Route Parameters & Navigation

Putting variable values inside a URL (route parameters), reading them with useParams, building nested routes with Outlet, and programmatic navigation with useNavigate -- with simple examples.

Intermediate 14 min

Last updated: August 18, 2026
Read online: https://www.learnforgex.com/en/topics/route-parameters-navigation

---

Route Parameters & Navigation

In React Router Basics, we worked with fixed URLs (`/courses` , `/about`). This lesson covers putting a variable value INSIDE a URL (like `/courses/java`), and changing pages from code without a link click.

Route Parameters: Reading Data from the URL

Instead of writing a separate `Route` for `/courses/java` and `/courses/react` , we can define part of the URL as a VARIABLE:

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

function CourseList() {
  return (
    <div>
      <h1>Courses</h1>
      <Link to="/courses/java">Java</Link>
      <Link to="/courses/react">React</Link>
    </div>
  );
}

function CourseDetail() {
  // The URL segment defined as ":courseSlug" is read as an object via
  // useParams() -- the key matches the name used in the Route (courseSlug).
  const { courseSlug } = useParams();

  return <h1>Course: {courseSlug}</h1>;
}

function RouteParamExample() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/courses" element={<CourseList />} />
        <Route path="/courses/:courseSlug" element={<CourseDetail />} />
      </Routes>
    </BrowserRouter>
  );
}
```

Writing `path="/courses/:courseSlug"` makes `:courseSlug` a **route parameter** -- in the URL `/courses/java`, `courseSlug` takes the value `"java"`. Inside the component, we read this value with the `useParams()` hook; the key on the returned object matches the name used in the Route (`courseSlug`).

Nested Routes and Outlet

Sometimes a page has an INNER section that changes based on the URL -- for example, a course page with a content area that changes depending on the selected topic:

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

function CourseLayout() {
  // This is the "parent" route matched by /courses/:courseSlug. The
  // <Outlet /> inside it specifies where the matched CHILD route
  // (if any) should be rendered.
  const { courseSlug } = useParams();

  return (
    <div>
      <h1>Course: {courseSlug}</h1>
      <Link to={`/courses/${courseSlug}/enum`}>Enum Topic</Link>
      <Outlet />
    </div>
  );
}

function TopicDetail() {
  const { courseSlug, topicSlug } = useParams();

  return (
    <p>
      Topic: {topicSlug} (course: {courseSlug})
    </p>
  );
}

function NestedRouteExample() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/courses/:courseSlug" element={<CourseLayout />}>
          {/* A nested Route -- it is only rendered at the <Outlet />
              position INSIDE CourseLayout, for a URL like
              /courses/java/enum. */}
          <Route path=":topicSlug" element={<TopicDetail />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
}
```

Writing one `Route` INSIDE another (`:topicSlug` inside the `:courseSlug` route) creates a nested structure. The `<Outlet />` we place inside the parent component (`CourseLayout`) marks EXACTLY where the matching child route should render -- without `Outlet` , the child route wouldn't appear anywhere.

Programmatic Navigation with useNavigate

`Link` always requires the user to CLICK something. Sometimes we want to change pages from code AS A RESULT of something -- clicking a button, finishing a calculation:

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

function CourseList() {
  const navigate = useNavigate();

  function handleSelect(courseSlug) {
    // Link always requires a click -- useNavigate(), on the other hand,
    // lets us CHANGE the URL from INSIDE a function (e.g. an event
    // handler).
    navigate(`/courses/${courseSlug}`);
  }

  return (
    <div>
      <button onClick={() => handleSelect("java")}>Go to Java</button>
      <button onClick={() => handleSelect("react")}>Go to React</button>
    </div>
  );
}

function CourseDetail() {
  return <h1>Course Detail</h1>;
}

function UseNavigateExample() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/courses" element={<CourseList />} />
        <Route path="/courses/:courseSlug" element={<CourseDetail />} />
      </Routes>
    </BrowserRouter>
  );
}
```

The `useNavigate()` hook gives us a `navigate` function; calling it from INSIDE an event handler changes the URL -- unlike `Link`, this isn't tied to a click, but to a CONDITION in the code.

Navigating After an Action

One of the most common uses of `useNavigate` is redirecting the user to another page after a form is submitted:

```
import { useState } from "react";
import { BrowserRouter, Routes, Route, useNavigate } from "react-router";

function NewCourseForm() {
  const [title, setTitle] = useState("");
  const navigate = useNavigate();

  function handleSubmit(event) {
    event.preventDefault();

    // A common pattern is to redirect the user to another page AFTER
    // the form is submitted (e.g. "saved").
    navigate("/courses");
  }

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        value={title}
        onChange={(event) => setTitle(event.target.value)}
        placeholder="Course title"
      />
      <button type="submit">Save</button>
    </form>
  );
}

function CourseList() {
  return <h1>Courses</h1>;
}

function NavigateAfterActionExample() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/new-course" element={<NewCourseForm />} />
        <Route path="/courses" element={<CourseList />} />
      </Routes>
    </BrowserRouter>
  );
}
```

The `onSubmit` + `preventDefault` pattern from Form Handling is the same here -- the only difference is that after the form is "submitted" (directly in this example, since there's no real save operation), we redirect the user to the course list with `navigate("/courses")`.

Going Back: navigate(-1)

`navigate` can also be given a NUMBER instead of a URL -- this is used to move forward or backward in browser history:

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

function CourseList() {
  return (
    <div>
      <h1>Courses</h1>
      <Link to="/courses/java">Java</Link>
    </div>
  );
}

function CourseDetail() {
  const navigate = useNavigate();

  function handleBack() {
    // navigate(-1) does the same thing as the browser's "back" button --
    // it goes one step back in the history. navigate(-2) goes two steps
    // back, and so on.
    navigate(-1);
  }

  return (
    <div>
      <h1>Course Detail</h1>
      <button onClick={handleBack}>Back</button>
    </div>
  );
}

function GoBackNavigateExample() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/courses" element={<CourseList />} />
        <Route path="/courses/:courseSlug" element={<CourseDetail />} />
      </Routes>
    </BrowserRouter>
  );
}
```

`navigate(-1)` does the same thing as the browser's "back" button: it goes back one step in history. Since this returns the user to "wherever they came from" instead of pinning them to a fixed page (like `/courses`), it's usually preferred for "Back" buttons.

Summary and Glossary

We can create a route parameter by marking part of a URL with `:name`, and read its value with `useParams()`. Writing one `Route` inside another and adding `<Outlet />` to the parent component lets us build nested routes. `useNavigate()` lets us change pages from inside an event handler or as a result of some action, without needing a click; `navigate(-1)` moves backward in browser history.

Glossary

Route Parameter — A part of a URL pattern marked with `:name` that takes a variable value in the real URL.

Nested Route — A Route written inside another Route, rendered only when the parent Route matches and only at the `<Outlet />` position.

Outlet — A component inside a parent route component that marks where the matching child route should render.

Programmatic Navigation — Changing pages from code (e.g. from an event handler) with `useNavigate()`, instead of clicking a link.

Practical Project

There's a real, runnable example project that brings together the concepts from this category (React Router Basics, Route Parameters & Navigation): **Routing Demo** -- a small course-browsing app that mirrors the learning platform's own course structure (`/courses`, `/courses/java`, `/courses/java/enum`, and similar).

It shows defining pages with `BrowserRouter` + `Routes` + `Route`, navigating with `Link`/ `NavLink`, reading data from the URL with route parameters (`useParams`), nested routes with `Outlet`, and programmatic navigation with `useNavigate`, all working together. You can download it and run it yourself, and read through the code line by line:

```
git clone https://github.com/cdurgun/react-course-projects.git
cd react-course-projects
npm install
cd projects/routing
npm run dev
```

The `react-course-projects` repo uses **npm workspaces** -- `npm install` only needs to run once, at the repo root, and every project folder shares the same dependencies (no separate `node_modules` per folder). If you've already run `npm install` at the root, you can just `cd react-course-projects/projects/routing` and run `npm run dev`.