CHAPTER 21

Using URL Routing

At the moment, the selection of content displayed to the user is controlled by the application's state data. Some of that state data is specific to a single component, such as the `Selector` component, which manages the choice between products and supplier data. The rest of the data is in the Redux data store and is used by the connected components to decide whether the data table or editor components are required and to obtain the data to populate those components' content.

In this chapter, I introduce a different approach to structuring the application, which is to select content based on the browser's URL, known as *URL routing*. Instead of button elements whose event handlers dispatch Redux actions, I am going to render anchor elements that navigate to new URLs and respond to those URLs by selecting content and presenting it to the user. For complex applications, URL routing can make it easier to structure a project and make it easy to scale up and maintain features. Table 21-1 puts URL routing in context.

Table 21-1. *Putting URL Routing in Context*

Question	Answer
What is it?	URL routing uses the browser's current URL to select the content presented to the user.
Why is it useful?	URL routing allows applications to be structured without the need for shared state data, which becomes encoded in the URL, which also makes it easier to change the structure of an application.
How is it used?	Navigation elements are rendered that change the browser's URL without triggering a new HTTP request. The new URL is used to select the content presented to the user.
Are there any pitfalls or limitations?	Thorough testing is required to ensure that all of the URLs to which the user can navigate are handled correctly and display the appropriate content.
Are there any alternatives?	URL routing is entirely optional, and there are other ways to compose an application and its data, as demonstrated in earlier chapters.

A. Freeman, *Pro React 16*, https://doi.org/10.1007/978-1-4842-4451-7_21

Table 21-2 summarizes the chapter.

Table 21-2. Chapter Summary

Problem	Solution	Listing
Create a navigation element	Use the `Link` component	4, 13
Respond to navigation	Use the `Route` component	5–6
Match a specific URL	Use the `Route` component's `exact` prop	7
Match several URLs	Specify the URLs as an array in the `Route` component's `path` prop or use a regular expression	8–9
Select a single route	Use the `Switch` component	10
Define a fallback route	Use the `Redirect` component	11, 12
Indicate the active route	Use the `NavLink` component	14, 15
Choose the mechanism used to represent the route in the URL	Select the router component	16

Preparing for This Chapter

In this chapter, I continue using the `productapp` project created in Chapter 18 and used most recently in Chapter 20. To prepare for this chapter, open a new command prompt, navigate to the `productapp` folder, and run the command shown in Listing 21-1 to add a package to the project. The React Router package is available for a range of application types. The package installed in Listing 21-1 is for web applications.

■ **Tip** You can download the example project for this chapter—and for all the other chapters in this book—from `https://github.com/Apress/pro-react-16`.

Listing 21-1. Adding a Package to the Project

```
npm install react-router-dom@4.3.1
```

To simplify the content presented to the user, I have removed some of the content rendered by the `App` component, as shown in Listing 21-2.

Listing 21-2. Simplifying Content in the App.js File in the src Folder

```
import React, { Component } from "react";
import { Provider } from "react-redux";
import dataStore from "./store";
import { Selector } from "./Selector";
import { ProductDisplay } from "./ProductDisplay";
import { SupplierDisplay } from "./SupplierDisplay";
```

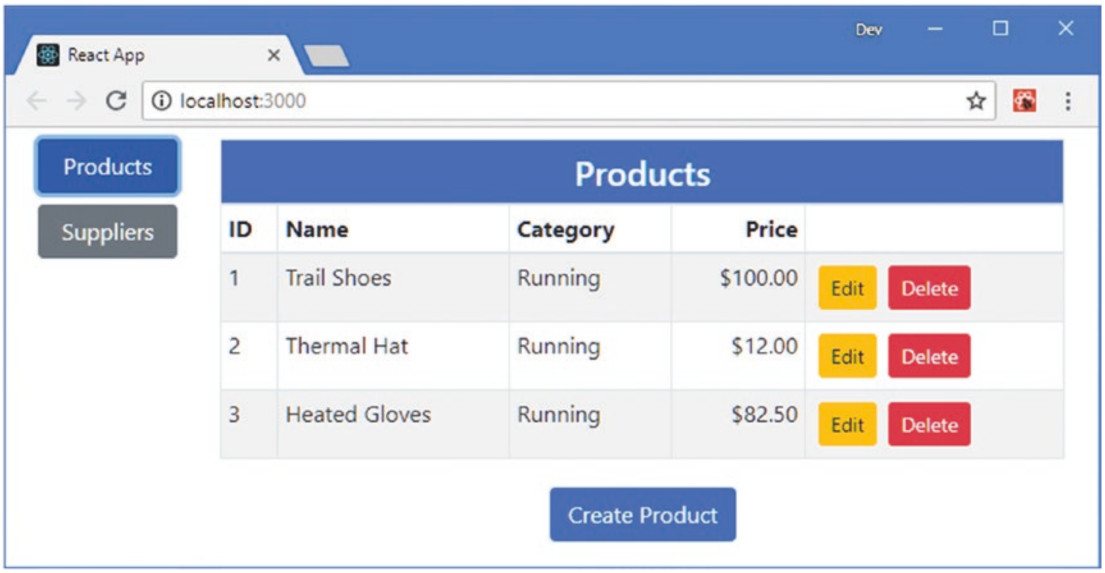
```
export default class App extends Component {

    render() {
        return  <Provider store={ dataStore }>
                    <Selector>
                        <ProductDisplay name="Products" />
                        <SupplierDisplay name="Suppliers" />
                    </Selector>
                </Provider>
    }
}
```

Save the changes to the component JavaScript file and use the command prompt to run the command shown in Listing 21-3 in the productapp folder to start the React development tools.

Listing 21-3. Starting the Development Tools

```
npm start
```

The project will be compiled, and the development HTTP server will be started. A new browser window will open and display the application, as shown in Figure 21-1.



Products

ID	Name	Category	Price	
1	Trail Shoes	Running	$100.00	Edit Delete
2	Thermal Hat	Running	$12.00	Edit Delete
3	Heated Gloves	Running	$82.50	Edit Delete

Figure 21-1. *Running the example application*

Getting Started with URL Routing

To get started, I am going to use URL routing in the `Selector` component so that it doesn't need its own state data to keep track of whether the user wants to work with products or suppliers.

There are two steps to setting up URL routing. The first step is to create the links that the user will click to navigate to a different part of the application. The second step is to select the content that will be displayed for each URL that the user can navigate to. These steps are performed using React components provided by the React-Router package, as shown in Listing 21-4.

Listing 21-4. Adding URL Routing in the Selector.js File in the src Folder

```
import React, { Component } from "react";
import { BrowserRouter as Router, Link, Route } from "react-router-dom";
import { ProductDisplay } from "./ProductDisplay";
import { SupplierDisplay } from "./SupplierDisplay";

export class Selector extends Component {

    // constructor(props) {
    //     super(props);
    //     this.state = {
    //         selection: React.Children.toArray(props.children)[0].props.name
    //     }
    // }

    // setSelection = (ev) => {
    //     ev.persist();
    //     this.setState({ selection: ev.target.name});
    // }

    render() {
        return <Router>
            <div className="container-fluid">
                <div className="row">
                    <div className="col-2">
                        <div><Link to="/products">Products</Link></div>
                        <div><Link to="/suppliers">Suppliers</Link></div>
                    </div>
                    <div className="col">
                        <Route path="/products" component={ ProductDisplay } />
                        <Route path="/suppliers" component={ SupplierDisplay} />
                    </div>
                </div>
            </div>
        </Router>
    }
}
```

Three components are required to set up a basic routing configuration. The `Router` component is used to provide access to the URL routing features. There are different ways of using the URL for navigation, each of which has its own React-Router component that I describe in the "Selecting and Configuring the Router" section. The convention is to import the component you require, which is BrowserRouter in this case, and assign it the name Router, which is then used as a container for the content that requires access to the routing features.

CHOOSING AN ALTERNATIVE ROUTING PACKAGE

React-Router is by far the most widely used routing package for React projects and is a good place to start for most applications. There are other routing packages available, but not all of them are specific to React and can require awkward adaptations.

If you can’t get along with React-Router, then the best alternative is Backbone (https://backbonejs.org). This well-regarded package provides routing for any JavaScript application and works well with React.

Getting Started with the Link Component

The Link component renders an element that the user can click to navigate to a new URL, like this:

```
...
<div><Link to="/products">Products</Link></div>
...
```

The navigation URL is specified using the to prop, and this Link will navigate to the /products URL. Navigation URLs are specified relative to the application’s starting URL, which is http://localhost:3000 during development. That means that specifying /products for the to prop of a Link component tells it to render an element that will navigate to http://localhost:3000/products. These relative URLs will continue to work when the application is deployed and has a public URL.

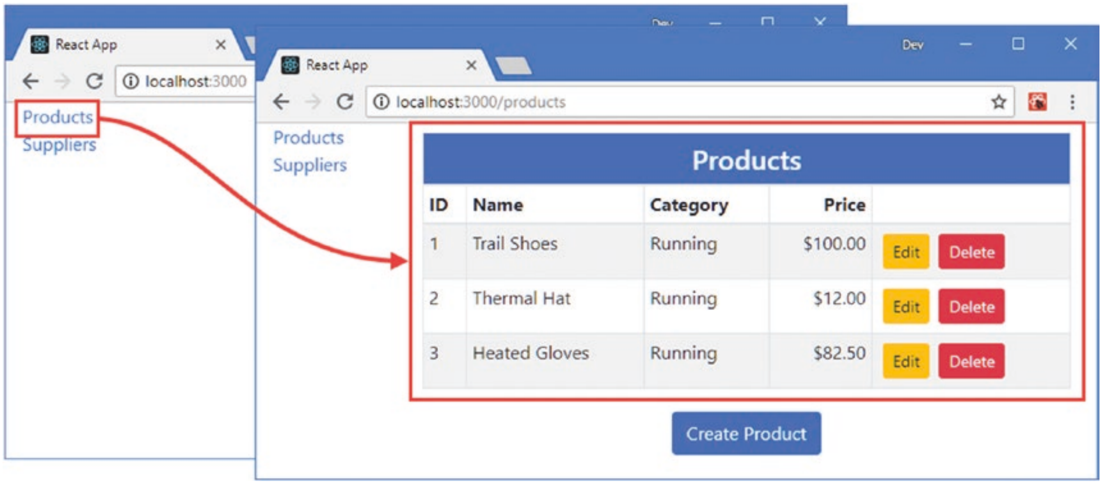
Getting Started with the Route Component

The final component added to Listing 21-4 is Route, which waits until the browser navigates to a specific URL and then displays its content, like this:

```
...
<Route path="/products" component={ ProductDisplay } />
...
```

This Route component has been configured to wait until the browser navigates to the /products URL, at which point it will show the ProductDisplay component. For all other URLs, this Route component will not render any content.

The result of the changes in Listing 21-4 is not visually impressive, as Figure 21-2 shows, but it demonstrates the basic nature of URL routing. When the browser displays the application’s starting URL, http://localhost:3000, no content is shown. When you click the Products or Suppliers links, the browser navigates to http://localhost:3000/products or http://localhost:3000/suppliers, and the ProductDisplay or SupplierDisplay components are shown.



Products

ID	Name	Category	Price	
1	Trail Shoes	Running	$100.00	Edit Delete
2	Thermal Hat	Running	$12.00	Edit Delete
3	Heated Gloves	Running	$82.50	Edit Delete

Figure 21-2. *Adding navigation elements*

Right-click either of navigation elements created by the `Link` components and select Inspect or Inspect Element from the pop-up menu, and you will see the HTML that has been rendered, which looks this:

```
...
<div><a href="/products">Products</a></div>
<div><a href="/suppliers">Suppliers</a></div>
...
```

The `Link` components have been rendered to produce anchor (elements whose tag is `a`) elements, and the value of the `to` prop has translated into URLs for the anchor element's `href` attributes. When you click one of the anchor elements, the browser navigates to a new URL, and the corresponding `Route` component displays its content. If the browser navigates to a URL for which no `Route` component has been configured, then no content is displayed, which is why a component was not shown until one of the links had been clicked.

■ **Caution** Do not try to create your own anchor elements for navigation because they will cause the browser to send an HTTP request to the server for the URL you specify with the effect that the application will be reloaded. The anchor elements rendered by the `Link` component have event handlers that change the URL using the HTML5 History API without triggering a new HTTP request.

Responding to Navigation

The `Route` component is used to implement an application's routing scheme, which it does by waiting until the browser navigates to a specific URL and displaying a component when it does. The mapping between URLs and components can be complex in real applications, and the matching of URLs and the selection of content by the `Route` component can be configured using the props described in Table 21-3, which I demonstrate in the sections that follow.

Table 21-3. The Route Component Props

Name	Description
`path`	The prop is used to specify the URL or URLs that the component should wait for.
`exact`	When this prop is `true`, only URLs that precisely equal the path prop are matched, as demonstrated in the "Restricting Matches with Props" section.
`sensitive`	When this prop is `true`, matching URLs is case-sensitive.
`strict`	When this prop is `true`, `path` values that end in a / will only match URLs whose corresponding segment also ends with a /.
`component`	This prop is used to specify a single component that will be displayed when the `path` prop matches the browser's current URL.
`render`	This prop is used to specify a function that returns the content that will be displayed when the `path` prop matches the browser's current URL.
`children`	This prop is used to specify a function that will always render content, even when the URL specified by the path prop doesn't match. This is useful for displaying content in descendent components or components that are not rendered in response to URL changes, as described in Chapter 22.

Selecting Components and Content

The `component` prop is used to specify a single component that will be displayed if the current URL is matched by the `path` prop. The component type is specified directly as the `component` prop value, like this:

```
...
<Route path="/products" component={ ProductDisplay } />
...
```

The value of the `component` prop should not be a function because it can lead to a new instance of the specified component being created each time that the application updates.

Using the render Prop

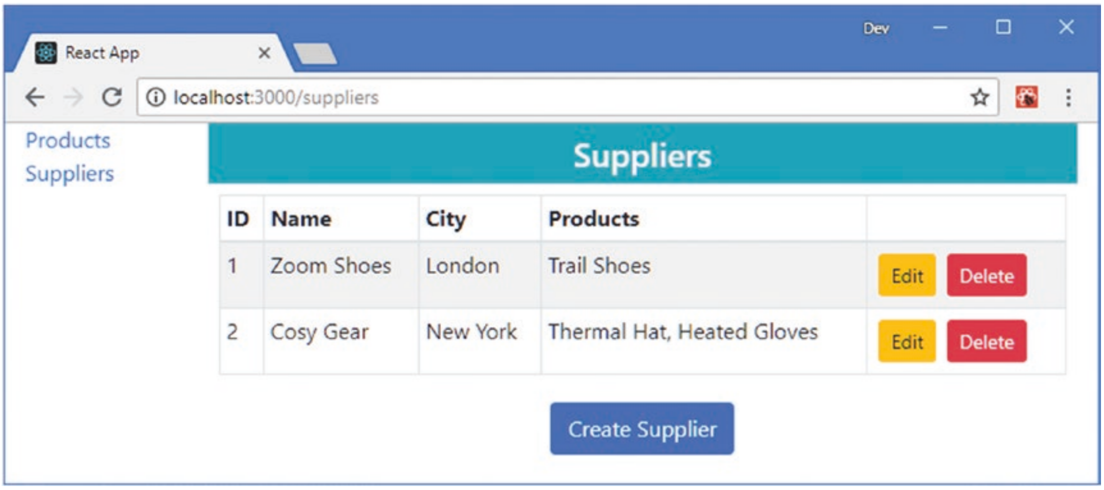
The advantage of the `component` prop is simplicity, and it works well for projects that have self-contained components that render all the required content and don't require props. The `Route` component provides the `render` prop for more complex content and to pass on props, as shown in Listing 21-5.

Listing 21-5. Using the render Prop in the Selector.js File in the src Folder

```
import React, { Component } from "react";
import { BrowserRouter as Router, Link, Route } from "react-router-dom";
import { ProductDisplay } from "./ProductDisplay";
import { SupplierDisplay } from "./SupplierDisplay";

export class Selector extends Component {

    render() {
        return <Router>
```

```
            <div className="container-fluid">
                <div className="row">
                    <div className="col-2">
                        <div><Link to="/products">Products</Link></div>
                        <div><Link to="/suppliers">Suppliers</Link></div>
                    </div>
                    <div className="col">
                        <Route path="/products" render={ (routeProps) =>
                            <ProductDisplay myProp="myValue" /> } />
                        <Route path="/suppliers" render={ (routeProps) =>
                             <React.Fragment>
                                <h4 className="bg-info text-center text-white p-2">
                                    Suppliers
                                </h4>
                                <SupplierDisplay />
                            </React.Fragment>
                        } />
                    </div>
                </div>
            </div>
        </Router>
    }
}
```

The result of the function is the content that should be displayed by the Route component. In the listing, I passed on a prop to the ProductDisplay component and included the SupplierDisplay component in a larger fragment of content, as shown in Figure 21-3.



ID	Name	City	Products	
1	Zoom Shoes	London	Trail Shoes	Edit Delete
2	Cosy Gear	New York	Thermal Hat, Heated Gloves	Edit Delete

Figure 21-3. *Using the Route component's render prop*

■ **Tip** The function passed to the `render` prop receives an object that provides information about the state of the routing system, which I describe in Chapter 22.

Matching URLs

One of the most difficult aspects of using URL routing is making sure that the URLs you want to support are correctly matched by Route components. The Route component provides a range of features that allow you to expand or narrow the range of URLs that will be matched, which I describe in the sections that follow.

Matching Using Segments

The simplest way to match a URL is to provide one or more target segments to the Route component's path prop. This will match any URL that starts with the segments you specify, as shown in Listing 21-6.

Listing 21-6. Matching URLs in the Selector.js File in the src Folder

```
import React, { Component } from "react";
import { BrowserRouter as Router, Link, Route } from "react-router-dom";
//import { ProductDisplay } from "./ProductDisplay";
//import { SupplierDisplay } from "./SupplierDisplay";

export class Selector extends Component {

    renderMessage = (msg) => <h5 className="bg-info text-white m-2 p-2">{ msg }</h5>

    render() {
        return <Router>
            <div className="container-fluid">
                <div className="row">
                    <div className="col-2">
                        <div><Link to="/data/one">Link #1</Link></div>
                        <div><Link to="/data/two">Link #2</Link></div>
                        <div><Link to="/people/bob">Bob</Link></div>
                    </div>
                    <div className="col">
                        <Route path="/data"
                            render={ () => this.renderMessage("Route #1") } />
                        <Route path="/data/two"
                            render={ () => this.renderMessage("Route #2") } />
                    </div>
                </div>
            </div>
        </Router>
    }
}
```

I replaced the ProductDisplay and SupplierDisplay components with content generated by a method called renderMessage. There are three Link components, which target the URLs /data/one, data/two, and /people/bob.

The first Route component is configured with /data as its path prop. This will match any URL whose first segment is data, which means it will match the /data/one and /data/two URLs but not /people/bob. The second Route component has /data/two as the value of its path prop, so it will only match the /data/two URL. Each Route component evaluates its path prop independently, and you can see how they match URLs by clicking the navigation links, as shown in Figure 21-4.

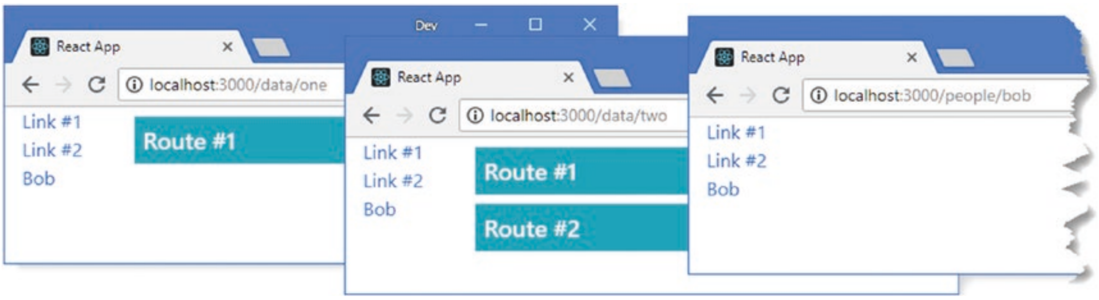


***Figure 21-4.** Matching URLs with the Route component*

One Route component matches the /data/one URL, both match the /data/two URL, and neither matches /people/bob, and so no content is displayed.

Restricting Matches with Props

The default behavior of the Route component can lead to over-matching, where a component matches a URL when you don't want it to. I might want to distinguish between the /data and /data/one URLs, for example, so that the first URL displays a list of data items and the second displays the details of a specific object. The default matching makes this difficult because a path prop of /data matches any URL whose first segment is /data, regardless of how many segments the URL contains in total.

To help restrict the range of URLs that a path will match, the Route component supports three additional props: exact, strict, and sensitive. The most useful of the three props is exact, which will match a URL only if it exactly matches the path prop value so that a URL of /data/one won't be matched by a path of /data, as shown in Listing 21-7.

Listing 21-7. Making Exact Matches in the Selector.js File in the src Folder

```
import React, { Component } from "react";
import { BrowserRouter as Router, Link, Route } from "react-router-dom";

export class Selector extends Component {

    renderMessage = (msg) => <h5 className="bg-info text-white m-2 p-2">{ msg }</h5>

    render() {
        return <Router>
            <div className="container-fluid">
                <div className="row">
                    <div className="col-2">
                        <div><Link to="/data">Data</Link></div>
                        <div><Link to="/data/one">Link #1</Link></div>
                        <div><Link to="/data/two">Link #2</Link></div>
                        <div><Link to="/people/bob">Bob</Link></div>
                    </div>
                    <div className="col">
                        <Route path="/data" exact={ true }
                            render={ () => this.renderMessage("Route #1") } />
                        <Route path="/data/two"
                            render={ () => this.renderMessage("Route #2") } />
                    </div>
                </div>
            </div>
        </Router>
    }
}
```

Setting the exact prop affects only the Route component to which it is applied. In the example, the exact prop prevents the first Route component from matching the /data/one and /data/two URLs, as shown in Figure 21-5.

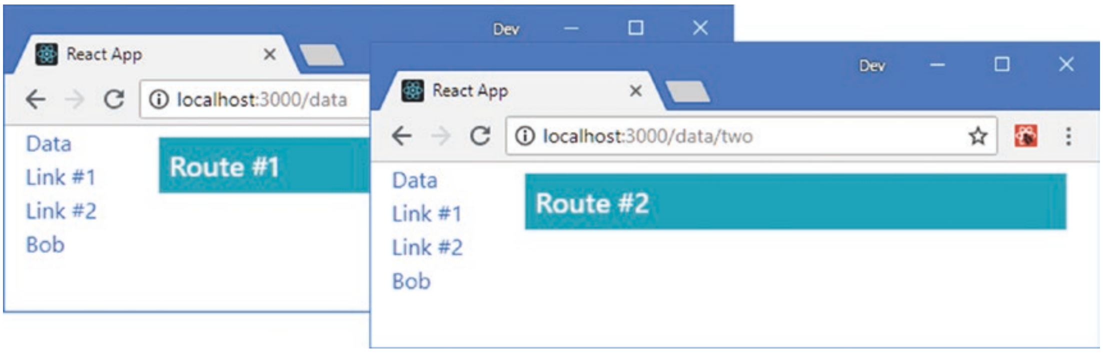


Figure 21-5. *Making exact matches*

When set to true, the strict prop is used to restrict matches for a path that has a trailing slash to URLs that have one too, so a path of /data/ will only match the /data/ URL and not /data. The strict prop does match URLs with additional segments, however, so that a path of /data/ will match /data/one.

The sensitive prop is used to control case sensitivity. When true, it will allow matches only when the case of the path prop matches the case of the URL, so a path of /data will not match a /Data URL.

Specifying Multiple URLs in a Path

The value of the Route component's path prop can be an array of URLs, which causes content to be displayed if any of them are matched. This can be useful when the same content is required in response to URLs that don't have a common structure (such as displaying the same component in response to /data/list and /people/list) or when a specific number of exact matches are required, such as matching /data/one and /data/two but not any other URL that starts with /data, as demonstrated in Listing 21-8.

■ **Note** At the time of writing, there is a mismatch with the prop types that are expected by the Route component that results in a JavaScript console warning when an array is used. This warning can be ignored and may be fixed by the time you read this chapter. See Chapters 10 and 11 for details of how the data types that a component expects for its props can be specified.

Listing 21-8. Using an Array of Paths in the Selector.js File in the src Folder

```
import React, { Component } from "react";
import { BrowserRouter as Router, Link, Route } from "react-router-dom";

export class Selector extends Component {

    renderMessage = (msg) => <h5 className="bg-info text-white m-2 p-2">{ msg }</h5>

    render() {
        return <Router>
            <div className="container-fluid">
                <div className="row">
                    <div className="col-2">
                        <div><Link to="/data">Data</Link></div>
                        <div><Link to="/data/one">Link #1</Link></div>
                        <div><Link to="/data/two">Link #2</Link></div>
                        <div><Link to="/people/bob">Bob</Link></div>
                    </div>
                    <div className="col">
                        <Route path={["/data/one", "/people/bob" ] }  exact={ true }
                            render={ () => this.renderMessage("Route #1") } />
                        <Route path={["/data", "/people" ] }
                            render={ () => this.renderMessage("Route #2") } />
                    </div>
                </div>
            </div>
        </Router>
    }
}
```

The path array is provided as an expression using curly braces. The path property of the first Route component is set to an array containing /data/one and /people/bob. These paths are combined with the exact prop to restrict the URLs that the component will match. The second Route component is configured to match more widely and will respond to any URL whose first segment is data or people, as shown in Figure 21-6.

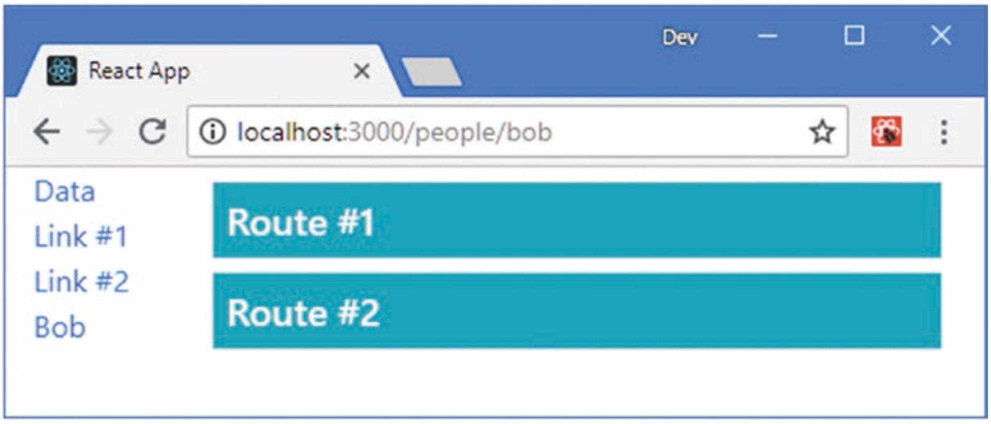


Figure 21-6. *Using an array to specify paths*

Matching URLs with Regular Expressions

Not all combinations of URLs can be expressed using individual segments, and the Route component supports regular expressions in its path prop for more complex matches, as shown in Listing 21-9.

REGULAR EXPRESSION CLARITY VERSUS CONCISENESS

Most programmers have a tendency to express routes with the fewest regular expressions possible, but the result can be a routing configuration that is hard to read and breaks easily when changes are required. When deciding how to match URLs, keep expressions simple and use a path array to expand the range of URLs that a Route can match without using regular expressions that are difficult to understand.

Listing 21-9. Using a Regular Expression in the Selector.js File in the src Folder

```
import React, { Component } from "react";
import { BrowserRouter as Router, Link, Route } from "react-router-dom";

export class Selector extends Component {

    renderMessage = (msg) => <h5 className="bg-info text-white m-2 p-2">{ msg }</h5>

    render() {
        return <Router>
            <div className="container-fluid">
                <div className="row">
                    <div className="col-2">
                        <div><Link to="/data">Data</Link></div>
                        <div><Link to="/data/one">Link #1</Link></div>
                        <div><Link to="/data/two">Link #2</Link></div>
                        <div><Link to="/data/three">Link #3</Link></div>
                        <div><Link to="/people/bob">Bob</Link></div>
                        <div><Link to="/people/alice">Alice</Link></div>
                    </div>
                    <div className="col">
                        <Route path={["/data/(one|three)", "/people/b*" ] }
                            render={ () => this.renderMessage("Route #1") } />
                    </div>
                </div>
            </div>
        </Router>
    }
}
```

The first item in the path array matches URLs whose first segment is `data` and second segment is `one` or `three`. The second item matches URLs whose first segment is people and whose second segment starts with `b`. The result is that the `Route` component will match the `/data/one`, `/data/two`, and `/people/bob` URLs but not the `/data/two` and `/people/alice` URLs.

■ **Note** See https://github.com/pillarjs/path-to-regexp for the full range of regular expression features that can be used to match URLs.

Making a Single Route Match

Each `Route` component assesses its `path` prop independently; this can be useful but isn't ideal if you want just one component to be display based n the current URL. For these situations, the Redux-Router package provides the `Switch` component, which acts as a wrapper around multiple `Route` components, queries them in order, and displays the content rendered by the first one to match the current URL. Listing 21-10 shows the use of the `Switch` component.

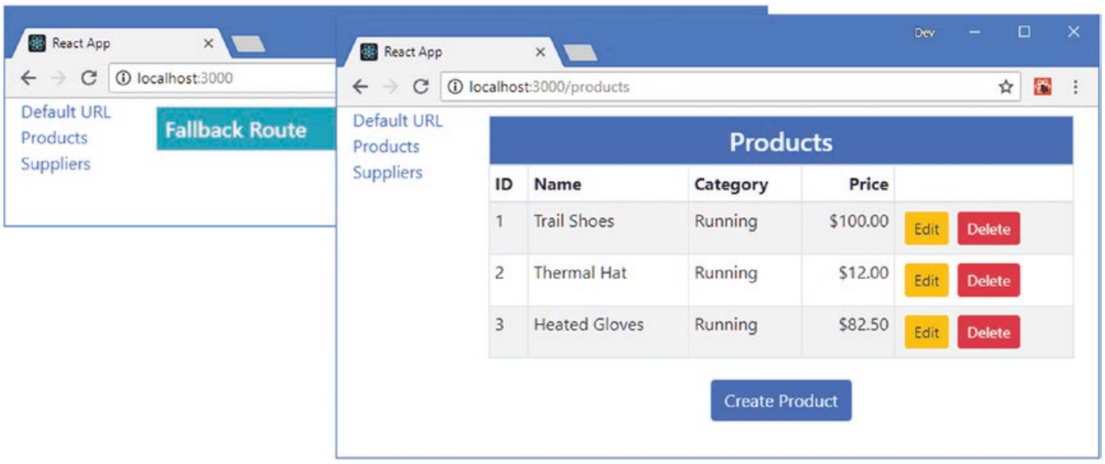
Listing 21-10. Using the Switch Component in the Selector.js File in the src Folder

```
import React, { Component } from "react";
import { BrowserRouter as Router, Link, Route, Switch } from "react-router-dom";
import { ProductDisplay } from "./ProductDisplay";
import { SupplierDisplay } from "./SupplierDisplay";

export class Selector extends Component {

    renderMessage = (msg) => <h5 className="bg-info text-white m-2 p-2">{ msg }</h5>

    render() {
        return <Router>
            <div className="container-fluid">
                <div className="row">
                    <div className="col-2">
                        <div><Link to="/">Default URL</Link></div>
                        <div><Link to="/products">Products</Link></div>
                        <div><Link to="/suppliers">Suppliers</Link></div>
                    </div>
```

```
          <div className="col">
            <Switch>
              <Route path="/products" component={ ProductDisplay} />
              <Route path="/suppliers" component={ SupplierDisplay } />
              <Route render={ () =>
                this.renderMessage("Fallback Route")} />
            </Switch>
          </div>
        </div>
      </div>
    </Router>
  }
}
```

The Switch component checks its children in the order they are defined, which means that the Route components must be arranged so that the most specific URLs appear first. A Route component with no path prop will always match the current URL and can be used as the default by the Switch component, similar to the default clause in a regular JavaScript switch statement.

The changes in the listing associated the /products URL with the ProductDisplay component and the /suppliers URL with the SupplierDisplay component. Any other URL will cause a message to be rendered using the renderMessage method, as shown in Figure 21-7.



ID	Name	Category	Price
1	Trail Shoes	Running	$100.00
2	Thermal Hat	Running	$12.00
3	Heated Gloves	Running	$82.50

Figure 21-7. *Using a Switch component*

The use of the Switch component allows me to render content when the application first starts before the user has clicked one of the navigation links. However, this is only one way to select content for the default URL, and a more elegant approach is to use the Redirect component, as described in the next section.

Using Redirection as the Fallback Route

For some applications, it doesn't make sense to introduce a separate URL as a fallback, in which case the Redirect component can be used to automatically trigger navigation to a URL that can be handled by a Route component. In Listing 21-11, I have replaced the existing fallback with a redirection to the /product URL.

Listing 21-11. Using a Redirection in the Selector.js File in the src Folder

```
import React, { Component } from "react";
import { BrowserRouter as Router, Link, Route, Switch, Redirect }
    from "react-router-dom";
import { ProductDisplay } from "./ProductDisplay";
import { SupplierDisplay } from "./SupplierDisplay";

export class Selector extends Component {

    renderMessage = (msg) => <h5 className="bg-info text-white m-2 p-2">{ msg }</h5>

    render() {
        return <Router>
            <div className="container-fluid">
                <div className="row">
                    <div className="col-2">
                        <div><Link to="/">Default URL</Link></div>
                        <div><Link to="/products">Products</Link></div>
                        <div><Link to="/suppliers">Suppliers</Link></div>
                    </div>
                    <div className="col">
                        <Switch>
                            <Route path="/products" component={ ProductDisplay} />
                            <Route path="/suppliers" component={ SupplierDisplay } />
                            <Redirect to="/products" />
                        </Switch>
                    </div>
                </div>
            </div>
        </Router>
    }
}
```

The to prop specifies the URL that the Redirect component will navigate to. The Redirect component won't be used if the Route components are able to match the current URL. But if the Switch component reaches the Redirect component without having found a matching Route, then redirection to /products will be performed.

Performing Selective Redirection

The most common way to use the Redirect component is with just the to prop, but there are additional props available that can be used to restrict when redirection is performed, as described in Table 21-4.

Table 21-4. *The Redirect Component Props*

Name	Description
`to`	This prop specifies the location to which the browser should be redirected.
`from`	This prop restricts the redirection so that it is performed only when the current URL matches the specified path.
`exact`	When `true`, this prop restricts redirection so that it is performed only when the current URL exactly matches the `from` prop, performing the same role as the `Route` component's `exact` prop.
`strict`	When `true`, this prop restricts redirection so that it is performed only when the current URL ends with a / if the `path` also ends with a `/`, performing the same role as the `Route` component's `strict` prop.
`push`	When `true`, the redirection will add a new item to the browser's history. When `false`, the redirection will replace the current location.

Selectively redirecting to a new URL is a useful way of maintaining support for URLs that are no longer directly handled by a `Route`, as shown in Listing 21-12. (A similar effect can be achieved using a `path` array for a `Route`, but that can lead to complications when matching URL parameters, as described in Chapter 22.)

Listing 21-12. Selectively Redirecting URLs in the Selector.js File in the src Folder

```
import React, { Component } from "react";
import { BrowserRouter as Router, Link, Route, Switch, Redirect }
    from "react-router-dom";
import { ProductDisplay } from "./ProductDisplay";
import { SupplierDisplay } from "./SupplierDisplay";

export class Selector extends Component {

    renderMessage = (msg) => <h5 className="bg-info text-white m-2 p-2">{ msg }</h5>

    render() {
        return <Router>
            <div className="container-fluid">
                <div className="row">
                    <div className="col-2">
                        <div><Link to="/">Default URL</Link></div>
                        <div><Link to="/products">Products</Link></div>
                        <div><Link to="/suppliers">Suppliers</Link></div>
                        <div><Link to="/old/data">Old Link</Link></div>
                    </div>
                    <div className="col">
                        <Switch>
                            <Route path="/products" component={ ProductDisplay} />
                            <Route path="/suppliers" component={ SupplierDisplay } />
                            <Redirect from="/old/data" to="/suppliers" />
                            <Redirect to="/products" />
                        </Switch>
                    </div>
                </div>
            </div>
        </Router>
    }
}
```

The new Redirect will perform a redirection from the /old/data URL to /suppliers. The order of selective Redirect components is important, and they must be placed before the nonselective redirections are performed; otherwise, the Switch will not reach them as it works its way through the list of routing components.

Rendering Navigation Links

The Link component is responsible for generating the elements that navigate to new URLs, which it does by rendering an anchor element with an event handler that changes the browser's URL without reloading the application. To configure its behavior, the Link component accepts the props described in Table 21-5.

Table 21-5. *The Link Component Props*

Name	Description
to	This prop is used to specify the location that clicking the link will navigate to.
replace	This prop is used to specify whether clicking the navigation link will add an entry to the browser's history or replace the current entry, which determines whether the user will be able to use the back button to return to the previous location. The default value is false.
innerRef	This prop is used to access a ref for the underlying HTML element. See Chapter 16 for details of refs.

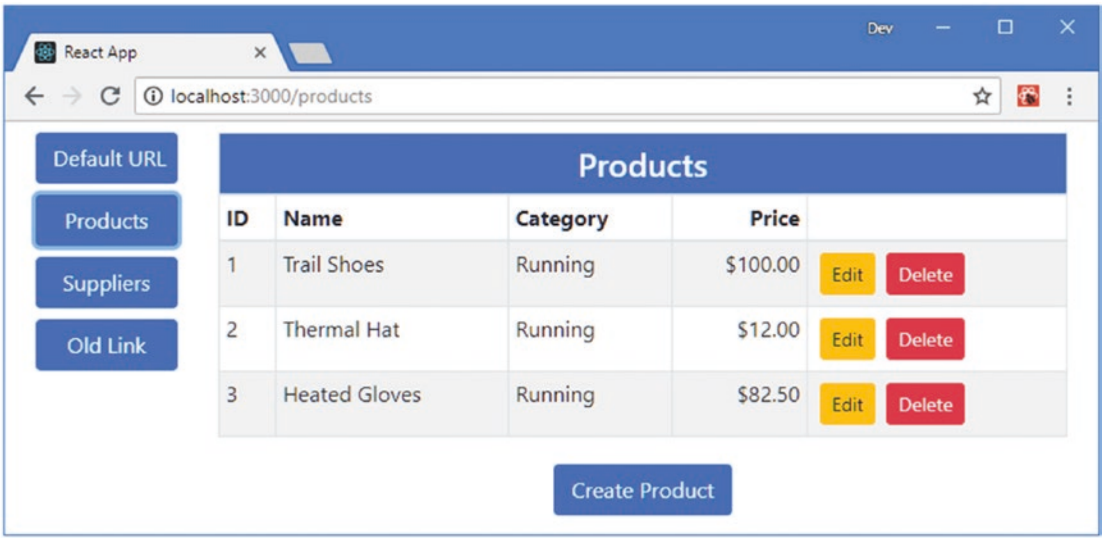
The Link component will pass on any other props to the anchor element that it renders. The main use for this feature is to apply the className prop to the Link to style the navigation links, as shown in Listing 21-13.

Listing 21-13. Applying Classes in the Selector.js File in the src Folder

```
import React, { Component } from "react";
import { BrowserRouter as Router, Link, Route, Switch, Redirect }
    from "react-router-dom";
import { ProductDisplay } from "./ProductDisplay";
import { SupplierDisplay } from "./SupplierDisplay";

export class Selector extends Component {

    renderMessage = (msg) => <h5 className="bg-info text-white m-2 p-2">{ msg }</h5>

    render() {
        return <Router>
            <div className="container-fluid">
                <div className="row">
                    <div className="col-2">
```

```
                        <Link className="m-2 btn btn-block btn-primary"
                            to="/">Default URL</Link>
                        <Link className="m-2 btn btn-block btn-primary"
                            to="/products">Products</Link>
                        <Link className="m-2 btn btn-block btn-primary"
                            to="/suppliers">Suppliers</Link>
                        <Link className="m-2 btn btn-block btn-primary"
                            to="/old/data">Old Link</Link>
                    </div>
                    <div className="col">
                        <Switch>
                            <Route path="/products" component={ ProductDisplay} />
                            <Route path="/suppliers" component={ SupplierDisplay } />
                            <Redirect from="/old/data" to="/suppliers" />
                            <Redirect to="/products" />
                        </Switch>
                    </div>
                </div>
            </div>
        </Router>
    }
}
```

The Bootstrap CSS framework is able to style anchor elements as buttons, and the classes that I have applied in Listing 21-13 apply a button style that fills the available horizontal space and allows me to remove the div elements that I used to stack the navigation links vertically. When the Link components render their content, the result is a navigation link that appears as a button, as shown in Figure 21-8.



Products

ID	Name	Category	Price	
1	Trail Shoes	Running	$100.00	Edit Delete
2	Thermal Hat	Running	$12.00	Edit Delete
3	Heated Gloves	Running	$82.50	Edit Delete

Figure 21-8. *Passing on classes to the navigation element*

Indicating the Active Route

The NavLink component builds on the basic Link features but will add a class or style to the anchor element when the value of its to property matches the current URL. Table 21-6 describes the properties provided by the NavLink component, which are defined in addition to those described in Table 21-5. In Listing 21-14, I have introduced NavLink components that apply the active class.

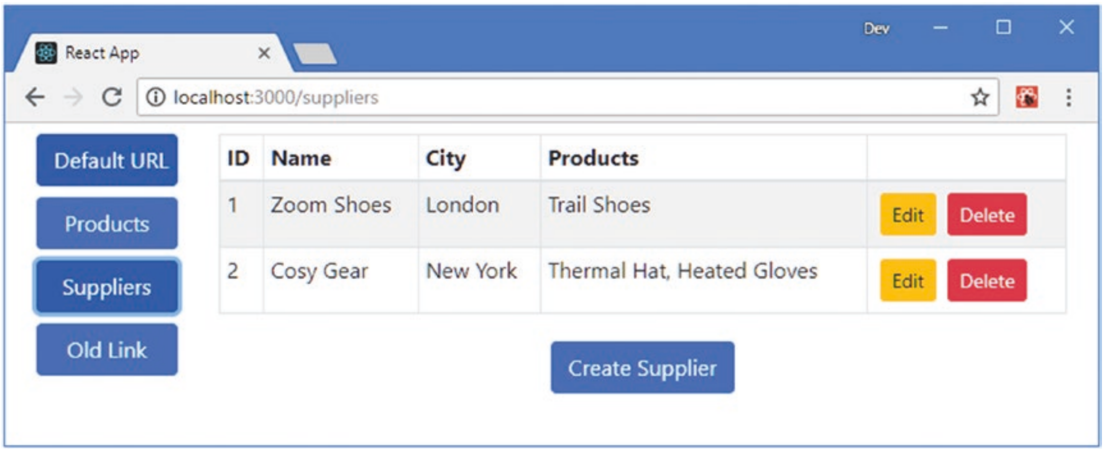
Table 21-6. *The NavLink Component Properties*

Name	Description
activeClassName	This prop specifies the classes that will be added to the anchor element when the link is active.
activeStyle	This prop specifies the styles that will be added to the anchor element when the link is active. Styles are specified as a JavaScript object whose properties are the style names.
exact	When true, this prop enforces exact matching, as described in the "Matching URLs" section.
strict	When true, this prop enforces strict matching, as described in the "Matching URLs" section.
isActive	This prop can be used to specify a custom function that determines whether the link is active. The function receives match and location arguments, as described in Chapter 22. The default behavior compares the current URL with the to prop.

Listing 21-14. Using NavLink Components in the Selector.js File in the src Folder

```
import React, { Component } from "react";
import { BrowserRouter as Router, NavLink, Route, Switch, Redirect }
    from "react-router-dom";
import { ProductDisplay } from "./ProductDisplay";
import { SupplierDisplay } from "./SupplierDisplay";

export class Selector extends Component {

    renderMessage = (msg) => <h5 className="bg-info text-white m-2 p-2">{ msg }</h5>

    render() {
        return <Router>
            <div className="container-fluid">
                <div className="row">
                    <div className="col-2">
                        <NavLink className="m-2 btn btn-block btn-primary"
                            activeClassName="active"
                            to="/">Default URL</NavLink>
                        <NavLink className="m-2 btn btn-block btn-primary"
                            activeClassName="active"
                            to="/products">Products</NavLink>
                        <NavLink className="m-2 btn btn-block btn-primary"
                            activeClassName="active"
                            to="/suppliers">Suppliers</NavLink>
                        <NavLink className="m-2 btn btn-block btn-primary"
                            activeClassName="active"
                            to="/old/data">Old Link</NavLink>
                    </div>
                    <div className="col">
                        <Switch>
                            <Route path="/products" component={ ProductDisplay} />
                            <Route path="/suppliers" component={ SupplierDisplay } />
                            <Redirect from="/old/data" to="/suppliers" />
                            <Redirect to="/products" />
                        </Switch>
                    </div>
                </div>
            </div>
        </Router>
    }
}
```

When the browser's URL matches the value of a component's `to` prop, the anchor element is added to the `active` class, which provides a useful indicator to the user, as shown in Figure 21-9.



ID	Name	City	Products	
1	Zoom Shoes	London	Trail Shoes	Edit Delete
2	Cosy Gear	New York	Thermal Hat, Heated Gloves	Edit Delete

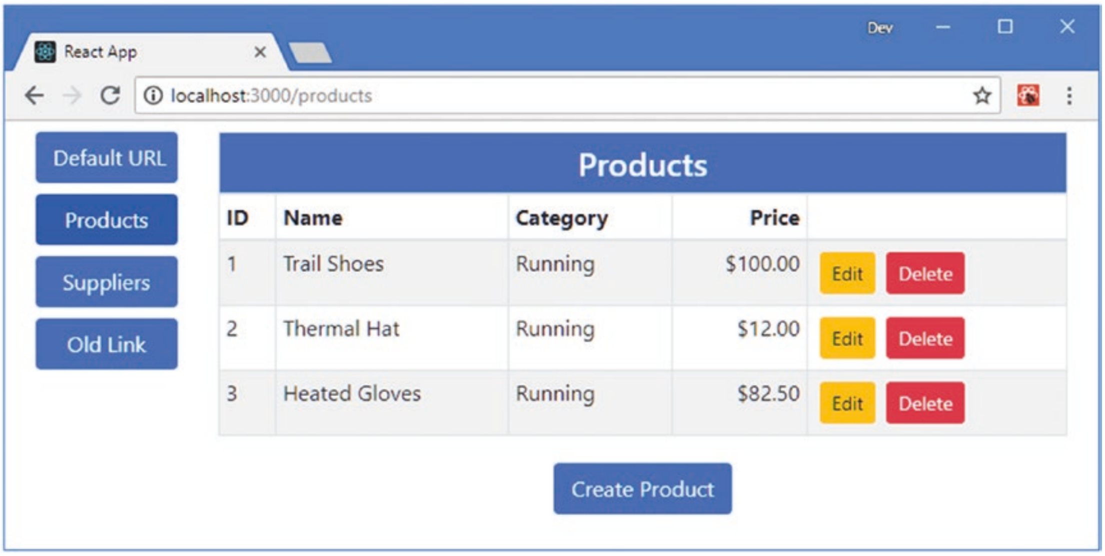
Figure 21-9. *Responding to route activation*

Notice that the Default URL button is always highlighted. The `NavLink` component relies on the `Route` URL matching, which means that a `to` prop of `/` will match any URL. The `exact` and `strict` props described in Table 21-6 have the same purpose as when applied to a `Route`, and Listing 21-15 shows the use of the `exact` prop to restrict matching.

Listing 21-15. Restricting NavLink Matching in the Selector.js File in the src Folder

```
...
<div className="col-2">
    <NavLink className="m-2 btn btn-block btn-primary"
        activeClassName="active" exact={ true }
        to="/">Default URL</NavLink>
    <NavLink className="m-2 btn btn-block btn-primary"
        activeClassName="active"
        to="/products">Products</NavLink>
    <NavLink className="m-2 btn btn-block btn-primary"
        activeClassName="active"
        to="/suppliers">Suppliers</NavLink>
    <NavLink className="m-2 btn btn-block btn-primary"
        activeClassName="active"
        to="/old/data">Old Link</NavLink>
</div>
...
```

The result is that the NavLink is no longer highlighted, as shown in Figure 21-10.



ID	Name	Category	Price	
1	Trail Shoes	Running	$100.00	Edit Delete
2	Thermal Hat	Running	$12.00	Edit Delete
3	Heated Gloves	Running	$82.50	Edit Delete

Figure 21-10. *Restricting URL matching for highlighting*

■ **Note** The NavLink component doesn't allow classes to be removed when the activeClassName value is applied, which means that I can't accurately re-create the original effect from the example project. I demonstrate how to create this functionality with a custom navigation component in Chapter 22.

Selecting and Configuring the Router

URL routing relies on manipulating the browser's URL to perform navigation without sending HTTP requests to the server. In a web application, the core routing functionality is provided by either the BrowserRouter or HashRouter component; both are conventionally given the name Router when they are imported, like this:

```
...
import { BrowserRouter as Router, Link, Switch, Route, Redirect }
    from "react-router-dom";
...
```

BrowserRouter uses the HTML5 History API. This API provides natural URLs for routing, such as http://localhost:3000/products, which is the type of URL you have seen in the examples in this chapter. The BrowserRouter component can accept a range of props that configure its behavior, as described in Table 21-7. The default values for the props are suitable for most applications.

Table 21-7. *The BrowserRouter Props*

Name	Description
basename	This prop is used when the application isn't at the root of its URL, such as http://localhost:3000/myapp.
getUserConfirmation	This prop is used to specify the function used to obtain user confirmation for navigation with the Prompt component, described in Chapter 22.
forceRefresh	When true, this prop forces a complete refresh during navigation with an HTTP request sent to the server. This undermines the point of a rich client-side application and should be used only for testing and when browsers are unable to use the History API.
keyLength	Each change in navigation is given a unique key. This prop is used to specify the length of the key and defaults to six characters. The key is incorporated into the location objects that identify each navigation location, described in Chapter 22.
history	This prop allows a custom history object to be used. The history object is described in Chapter 22.

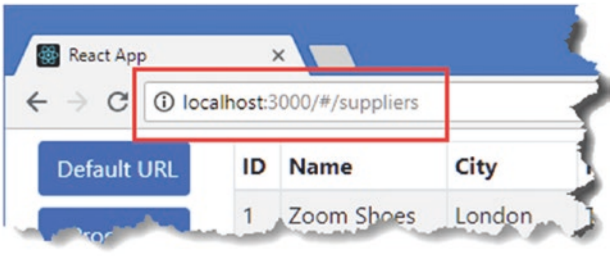
Using the HashRouter Component

Older browsers don't support the History API, and the navigation details have to be added as a fragment at the end of the URL, following the # character. Routing with URL fragments is provided by the HashRouter component, as shown in Listing 21-16.

Listing 21-16. Using the HashRouter Component in the Selector.js File in the src Folder

```
import React, { Component } from "react";
import { HashRouter as Router, NavLink, Route, Switch, Redirect }
    from "react-router-dom";
import { ProductDisplay } from "./ProductDisplay";
import { SupplierDisplay } from "./SupplierDisplay";

export class Selector extends Component {

    // ...methods omitted for brevity...
}
```

Using the as keyword to import the routing component means that only the import statement has to change. Save the changes to the file and navigate to http://localhost:3000, and you will see that the style of the URL has changed, as shown in Figure 21-11.



ID	Name	City
1	Zoom Shoes	London

Figure 21-11. *Using hash routing*

■ **Tip** You may see a URL like `http://localhost:3000/suppliers/#/suppliers` when the browser first reloads. This happens because the browser reloads from its current URL, which is then assumed to be the base URL for the application. Manually navigate to `http://localhost:3000` and you should see the URL shown in the figure.

The part of the URL that is used for routing now follows the # character. URL routing still works the same way, but the URLs are less natural compared with those generated by the `BrowserRouter` component. The `HashRouter` component can be configured with the props shown in Table 21-8.

Table 21-8. *The HashRouter Component Props*

Name	Description
`basename`	This prop is used when the application isn't at the root of its URL, such as `http://localhost:3000/myapp`.
`getUserConfirmation`	This prop is used to specify the function used to obtain user confirmation for navigation with the `Prompt` component, described in Chapter 22.
`hashType`	This prop sets the style used to encode the routing in the URL. The options are `slash`, which creates the URL style shown in Figure 21-11; `noslash`, which omits the leading / after the # character; and `hashbang`, which creates URLs such as `#!/products` by inserting an exclamation mark after the # character.

Summary

In this chapter, I showed you how to use the React-Router package to add URL routing to a React application. I explained how routing can simplify applications by moving state data into the URL and how `Link` and `Route` components are used to create navigation elements and respond to URL changes. In the next chapter, I describe the advanced URL routing features.