React Router

Many pages, no reloads: BrowserRouter, Routes and Route, Link vs a tags, the useParams hook for detail pages, and the 404 catch-all route.

Introduction

A plain React app is one page whose components swap in and out — but users expect real URLs: `/courses`, `/courses/5`, a back button that works, links they can share. **React Router** is the library that maps URLs to components while staying a single-page app: the address changes, the screen changes, and the page **never reloads**. *(Roman Urdu: React Router aik hi app ko kayi "pages" deta hai — URL badalta hai, screen badalti hai, reload kabhi nahi.)*

Key points

- **Install once:** `npm install react-router-dom` — then wrap the app in **`<BrowserRouter>`** (usually in `main.jsx`), the component that watches the URL.
- **Routes are a table:** `<Routes>` holds `<Route path="/courses" element={<Courses />} />` rows — path on the left, component on the right. React Router picks the row that matches the current URL.
- **`Link` replaces `a`:** `<Link to="/courses">` changes the URL *and* the screen without a reload. A plain `<a href>` throws the whole app away and reloads — the video's side-by-side demo shows the state loss it causes.

  (Roman Urdu: Link andar ka safar hai, a tag poori app ko dobara load kar deta hai.)
- **Dynamic segments carry ids:** `path="/courses/:id"` matches `/courses/5`, `/courses/42` — and the component reads the value with the **`useParams`** hook: `const { id } = useParams();`
- **The catch-all route handles 404s:** `<Route path="*" element={<NotFound />} />` placed last — any URL no other row claimed lands here instead of a blank screen.
- **The back button just works:** Router keeps a history stack in the browser, so back/forward behave like a normal website — a detail users notice instantly when it is missing.
- **Nested layouts come later naturally:** a shared navbar around changing pages is done by rendering the nav outside `<Routes>` (as the Weather App becomes a site) — no new concept needed today.

Code examples

Example 1 — The whole router in one small app

```
import { BrowserRouter, Routes, Route, Link } from "react-router-dom";

function Home() {
  return <h2>Home — welcome to Code Scope</h2>;
}
function Courses() {
  return <h2>Courses — React, JavaScript, CSS</h2>;
}
function NotFound() {
  return <h2>404 — this page does not exist</h2>;
}

function App() {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Home</Link> | <Link to="/courses">Courses</Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/courses" element={<Courses />} />
        <Route path="*" element={<NotFound />} />
      </Routes>
    </BrowserRouter>
  );
}
```

What it does: `BrowserRouter` watches the address bar, `Link`s move between URLs without reloads, and `Routes` swaps the page component to match. The `*` route catches every unknown address — type `/anything-else` and the 404 page, styled like your app, appears.

What you will see: clicking Courses changes the URL to `/courses` and the heading swaps — the navbar never flickers, no reload spinner, and the browser's back button returns Home.

Example 2 — Detail pages with useParams

```
import { Link, Route, Routes, useParams } from "react-router-dom";

const courses = [
  { id: 1, title: "React", lessons: 26 },
  { id: 2, title: "JavaScript", lessons: 26 },
];

function CourseList() {
  return (
    <ul>
      {courses.map((c) => (
        <li key={c.id}>
          <Link to={`/courses/${c.id}`}>{c.title}</Link>
        </li>
      ))}
    </ul>
  );
}

function CourseDetail() {
  const { id } = useParams();
  const course = courses.find((c) => c.id === Number(id));
  if (!course) return <p>Course not found.</p>;
  return (
    <div>
      <h2>{course.title}</h2>
      <p>{course.lessons} lessons in this course.</p>
    </div>
  );
}
```

What it does: One `CourseDetail` component serves every course: the list's links bake the id into the URL, the route `/courses/:id` claims it, and `useParams` hands the id to the component, which finds and shows that course. One component, infinite detail pages — the pattern behind every product page and profile page on the web.

What you will see: click "JavaScript" → the URL reads `/courses/2` and the page shows "JavaScript — 26 lessons"; paste `/courses/99` into the bar and you get the honest "Course not found."

Example 3 — Link vs a: the reload, demonstrated

```
function NavDemo() {
  return (
    <nav>
      {/* Wrong for internal pages: full reload, all state is lost */}
      <a href="/courses">Courses (a tag)</a>

      {/* Right: client-side travel, state survives */}
      <Link to="/courses">Courses (Link)</Link>
    </nav>
  );
}
```

What it does: The two links point at the same address with opposite mechanics. The `<a>` asks the browser for a fresh page — the entire React app restarts and every state (cart, form half-typed) is gone. The `<Link>` lets Router swap the screen in place; the app and its state never blink. External sites (another domain) still use `<a>` — the rule is only about travel *inside* your app.

What you will see: put a counter on the page, raise it, then click each link in turn — the `a` version resets it to 0; the Link version preserves it.

Quick recap

- **`<BrowserRouter>`** watches the URL, **`<Routes>`** / **`<Route>`** form the address→component table, **`<Link>`** travels without reloads.
- **`:id` segments + `useParams`** turn one component into every detail page; the **`path="*"`** route catches 404s like a designed page.
- Internal travel is always `Link`; `<a>` is only for leaving the app.

Common mistakes

1. **Using `<a href>` for internal pages** — every click reloads the app and wipes state. The fix: internal navigation is `Link` (or `NavLink` when it needs an active style), always.
2. **Forgetting the catch-all route** — an unknown URL renders a blank content area inside your navbar, which looks like a crash. The fix: `path="*"` last in the table, with a friendly 404 page and a link home.
3. **Reading params as strings in comparisons** — `useParams` values are always strings, so `c.id === id` silently fails for numeric ids. The fix: convert at the comparison — `Number(id)` — or store ids as strings in the data.

Practice

1. Grow Example 1 into a three-page mini-site: Home, About, Courses — with the nav outside `Routes`, a catch-all 404, and an active-page underline done with simple conditional styling on the current path (check `window.location.pathname` is NOT the way — find how Link-based navs highlight; a hint: look up `NavLink` in the Router docs and use it).
2. Add detail pages to your Project 5.12 Weather App thinking: routes `/` (search + current weather) and `/city/:name` (a page that reads the name from the URL and loads that city's weather on mount). The home page's quick-city buttons become `Link`s to `/city/Karachi` and friends.

Practice quiz

1. What does <BrowserRouter> do, and what do <Routes> and <Route> form?

- BrowserRouter renders pages, Routes fetches data
- BrowserRouter watches the URL, and Routes/Route form the address-to-component table
- BrowserRouter replaces the browser, Routes replaces Links
- They are the same component

Answer: BrowserRouter watches the URL, and Routes/Route form the address-to-component table

2. Why use <Link to="/courses"> instead of <a href="/courses"> inside the app?

- There is no difference
- Link changes the URL and the screen without a reload; a plain <a> throws the whole app away and reloads, wiping state
- Link is faster to type
- Anchors do not work in React

Answer: Link changes the URL and the screen without a reload; a plain <a> throws the whole app away and reloads, wiping state

3. A route path="/courses/:id" matches /courses/5. How does the component read the 5, and what type is it?

- With useState, as a number
- With the useParams hook, const { id } = useParams();, and it is always a string, so convert with Number(id) for numeric ids
- With props.id, as a number
- It cannot be read, it is only for display

Answer: With the useParams hook, const { id } = useParams();, and it is always a string, so convert with Number(id) for numeric ids

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/react-router · More free tools and notes at https://scopetreasure.com