Introduction

React Router is React's standard routing library. When you need to navigate through a React application with multiple views, you'll need a router to manage the URLs. React Router does that while keeping your application's UI and URL in sync.

One of the main features of React is that it allows the creation of single-page applications (**SPA**s) that are rendered on the client's side. An SPA might have multiple **views** (aka **pages**), and unlike the conventional multi-page apps, navigating through these views should not result in the entire page being reloaded. Instead, we want the views to be rendered inline within the current page. The end user, who's accustomed to multi-page apps, expects the following features to be present in an SPA:

- Each view in an application should have a URL that uniquely specifies that view. This is so that the user can bookmark the URL for reference at a later time — e.g. www.example.com/products.
- The browser's back and forward button should work as expected.
- The dynamically generated nested views should preferably have a URL of their own too — e.g. www.example.com/products/shoes/101, where 101 is the product id.

Introduction

Routing is the process of keeping the browser URL in sync with what's being rendered on the page. React Router lets you handle routing **declaratively**. The declarative routing approach allows you to control the data flow in your application, by saying "the route should look like this":

```
<Route path="/about" element={<About/>}/>
```

You can place your `<Route>` component anywhere that you want your route to be rendered. Since `<Route>`, `<Link>` and all the other React Router API that we'll be dealing with are just components, you can easily get used to routing in React.

One remark before getting started. There's a common misconception that React Router is an official routing solution developed by Facebook. In reality, it's a third-party library that's widely

popular for its design and simplicity. If your requirements are limited to routers for navigation, you could implement a custom router from scratch without any hassle. However, understanding the basics of React Router will give you better insights into how a router should work.

Overview

In this Super Skill, we'll cover four main topics. First, we'll be setting up React and React Router using npm. Then we'll jump right into React Router basics. You'll find different code demonstrations of React Router in action. The examples covered include:

- Basic navigational routing.
- Nested routing.
- Nested routing with path parameters.
- Protected routing.

Let's get started!

Setting up React Router

Let's assume you already have a development environment up and running where you've used Create React App to generate the files required for creating a basic React project. The default directory structure generated by Create React App should look something like this:

```
react-routing-super-skill
├── README.md
├── node_modules
├── package.json
```

```
├── .gitignore
├── public
│   ├── favicon.ico
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
└── src
    ├── App.css
    ├── App.js
    ├── App.test.js
    ├── index.css
    ├── index.js
    ├── logo.svg
    └── serviceWorker.js
```

The React Router library comprises three packages: `react-router`, `react-router-dom`, and `react-router-native`. `react-router` is the core package for the router, whereas the other two are environment-specific. You should use `react-router-dom` if you're building a website, and `react-router-native` if you're on a mobile app development environment using React Native.

Let's use npm to install `react-router-dom`:

```
npm install --save react-router-dom
```

‹ Previous

next ›