**🛣️ Traditional vs React Routing**

**📊 Comparison**

**🔄 Traditional Routing**

* Server-side routing
* Page reloads on navigation
* New HTTP request for each page
* Fresh page load each time

**⚛️ React Routing**

* Client-side routing
* No page reloads
* Smooth transitions
* Better user experience
* Maintains state between routes

**🛠️ Setup**

**📦 1. Installation**

npm install react-router-dom

**🔧 2. Basic Setup (main.jsx)**

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

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

<BrowserRouter>

<App />

</BrowserRouter>

);

**🔗 Creating Navigation**

**📋 Basic Navbar Component**

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

function Navbar() {

return (

<nav className="navbar">

<div className="logo">

<Link to="/">MyShop</Link>

</div>

<ul className="nav-links">

<li><NavLink to="/">Home</NavLink></li>

<li><NavLink to="/products">Products</NavLink></li>

<li><NavLink to="/about">About</NavLink></li>

<li><NavLink to="/contact">Contact</NavLink></li>

</ul>

</nav>

);

}

export default Navbar;

**🎨 Styling Active Links**

/\* Styling for NavLink's active class \*/

.nav-links .active {

color: #007bff;

font-weight: bold;

}

.navbar {

display: flex;

justify-content: space-between;

padding: 1rem;

background-color: #f8f9fa;

}

.nav-links {

display: flex;

gap: 1rem;

list-style: none;

}

**🔀 Setting Up Routes**

**📝 Basic Route Setup (App.jsx)**

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

import Navbar from './components/Navbar';

import Home from './pages/Home';

import Products from './pages/Products';

import ProductDetails from './pages/ProductDetails';

import NotFound from './pages/NotFound';

function App() {

return (

<>

<Navbar />

<Routes>

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

<Route path="/products" element={<Products />} />

<Route path="/products/:id" element={<ProductDetails />} />

<Route path="\*" element={<NotFound />} />

</Routes>

</>

);

}

**🔄 Link vs NavLink**

**🔗 Link Component**

// Basic navigation without active states

<Link to="/products">Products</Link>

**🔗 NavLink Component**

// Navigation with active states

<NavLink

to="/products"

className={({ isActive }) => isActive ? 'active' : ''}

>

Products

</NavLink>

**🛍️ Products Page**

**📋 Basic Products Page**

function Products() {

const products = [

{ id: 1, name: "Laptop", price: 999 },

{ id: 2, name: "Smartphone", price: 699 },

{ id: 3, name: "Headphones", price: 199 }

];

return (

<div className="products-grid">

{products.map(product => (

<Link

to={`/products/${product.id}`}

key={product.id}

className="product-card"

>

<h3>{product.name}</h3>

<p>${product.price}</p>

</Link>

))}

</div>

);

}

**🎨 Styling Products**

.products-grid {

display: grid;

grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));

gap: 1rem;

padding: 1rem;

}

.product-card {

border: 1px solid #ddd;

padding: 1rem;

border-radius: 8px;

text-decoration: none;

color: inherit;

transition: transform 0.2s;

}

.product-card:hover {

transform: translateY(-5px);

}

**🔄 Dynamic Routing**

**📋 Product Details Page**

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

function ProductDetails() {

const { id } = useParams();

const navigate = useNavigate();

// Simulated product data

const product = {

id,

name: `Product ${id}`,

description: "Product description",

price: 999

};

return (

<div className="product-details">

<h2>{product.name}</h2>

<p>{product.description}</p>

<p>${product.price}</p>

<button onClick={() => navigate(-1)}>Go Back</button>

</div>

);

}

**🔄 Navigation with useNavigate**

**🔄 Programmatic Navigation**

function CheckoutButton() {

const navigate = useNavigate();

const handleCheckout = () => {

// Perform some action

navigate('/checkout');

// Or go back

// navigate(-1);

// Or replace current route

// navigate('/success', { replace: true });

};

return <button onClick={handleCheckout}>Checkout</button>;

}

**❌ 404 Page**

**📋 Not Found Component**

function NotFound() {

const navigate = useNavigate();

return (

<div className="not-found">

<h1>404 - Page Not Found</h1>

<p>The page you're looking for doesn't exist.</p>

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

Return to Home

</button>

</div>

);

}

**🔒 Protected Routes in React Router**

**🔑 1. Basic Authentication (Login Page)**

function Login() {

const [isLoggedIn, setIsLoggedIn] = useState(

Boolean(localStorage.getItem('authToken'))

);

const handleLogin = () => {

localStorage.setItem('authToken', 'yourDummyToken123');

setIsLoggedIn(true);

};

const handleLogout = () => {

localStorage.removeItem('authToken');

setIsLoggedIn(false);

};

return (

<div style={{ textAlign: "center" }}>

<h1>

{isLoggedIn ? "You are logged in" : "You are not logged in"}

</h1>

{isLoggedIn ? (

<button onClick={handleLogout}>Log Out</button>

) : (

<button onClick={handleLogin}>Log In</button>

)}

</div>

);

}

**🛡️ 2. Protected Routes Implementation**

**🔄 Method 1: Without Outlet (Component Wrapper)**

// utils/ProtectedRoutes.jsx

function ProtectedRoutes({ children }) {

const authToken = localStorage.getItem('authToken');

return authToken ? children : <Navigate to="/login" />;

}

// App.jsx

function App() {

return (

<Routes>

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

<Route

path="/profile"

element={

<ProtectedRoutes>

<Profile />

</ProtectedRoutes>

}

/>

</Routes>

);

}

**🔄 Method 2: Using Outlet (Parent Route)**

// utils/ProtectedRoutes.jsx

function ProtectedRoutes() {

const authToken = localStorage.getItem('authToken');

return authToken ? <Outlet /> : <Navigate to="/login" />;

}

// App.jsx

function App() {

return (

<Routes>

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

<Route element={<ProtectedRoutes />}>

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

<Route path="/settings" element={<Settings />} />

</Route>

</Routes>

);

}

**🔐 3. Enhanced Implementation with Auth Context**

// context/AuthContext.jsx

const AuthContext = React.createContext(null);

function AuthProvider({ children }) {

const [isAuthenticated, setIsAuthenticated] = useState(

Boolean(localStorage.getItem('authToken'))

);

const login = () => {

localStorage.setItem('authToken', 'yourDummyToken123');

setIsAuthenticated(true);

};

const logout = () => {

localStorage.removeItem('authToken');

setIsAuthenticated(false);

};

return (

<AuthContext.Provider value={{ isAuthenticated, login, logout }}>

{children}

</AuthContext.Provider>

);

}

// Enhanced Protected Routes

function ProtectedRoutes() {

const { isAuthenticated } = useContext(AuthContext);

const location = useLocation();

return isAuthenticated ? (

<Outlet />

) : (

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

);

}

**🔄 4. Handling Redirect After Login**

function Login() {

const { login } = useContext(AuthContext);

const navigate = useNavigate();

const location = useLocation();

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

const handleLogin = () => {

login();

navigate(from, { replace: true });

};

return (

<div>

<h1>Login</h1>

<button onClick={handleLogin}>Log In</button>

</div>

);

}

**⏳ 5. Loading States**

function ProtectedRoutes() {

const [isLoading, setIsLoading] = useState(true);

const [isAuthenticated, setIsAuthenticated] = useState(false);

useEffect(() => {

checkAuthStatus()

.then(status => setIsAuthenticated(status))

.finally(() => setIsLoading(false));

}, []);

if (isLoading) {

return <div>Loading...</div>;

}

return isAuthenticated ? <Outlet /> : <Navigate to="/login" />;

}

**💡 Remember:**

**🔐 For Protected Routes:**

* Always use secure storage methods
* Handle token expiration
* Implement proper error handling
* Consider using HTTP-only cookies
* Add loading states
* Handle redirect after login
* Clean up on logout

**🛣️ For General Routing:**

* Use NavLink for navigation with active states
* Implement proper error boundaries
* Consider lazy loading for large applications
* Protect sensitive routes
* Handle loading and error states
* Use descriptive route names

🚀 Happy routing!