# Day 16 — Nested Routes & Dynamic Params (React Router)

## What you’ll learn in this canvas

* Why nested routes are useful
* How React Router v6 models nested routes
* Outlet, useParams, index routes, and wildcards (\*)
* How to read query/search params (useSearchParams) briefly
* Best practices and gotchas
* A hands-on exercise at the end

## 1) Why nested routes?

* Real apps have layouts: header, sidebar, content area. Nested routes let you declare which child routes render inside a parent layout (via <Outlet/>).
* They keep the routing structure in the code matching the UI tree — easier to reason about and maintain.
* Example use cases: product lists with a product detail page, dashboards with sub-sections, profile pages with tabs.

## 2) Core concepts & API (short reference)

* <BrowserRouter> — wraps your app to enable client-side routing.
* <Routes> and <Route> — define routes; <Route> can be nested to express parent/child relationship.
* <Outlet /> — placeholder in parent UI where the child route component will render.
* useParams() — hook that returns route parameters (like :id).
* index route — a child <Route index element={...} /> acts as the default child for its parent.
* \* wildcard — match anything (useful for 404 pages or nested catch-alls).
* useNavigate() / <Link> — navigate programmatically or with links.
* useSearchParams() — read and write URL query parameters (e.g., ?q=term).

## 3) Concrete example (Products → Product Detail → Reviews)

**Routing tree (logical)**

/ -> Layout  
 index -> Home  
 /products -> ProductsList (shows all products)  
 /:productId -> ProductDetail (shows one product)  
 index -> Overview (default child)  
 /reviews -> Reviews  
 /about -> About  
 \* -> NotFound

**App.js (router setup)**

import { BrowserRouter, Routes, Route } from 'react-router-dom';  
import Layout from './Layout';  
import Home from './Home';  
import Products from './Products';  
import ProductDetail from './ProductDetail';  
import Overview from './Overview';  
import Reviews from './Reviews';  
import About from './About';  
import NotFound from './NotFound';  
  
export default function App(){  
 return (  
 <BrowserRouter>  
 <Routes>  
 <Route path="/" element={<Layout/>}>  
 <Route index element={<Home/>} /> {/\* index = default child \*/}  
 <Route path="products" element={<Products/>}>  
 <Route index element={<ProductsList/>} /> {/\* /products \*/}  
 <Route path=":productId" element={<ProductDetail/>}>  
 <Route index element={<Overview/>} /> {/\* /products/:productId \*/}  
 <Route path="reviews" element={<Reviews/>} /> {/\* /products/:productId/reviews \*/}  
 </Route>  
 </Route>  
 <Route path="about" element={<About/>} />  
 <Route path="\*" element={<NotFound/>} />  
 </Route>  
 </Routes>  
 </BrowserRouter>  
 );  
}

**Layout.js (parent layout — shows header + Outlet)**

import { Outlet, NavLink } from 'react-router-dom';  
export default function Layout(){  
 return (  
 <div>  
 <header>  
 <NavLink to="/">Home</NavLink> | <NavLink to="/products">Products</NavLink> | <NavLink to="/about">About</NavLink>  
 </header>  
 <main>  
 {/\* child routes render here \*/}  
 <Outlet />  
 </main>  
 </div>  
 )  
}

**Products.js (parent for product routes)**

import { Outlet, Link } from 'react-router-dom';  
  
export default function Products(){  
 const products = [ {id:1, name:'Chair'}, {id:2, name:'Table'} ];  
 return (  
 <div>  
 <h2>Products</h2>  
 <ul>  
 {products.map(p => (  
 // Link to a nested path — relative to parent (/products)  
 <li key={p.id}><Link to={`${p.id}`}>{p.name}</Link></li>  
 ))}  
 </ul>  
  
 {/\* where /products/:productId children appear \*/}  
 <Outlet />  
 </div>  
 )  
}

\*\*ProductDetail.js (uses useParams to read :productId)

import { useParams, Link, Outlet } from 'react-router-dom';  
import { useEffect, useState } from 'react';  
  
export default function ProductDetail(){  
 const { productId } = useParams(); // NOTE: this returns strings  
 const [product, setProduct] = useState(null);  
  
 useEffect(() => {  
 // Example: fetch product details from API using productId  
 // fetch(`/api/products/${productId}`)  
 // .then(r => r.json())  
 // .then(setProduct);  
  
 // For demo, fake it:  
 setProduct({ id: productId, name: 'Demo Product ' + productId, description: 'Detailed description' });  
 }, [productId]); // re-run when param changes  
  
 if(!product) return <p>Loading...</p>;  
  
 return (  
 <div>  
 <h3>{product.name}</h3>  
 <p>{product.description}</p>  
  
 {/\* links to nested child routes (relative) \*/}  
 <Link to=".">Overview</Link> | <Link to="reviews">Reviews</Link>  
  
 {/\* Outlet for nested children (Overview or Reviews) \*/}  
 <Outlet />  
 </div>  
 )  
}

**Overview.js** and **Reviews.js** are simple components — they render content for the nested routes.

## 4) Important notes & gotchas

* useParams() returns **strings**. Convert to number if needed: const id = Number(productId);
* When linking inside nested routes, **relative paths** (e.g., to="reviews") are handy — they resolve relative to the current route.
* The index child route shows when parent path is matched exactly (like /products/1 renders index inside ProductDetail).
* Use <Link to="."> to link to the parent index child.
* A wildcard \* inside a nested route can be used to catch anything deeper under that parent.
* React Router v6 matches routes by the nested structure, not by order — nested <Route>s reflect UI nesting.

## 5) Reading query/search params (optional but useful)

import { useSearchParams } from 'react-router-dom';  
  
function SearchPage(){  
 const [searchParams, setSearchParams] = useSearchParams();  
 const q = searchParams.get('q') || '';  
  
 // update: setSearchParams({ q: 'term' });  
}

Query params are great for filters, pagination, or preserving UI state in URL.

## 6) Exercise (15–25 minutes)

Build the small demo below to practice nested routes and params.

### Task: Products app with nested product detail

1. Create routes as in the example above (Layout, Products, ProductDetail, Overview, Reviews).
2. In Products, provide a list of 5 fake products with IDs and names and link each to its detail page.
3. In ProductDetail, read the productId with useParams and display the product name. Use useEffect to simulate fetching details when productId changes.
4. Create nested child routes Overview (index) and Reviews and link between them inside ProductDetail.
5. Ensure that navigating between different product IDs updates the detail view (i.e., useEffect dependency works).
6. Bonus: add a Back to products <Link> and show query param support (e.g., /products?q=chair — use useSearchParams to read it).

**Deliverable**: paste your App.js, Products.js, and ProductDetail.js here and I’ll review.

## 7) Quick checklist for debugging

* If child route doesn’t render: did you include <Outlet/> in the parent?
* If useParams() is empty: check that your Route path contains :paramName.
* If navigation isn’t relative as expected: check whether you used to="/absolute" instead of to="relative".

When you’re ready I can also scaffold starter files (copy/paste ready) for this demo so you can run it immediately.