# LESSON 07
# Basics of React Router

## WEEK 02

# Introduction to React Router v7

- **What is React Router?**
  Standard library for routing in React applications.
- Enables navigation without full page reloads.
- **Why v7?**
  Enhanced features, framework mode, seamless React 18-19 bridge.
- **Key Features**
  Declarative routing, data loading, server-side rendering (SSR).

# Installation and Setup

- **Install React Router**
  Use npm to add react-router (no react-router-dom in v7).
- **Basic Setup**
  Wrap app with BrowserRouter or RouterProvider.
- **Dependencies**
  Requires React 18+, Node 20+.

# Declarative Routing

- **Definition**
  Define routes using components (Routes, Route).
- **Benefits**
  Intuitive, matches React's component model.
- **Syntax**
  <Route path="..." element={<Component />}>.

# Declarative Routing (Example)

```jsx
export default function DeclarativeRouting() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path='/' element={<Home />} />
        <Route path='/about' element={<About />} />
        <Route path='*' element={<NotFound />} />
      </Routes>
    </BrowserRouter>
  );
}
```

# Data Mode with RouterProvider

- **What is Data Mode?**
  External route config with data loading.
- **RouterProvider**
  Replaces BrowserRouter for advanced routing.
- **Benefits**
  Parallel data fetching, better SSR.

# Data Mode with RouterProvider (Example)

```
1  const router = createBrowserRouter([
2    { path: '/', element: <Home /> },
3    { path: '/about', element: <About /> },
4    { path: '*', element: <NotFound /> },
5  ]);
6
7  export default function DataRouting() {
8    return <RouterProvider router={router} />;
9  }
```

# Framework Mode

- **Definition**
  Full-stack capabilities with Vite plugin.
- **Features**
  SSR, code splitting, file-based routing.
- **Use Case**
  Build Next.js-like apps with React Router.

# Navigation with Link and NavLink

- **Link Component**
  Replaces <a> for SPA navigation.
- **NavLink**
  Adds active state styling.
- **Benefits**
  Prevents full page reloads.

```jsx
function Nav() {
  return (
    <nav>
      <ul>
        <li>
          <Link to='/'>Home</Link>
        </li>
        <li>
          <NavLink to='/about'
            style={({ isActive }) => (isActive ? { color: 'red' } : {})}
          >
            About
          </NavLink>
        </li>
      </ul>
    </nav>
  );
}
```

# Nested Routes

- **Definition**
  Routes within routes for layout sharing.
- **Outlet Component**
  Renders child route elements.
- **Use Case**
  Shared layouts (e.g., sidebar).

```jsx
function Layout() {
  return (
    <div style={{ padding: '20px', backgroundColor: '#fcff5b' }}>
      <Outlet />
    </div>
  );
}

export default function NestedRoutes() {
  return (
    <BrowserRouter>
      <Nav />
      <hr />
      <Routes>
        <Route path='/' element={<Layout />}>
          <Route index element={<Home />} />
          <Route path='about' element={<About />} />
        </Route>
        <Route path='*' element={<NotFound />} />
      </Routes>
    </BrowserRouter>
  );
}
```

# Dynamic Routing

- **Definition**
  Routes with URL parameters (e.g., /user/:id).
- **useParams Hook**
  Access URL parameters in components.

```
function User() {
  const { id } = useParams();
  return <h2>User ID: {id}</h2>;
}
export default function DynamicRouting() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path='/' element={<Home />} />
        <Route path='/user/:id' element={<User />} />
        <Route path='/search' element={<Search />} />
        <Route path='*' element={<NotFound />} />
      </Routes>
    </BrowserRouter>
  );
}
```

# Dynamic Routing

- **Definition**
  Routes with URL query string (e.g., /search?name=John&address=Seoul).
- **useSearchParams Hook**
  Returns a tuple of the current URL's URLSearchParams and a function to update them.
  Setting the search params causes a navigation.

```
1  let location = useLocation();
2  // search?name=John&address=Seoul
3  let [searchParams, setSearchParams] = useSearchParams();
4
5  let name = searchParams.get('name');
6  let address = searchParams.get('address');
```

# Programmatic Navigation

## ❖ useNavigate Hook

- ➢ Navigate programmatically (e.g., after form submission).
- ➢ Can replace history, push new state.

```
1  function Home() {
2    const navigate = useNavigate();
3    return (
4      <div>
5        <h2>Home Page</h2>
6        <button
7          onClick={() => {
8            // Programmatic navigation to the About page
9            // window.location.href = '/about';
10           navigate('/about');
11         }}
12       >
13         Go to About Page
14       </button>
15     </div>
16   );
```

# Data APIs - Data Loaders

- **Definition**
  Fetch data before rendering route.
- **Benefits**
  Parallel loading, avoids waterfalls.
- **Syntax**
  loader function in route config.

```
1  const router = createBrowserRouter([
2    {
3      path: '/',
4      element: <Home />,
5      loader: async () => {
6        const response = await fetch('/api/data');
7        return response.json();
8      },
9    },
10   { path: '/about', element: <About /> },
11   { path: '*', element: <NotFound /> },
12 ]);
13
14 function Home() {
15   const data = useLoaderData();
16   return <h2>Home Page: {JSON.stringify(data)}</h2>;
17 }
```

# Data APIs - Actions for Form Submission

- **Definition**
  Handle form submissions with route actions.
- **Form Component**
  Replaces HTML <form> for SPA.

```jsx
const router = createBrowserRouter([
  { path: '/', element: <Home /> },
  { path: '/submit-success', element: <SubmitSuccess /> },
  {
    path: '/submit',
    action: async ({ request }) => {
      const formData = await request.formData();
      const data = Object.fromEntries(formData);
      console.log('Form submitted:', data);
      return redirect('/submit-success');
    },
  },
  { path: '*', element: <NotFound /> },
]);

function Home() {
  return (
    <Form method='post' action='/submit'>
      <input type='text' name='name' placeholder='Your Name' required />
      <input type='email' name='email' placeholder='Your Email' required />
      <button type='submit'>Submit</button>
    </Form>
  );
}
```

# Error Handling

- **ErrorBoundary**
  Catch errors in routes.
- **useRouteError**
  Access error details.

```jsx
function ErrorBoundary() {
  const error = useRouteError();
  return <div>Error: {error.message}</div>;
}

export default function ErrorHandling() {
  return (
    <BrowserRouter>
      <Nav />
      <hr />
      <Routes>
        <Route path='/' element={<Home />} errorElement={ErrorBoundary} />
        <Route path='/about' element={<About />} />
        <Route path='*' element={<NotFound />} />
      </Routes>
    </BrowserRouter>
  );
}
```

# Lazy Loading Routes

❖ **Definition**
Load route code only when needed.

❖ **Benefits**
Reduces initial bundle size.

```
1  const router = createBrowserRouter([
2    { path: '/', element: <Home /> },
3    {
4      path: '/lazy',
5      lazy: async () => {
6        const { default: Lazy } = await import('./LazyComponent');
7        return { Component: Lazy };
8      },
9    },
10   { path: '/about', element: <About /> },
11   { path: '*', element: <NotFound /> },
12 ]);
13
14 export default function LazyLoadingRoutes() {
15   return <RouterProvider router={router} />;
16 }
```

# Suspense Integration

- **Definition**
  Load route code only when needed.
- **Benefits**
  Reduces initial bundle size.

```
function Home() {
  const data = useLoaderData();
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <Await resolve={data}>
        <h2>Data Loaded</h2>
      </Await>
    </Suspense>
  );
}

export default function SuspenseIntegration() {
  return <RouterProvider router={router} />;
}
```

# Protected Routes

- **Definition**
  Restrict access based on auth state.
- **Implementation**
  Use loaders or redirects.

```
let isAuthenticated = () => {
  return false;
};
const router = createBrowserRouter([
  { path: '/', element: <Home /> },
  { path: '/login', element: <Login /> },
  {
    path: '/private',
    element: <Private />,
    loader: () => (!isAuthenticated() ? redirect('/login') : null),
  },
  { path: '/about', element: <About /> },
  { path: '*', element: <NotFound /> },
]);

export default function ProtectedRoutes() {
  return <RouterProvider router={router} />;
}
```

# Performance Optimization

- **Techniques**
  Code splitting, lazy loading, tree-shaking.
- **Tools**
  Vite, Webpack integration.

```
const Lazy = React.lazy(() => import("./Lazy"));
const router = createBrowserRouter([{ path: "/lazy", element: <Lazy /> }]);
```