

### ❌ 3.5 Handling Unknown Routes (404)

At the end of your routes, add:

<Route path="\*" element={<h1>404 - Page Not Found</h1>} />

This catches all unmatched paths.

## ****4. Route Parameters****

### 4.1 What Are Route Parameters?

Route parameters let you pass values **via the URL**, e.g.:

* /user/123 – user ID is 123
* /post/hello-world – post slug is hello-world

Defined using :paramName syntax.

### 4.2 Create a Dynamic User Page

**src/pages/User.jsx**:

import { useParams } from 'react-router-dom';

export default function User() {

const { userId } = useParams();

return <h1>User ID: {userId}</h1>;

}

**Update App.jsx**:

import User from './pages/User';

<Route path="/user/:userId" element={<User />} />

Now go to /user/42 → you'll see: User ID: 42

### 4.3 Using Multiple Params

**Route**:

<Route path="/user/:userId/post/:postId" element={<UserPost />} />

**Component**:

import { useParams } from 'react-router-dom';

function UserPost() {

const { userId, postId } = useParams();

return <h1>User {userId} - Post {postId}</h1>;

}

### 4.4 Optional Params (Hacky Way)

React Router v6 does not have true optional params, but you can **define multiple routes**:

<Route path="/product" element={<Product />} />

<Route path="/product/:productId" element={<Product />} />

Then in your component:

const { productId } = useParams();

### 4.5 Search Params?

Search params (query strings) are key-value pairs in the URL after ?.

Example:

/products?page=2&category=books

React Router provides a built-in hook:

import { useSearchParams } from 'react-router-dom';

function Products() {

const [searchParams, setSearchParams] = useSearchParams();

const page = searchParams.get('page'); // e.g. "2"

const category = searchParams.get('category'); // e.g. "books"

return (

<div>

<h2>Category: {category}</h2>

<p>Page: {page}</p>

</div>

);

}

## ****5. Nested Routing****

### 5.1 What is Nested Routing?

Nested routing lets you create **sub-routes inside parent routes** — like:

/dashboard → Dashboard Layout

/dashboard/stats → DashboardStats

/dashboard/profile → DashboardProfile

You define this **hierarchy of routes** using <Outlet /> to render children.

### 5.2 Create a Dashboard Layout and Pages

Create:

src/pages/dashboard/

├── DashboardLayout.jsx

├── Stats.jsx

└── Profile.jsx

**DashboardLayout.jsx**:

import { Outlet, Link } from 'react-router-dom';

export default function DashboardLayout() {

return (

<div>

<h2>Dashboard</h2>

<nav>

<Link to="stats">Stats</Link> |{' '}

<Link to="profile">Profile</Link>

</nav>

<Outlet /> {/\* nested routes render here \*/}

</div>

);

}

**Stats.jsx**:

export default function Stats() {

return <h3>Dashboard Stats</h3>;

}

**Profile.jsx**:

export default function Profile() {

return <h3>User Profile</h3>;

}

### 5.3 Add Nested Routes in App.jsx

import DashboardLayout from './pages/dashboard/DashboardLayout';

import Stats from './pages/dashboard/Stats';

import Profile from './pages/dashboard/Profile';

<Route path="/dashboard" element={<DashboardLayout />}>

<Route path="stats" element={<Stats />} />

<Route path="profile" element={<Profile />} />

</Route>

Now:

* /dashboard → just shows layout
* /dashboard/stats → renders Stats inside layout
* /dashboard/profile → renders Profile inside layout

### 5.4 Nested Path Notes

* Use **relative paths** inside nested routes ("stats" not "/stats")
* <Outlet /> is **required** in parent to show children

## ****6. Layouts and Shared Components****

### 6.1 What Are Layouts?

**Layouts** are components (like headers, sidebars, footers) shared across pages.

Use layouts to avoid repeating the same markup in every page.

### 6.2 Create a MainLayout

Let’s build a layout with a header and footer.

**src/layouts/MainLayout.jsx**:

import { Outlet, Link } from 'react-router-dom';

export default function MainLayout() {

return (

<>

<header style={{ background: '#ddd', padding: '1rem' }}>

<nav>

<Link to="/">Home</Link> |{' '}

<Link to="/about">About</Link> |{' '}

<Link to="/contact">Contact</Link>

</nav>

</header>

<main style={{ padding: '1rem' }}>

<Outlet />

</main>

<footer style={{ background: '#eee', padding: '1rem', marginTop: '1rem' }}>

<p>&copy; 2025 My App</p>

</footer>

</>

);

}

### 6.3 Wrap Public Routes with the Layout

In App.jsx:

import MainLayout from './layouts/MainLayout';

<Route element={<MainLayout />}>

<Route path="/" element={<Home />} />

<Route path="/about" element={<About />} />

<Route path="/contact" element={<Contact />} />

</Route>

This wraps **all public routes** with MainLayout.

### 6.4 You Can Use Multiple Layouts

Example: public vs admin.

<Route element={<MainLayout />}>

{/\* public routes \*/}

</Route>

<Route path="/dashboard" element={<DashboardLayout />}>

{/\* private or admin routes \*/}

</Route>

### Benefits of Layout + Nested Routing

* Reuse UI (headers, footers, navbars)
* Modular structure
* Clear separation of concerns
* Cleaner and scalable routing

### Final Layout Structure Recap

App.jsx

│

├── MainLayout

│ ├── Home

│ ├── About

│ └── Contact

│

└── DashboardLayout

├── Stats

└── Profile

## ****7. Protected Routes (Private Routing)****

### 7.1 What Are Protected Routes?

Protected (or private) routes restrict access to **authenticated users**. If someone isn’t logged in, they are **redirected** to login or another page.

### 7.2 Simulate Auth State

We'll simulate authentication using React state or context.

**Create src/hooks/useAuth.js**:

import { useState } from 'react';

export default function useAuth() {

const [user, setUser] = useState(null);

const login = (username) => setUser({ name: username });

const logout = () => setUser(null);

return { user, login, logout };

}

### 7.3 Create a ProtectedRoute Component

**src/components/ProtectedRoute.jsx**:

import { Navigate, useLocation } from 'react-router-dom';

export default function ProtectedRoute({ user, children }) {

const location = useLocation();

if (!user) {

return <Navigate to="/login" state={{ from: location }} replace />;

}

return children;

}

### 7.4 Create Login Page

**src/pages/Login.jsx**:

import { useLocation, useNavigate } from 'react-router-dom';

export default function Login({ login }) {

const navigate = useNavigate();

const location = useLocation();

const from = location.state?.from?.pathname;

const handleLogin = () => {

login('john'); // Simulate login

navigate(from, { replace: true });

};

return (

<div>

<h1>Login Page</h1>

<button onClick={handleLogin}>Login</button>

</div>

);

}

### 7.5 Add Protected Route in App

In App.jsx:

import { useState } from 'react';

import ProtectedRoute from './components/ProtectedRoute';

import Login from './pages/Login';

import DashboardLayout from './pages/dashboard/DashboardLayout';

function App() {

const [user, setUser] = useState(null);

const login = (username) => setUser({ name: username });

const logout = () => setUser(null);

return (

<Routes>

<Route path="/login" element={<Login login={login} />} />

<Route

path="/dashboard/\*"

element={

<ProtectedRoute user={user}>

<DashboardLayout />

</ProtectedRoute>

}

>

<Route path="stats" element={<Stats />} />

<Route path="profile" element={<Profile />} />

</Route>

</Routes>

);

}

### 

## ****8. Navigation & Redirection****

### 8.1 Navigate Programmatically with useNavigate

Import and use:

import { useNavigate } from 'react-router-dom';

const navigate = useNavigate();

navigate('/about'); // Go to About

Optionally:

navigate('/dashboard', { replace: true }); // Replaces history

### 8.2 Redirect Using <Navigate />

Use <Navigate /> when you want to **conditionally redirect** inside JSX:

if (!user) {

return <Navigate to="/login" replace />;

}

### 8.3 Redirect After Login

This is covered in Section 7:

const from = location.state?.from?.pathname || '/';

navigate(from, { replace: true });

It ensures users go **back to where they came from** after login.

### 8.4 Redirect on Button Click or Timeout

setTimeout(() => {

navigate('/home');

}, 2000);

Or use in a button:

<button onClick={() => navigate('/home')}>Go Home</button>

### 8.5 useLocation for Accessing Navigation State

You can pass and read values like this:

navigate('/checkout', { state: { price: 20 } });

Then access via useLocation:

const location = useLocation();

console.log(location.state.price); // 20

## ****9. Lazy Loading and Code Splitting****

### 9.1 What is Lazy Loading?

Lazy loading means **loading components only when needed**, which:

* Speeds up initial page load
* Reduces bundle size
* Delays loading heavy pages until user visits them

React supports this via React.lazy() and Suspense.

### 9.2 Basic Setup

Instead of:

import About from './pages/About';

Do this:

import { lazy, Suspense } from 'react';

const About = lazy(() => import('./pages/About'));

Wrap it in a Suspense boundary:

<Suspense fallback={<div>Loading...</div>}>

<Route path="/about" element={<About />} />

</Suspense>

### 9.3 Apply Lazy Loading to All Routes

For App.jsx:

import { lazy, Suspense } from 'react';

const Home = lazy(() => import('./pages/Home'));

const About = lazy(() => import('./pages/About'));

const Contact = lazy(() => import('./pages/Contact'));

const NotFound = lazy(() => import('./pages/NotFound'));

<Route element={<MainLayout />}>

<Route

path="/"

element={

<Suspense fallback={<div>Loading Home...</div>}>

<Home />

</Suspense>

}

/>

<Route

path="/about"

element={

<Suspense fallback={<div>Loading About...</div>}>

<About />

</Suspense>

}

/>

<Route

path="/contact"

element={

<Suspense fallback={<div>Loading Contact...</div>}>

<Contact />

</Suspense>

}

/>

<Route

path="\*"

element={

<Suspense fallback={<div>Loading...</div>}>

<NotFound />

</Suspense>

}

/>

</Route>

### 9.4 Tips for Effective Code Splitting

* Use **lazy loading for large or infrequent pages** (e.g., dashboard, settings)
* Avoid wrapping every component in its own Suspense unless needed
* Use a centralized fallback UI if possible (e.g., a loading spinner)

Now your app loads faster and handles unknown routes gracefully.