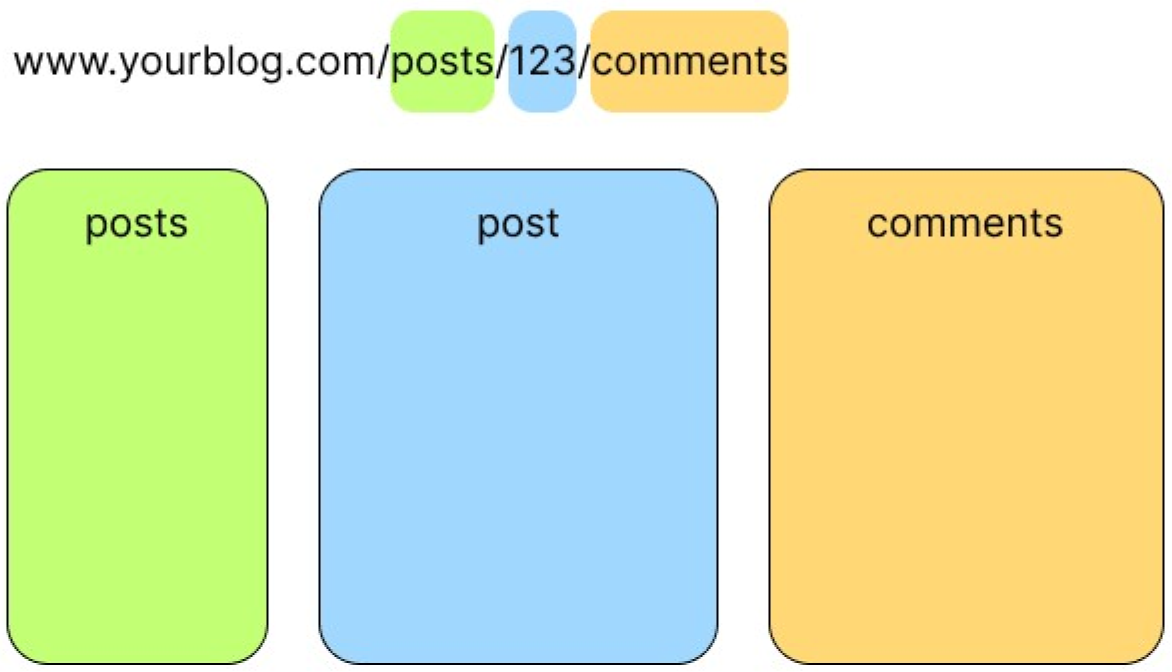
tywenk
Posted on Apr 13, 2022

How to Use Nested Routes in React Router 6

#react #javascript #tutorial #router

React Router version 6 makes it easy to nest routes. Nested routes enables you to have multiple components render on the same page with route parity. This is useful for app experiences where you want the user to be able to "drill down" into content and not lose their way, such as in forums or blogs.



Installing React Router

To get started install React Router 6 into your React app. In your app's directory open a terminal and input:

```
npm install react-router-dom@6
```

After it's installed, go to the top level component of your app. In my case, I like to leave `index.jsx` clean and standard, so I put my routes in `App.jsx`, which is the next highest component.

At the top of `App.jsx` import the required components:

`import { BrowserRouter as Router, Routes, Route } from "react-router-dom"`.

It is standard practice to import `BrowserRouter as Router` because `BrowserRouter` is too long to type!

Basic Routing

From here you can begin to set up your routes. An empty shell of nested routes looks like this, with the outermost component `<Router>` wrapping a `<Routes>` (plural!) which then wraps multiple `<Route>` (singular!):

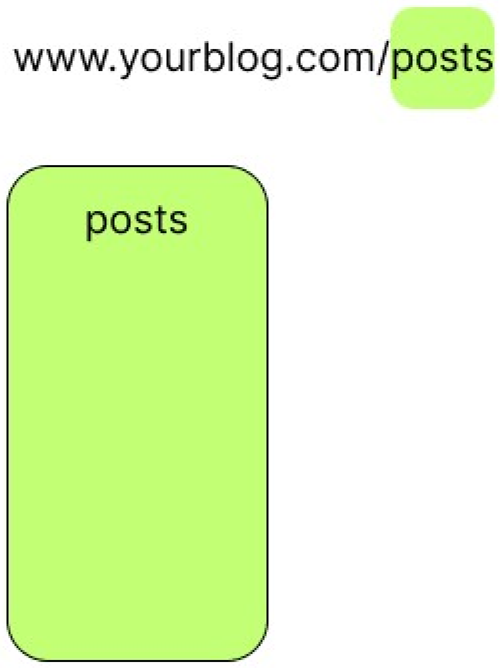
```
import { BrowserRouter as Router, Routes, Route } from "react-router-dom"

export default function App() {
    return (
        <Router>
            <Routes>
                <Route>
                    <Route />
                    <Route />
                </Route>
            </Routes>
        </Router>
    )
}
```

The `<Route>` component takes in a `path` and `element` prop. The `path` value holds the path of the route. The `element` value holds a pointer to a component or page.

`<Route path='/' element={<Home />} />` is a route that points to a `Home` component at the base route of `https://yourblog.com/`.

Let's try and make an Blog app which displays posts in a nested fashion. We want to see a list of post *previews* in a `<Posts/>` component. Eventually we want to be able to click a post preview to show a new route of the full post content. We also want to click a button in `<Posts/>` that will bring up a new post form in the same place as click a button in `<Posts/>` that will bring up a new post form in the same place as `<Post/>` Let's start by adding few routes to `App.jsx`.

```
import { BrowserRouter as Router, Routes, Route } from "react-router-dom"

export default function App() {
    return (
        <Router>
            <Routes>
                <Route path='/' element={<Home />} />
                <Route path='about' element={<About />} />
                <Route path='posts' element={<Posts />} />
            </Routes>
        </Router>
    )
}
```



> You will have to import the Home, About, and Posts components at the top of your page for this to work.

These separate routes now each point to a different component. These routes are *not* yet nested. If we were to visit `https://yourblog.com/about` the webpage would render only what is inside the `<About />` component. If we were to visit `https://yourblog.com/posts` the webpage would render only what is inside the `<Posts />` component.

/posts the webpage would render only what is inside the <Posts /> component.

With the above App we have the following routes:

- `"/"`
- `"/about"`
- `"/posts"`

Note that in your `path` props we don't write `/about` and instead write `about`. The forward slash `/` in path names is implicit in React Router 6.

Nested Routing

But we want nested routes! It's this easy:

```
export default function App() {
    return (
        <Router>
            <Routes>
                <Route path='/' element={<Home />} />
                <Route path='about' element={<About />} />
                <Route path='posts' element={<Posts />}>
                    <Route path='new' element={<NewPost />} /> {/*A nested route!*/}
                    <Route path=':postId' element={<Post />} /> {/*A nested route!*/
                </Route>
            </Routes>
        </Router>
    )
}
```

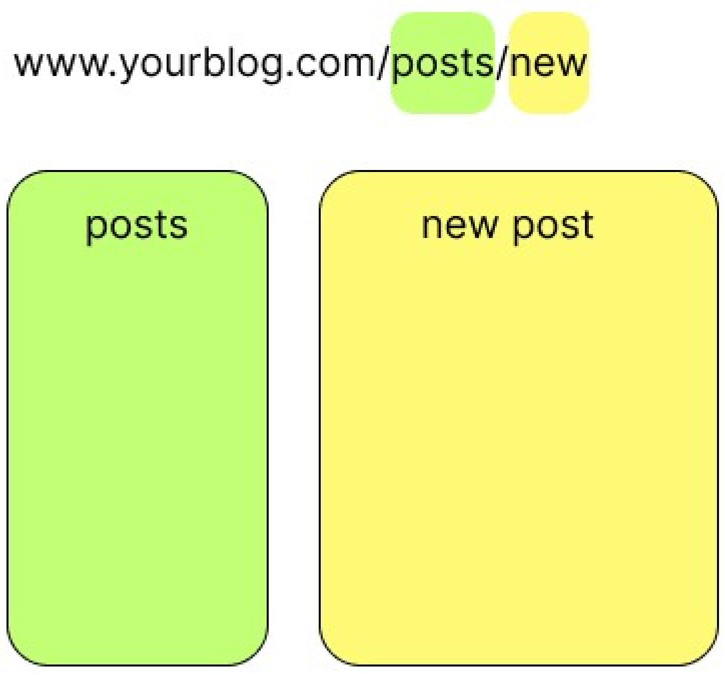
In between a `Route` component you can embed another Route: `<Route>{/* Children routes go here*/}</Route>`

Now our route structure looks like this:

- `"/"`
- `"/about"`
- `"/posts"`
- `"/posts/new"`
- `"/posts/123"`

Now we can learn how these nested routes can pass information to each other and ensure they display in a proper "nested" way.

Nesting routes with <Outlet/>



`/posts/new` points to a page to create new posts. This page will likely contain a controlled form to submit a new post.

When we head to `https://yourblog.com/posts/new` we will see both the `<Posts />` component AND the `<NewPost />` component. This is the magic of nesting routes! We are rendering two difference components with one descriptive path on the same page. Super cool!

At this point, if you're following along, you might not see the `<NewPost />` component appear. This is because we need to tell the parent component, `<Posts />` where to render its children. React Router provides a component called `Outlet` that renders a route's child component.

A `<Posts />` component might look like:

```
import { Outlet } from "react-router-dom"

export default function Posts() {
        <div>
            <h1>List of posts go here!</h1>
            <Outlet />
        </div>
    )
}
```

`<Outlet />` behaves a bit like `props.children` in standard React. `<Outlet />` is the placeholder location for where the nested children routes will be rendered.

You might be wondering how to pass props from a parent route to a child route. React Router 6 has a native prop of `Outlet` called `context`, which behind the scenes is a React context provider. `context` accepts an array of states.

In order to for a child route to accept the context, the child component must use React Router's provided hook `useOutletContext`. This whole getup might look like:

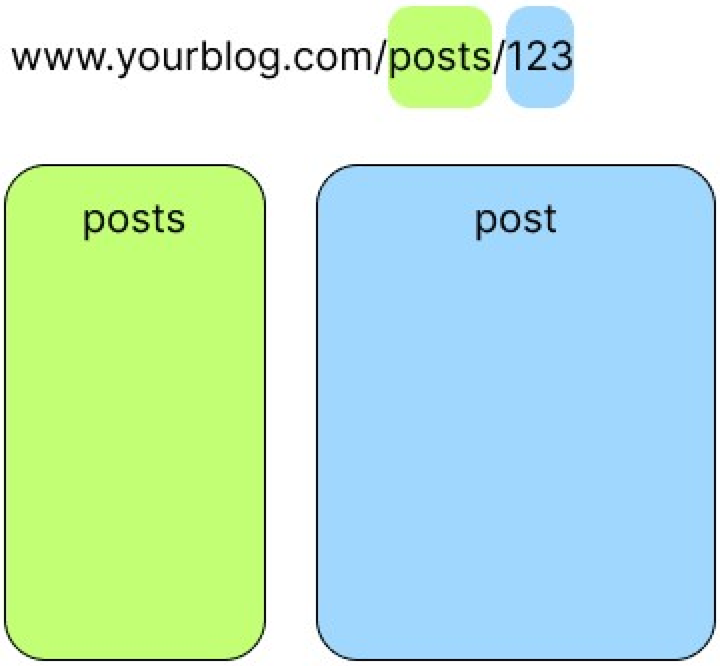
```
// Posts.jsx, the parent
import { Outlet } from "react-router-dom"

export default function Posts() {
  const [currentUser, setCurrentUser] = React.useState([/*array of post content*/])

    return (
        <div>
            <h1>List of posts go here!</h1>
            <Outlet context={[currentUser]}/>
        </div>
    )
}

//NewPost.jsx, the child
import { useOutletContext } from "react-router-dom"

export default function NewPost() {
  const [currentUser] = useOutletContext()

    return (
        <div>
            <h1>Welcome {currentUser}, write a new post!</h1>
      <form/>
        </div>
    )
}
```

In order for outlet:

- The parent must import `Outlet`
- The parent must embed `<Outlet/>` with `context=[state]` in the render
- The child must import `useOutletContext`
- The child must destructure the array with `useOutletContext()`. Order matters, unlike with props and object destructuring.

Now, on to the next nested route, `:postId`.

Nested Routes and useParams



`/posts/123` will display a specific post. Our `path` prop for this Route is a parameter, dentote by the prefix `":"` in the path:

```
<Route path=':postId' element={<Post />} />
```

In the `<Post />` component we can get the parameter `123` by importing useParams.

```
import { useParams } from "react-router-dom"
```

Then in the body of your functional component, call useParams.

```
let params = useParams()
```

Now within the scope of the component you can access `123`, or whatever post id is passed in the route, by called `params.postId`.

```
import { useParams } from "react-router-dom"

function Post() {
    let params = useParams()

    return <h1>{params.postId}</h1>
}
```

Placeholder Component with Index

Nesting routes can also go deeper than one level. Let's nest another route in `<Post />` to display comments.

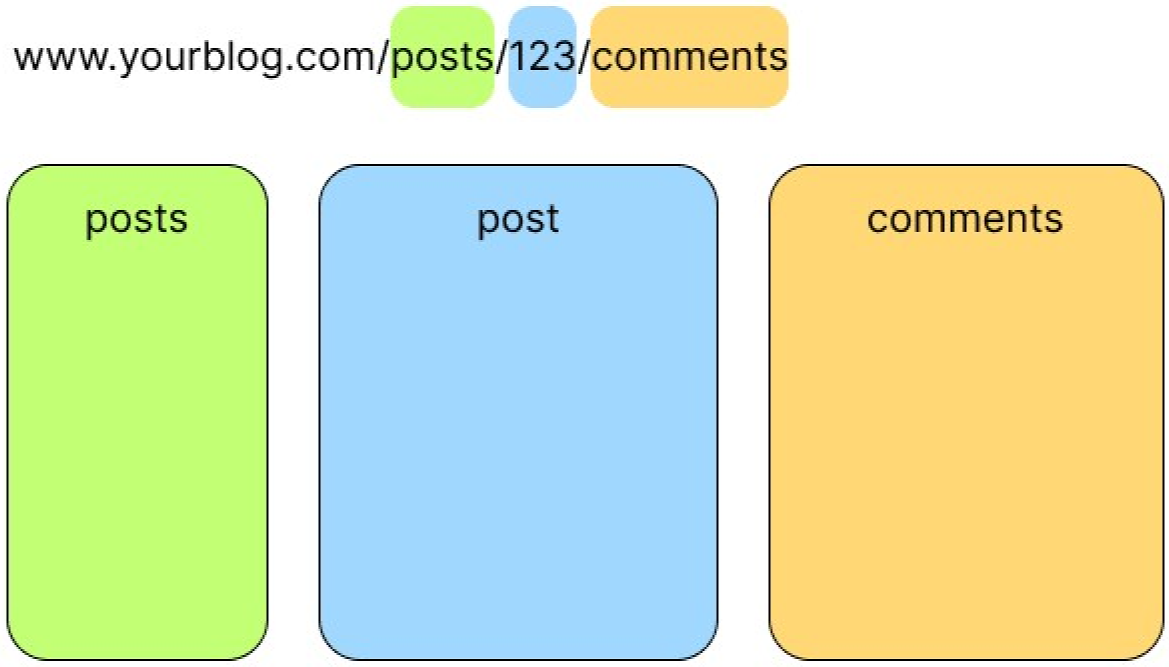
```
export default function App() {
    return (
        <Router>
            <Routes>
                <Route path='/' element={<Home />} />
                <Route path='about' element={<About />} />
                <Route path='posts' element={<Posts />}>
                    <Route path='new' element={<NewPost />} />
                    <Route path=':postId' element={<Post />}>
                        <Route index element={<PostIndex />} />
                        <Route path='comments' element={<Comments />} />
                    </Route>
                </Route>
            </Routes>
        </Router>
    )
}
```

Now our route structure looks like this:

- `"/"`
- `"/about"`

- `"/posts"`
- `"/posts/new"`
- `"/posts/123"`
- `"/posts/123/comments"`

As expected, `<Comments />` now renders as a child route of `<Post />`. Of course, now in `<Post />` we have to add an `<Outlet />` in order to render child routes. This will look something like:



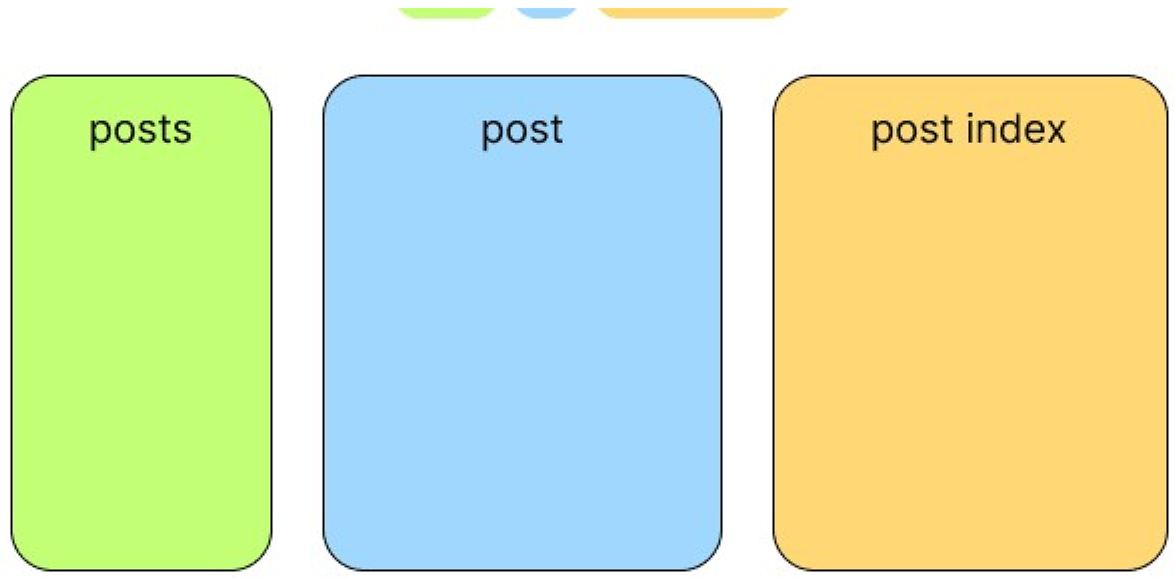
But wait, we added two routes, why is there no path for `<PostIndex />`?

`<Route index element={<CommentEmpty />} />` has no path prop and instead is given an `index` value.

`index` declares this route as the default child route to render in the parent's `Outlet` when there is no other child to render. `index` is the default placeholder content for an empty path.

In this case, when there is no `/comments` path, the render will look like:

www.yourblog.com/posts/123/



You could put a clean watermark in here for nicer UX, or put some interesting statistics about the Post, it's up to you! But you have the option to have something there and not some buggy-looking white space.

Navigating Between Nested Routes

Finally, React Router provides a handy way to navigate up and down nested routes.

The library provides a hook called `useNavigate()` that allows you to develop programatic path changes.

To use it import it at the top of a component where you want a user to be able to navigate out of.

```
import { useNavigate } from "react-router-dom"
```

Then instantiate it in the body of the component.

```
let navigate = useNavigate()
```

Now you can use `navigate({/*options*/})` to render a different route.

```
import { useNavigate } from "react-router-dom"

function Post() {
    let navigate = useNavigate()
    return (
        <div>
            <button onClick={() => navigate("./")}>Go Back One</button>
            <button onClick={() => navigate("../")}>Go Back Two</button>
        </div>
    )
}
```

Some useful useNavigate() options include:

- useNavigate("./") to go up a nested path
- useNavigate(-1) to go back in history, as if the user clicked the back button in a browser
- useNavigate("/pathname") to go to a specific path

End

Hopefully this helps you develop some cool nested router apps!

Resources

Helpful blog that goes more in depth: (https://ui.dev/react-router-nested-routes)

React Router 6 docs: https://reactrouter.com/docs/en/v6/api

Top comments (2)

Sophia Nelson • Jun 1 '22

Really helpful.

😺🕴😺 🕵🍾🕵 • Sep 24 '22

Awesome!

Code of Conduct • Report abuse