currying, in which you decompose a series of function parameters into a series of chained function calls, as shown in the following:

```
const makeContainer = element => content =>
                        `<${element}>${content}</${element}>`;
```

This will now work with our pipe(), for instance:

```
const makeNested = pipe(upper,
                        makeContainer("span"),
                        makeContainer("div"),
                        makeContainer("article"),
                        makeContainer("main"));
tag = makeNested(content);
console.log(tag);
```

What will be the output? It will be:

```
<main><article><div><span>DID YOU TRAVEL?</span></div></article></main>
```

This style of programming is an example of what is sometimes called functional programming. With functional programming, most of our functions should be **pure functions**. What is that? A pure function cannot change the content of its parameters nor the content of anything outside of itself (i.e., they produce no side effects). As well, a pure function, given the same input parameters, will always produce the same return value. As a result, pure functions are more testable and reliable.

While not every function can be pure in JavaScript, as a general rule of thumb, try to favor creating pure functions over impure ones.

TEST YOUR KNOWLEDGE #2

In this exercise, you will create an application with create-react-app that requires both routing and state handling. The HTML that your page must eventually render has been provided in the files **lab11b-test02-markup-only.html**, **home-markup-only.html**, **about-markup-only.html**, and **lab11b-test02-styles.css**, and can be seen in Figure 11.19. The provided readme file contains URLs for the API and the image file locations. If you are following the labs for this chapter, you will likely have already completed this exercise already.

1. Use create-react-app to create the starting files for this application.
2. Create functional components for the components shown in Figure 11.19. Remember that with create-react-app, each component will exist in its own file. To make your source code easier to manage, create a folder named **components** within the **src** folder.
3. The data for this exercise can be fetched from the URL in the readme file. You will have to perform the fetch within either the `componentDidMount()` handler (if

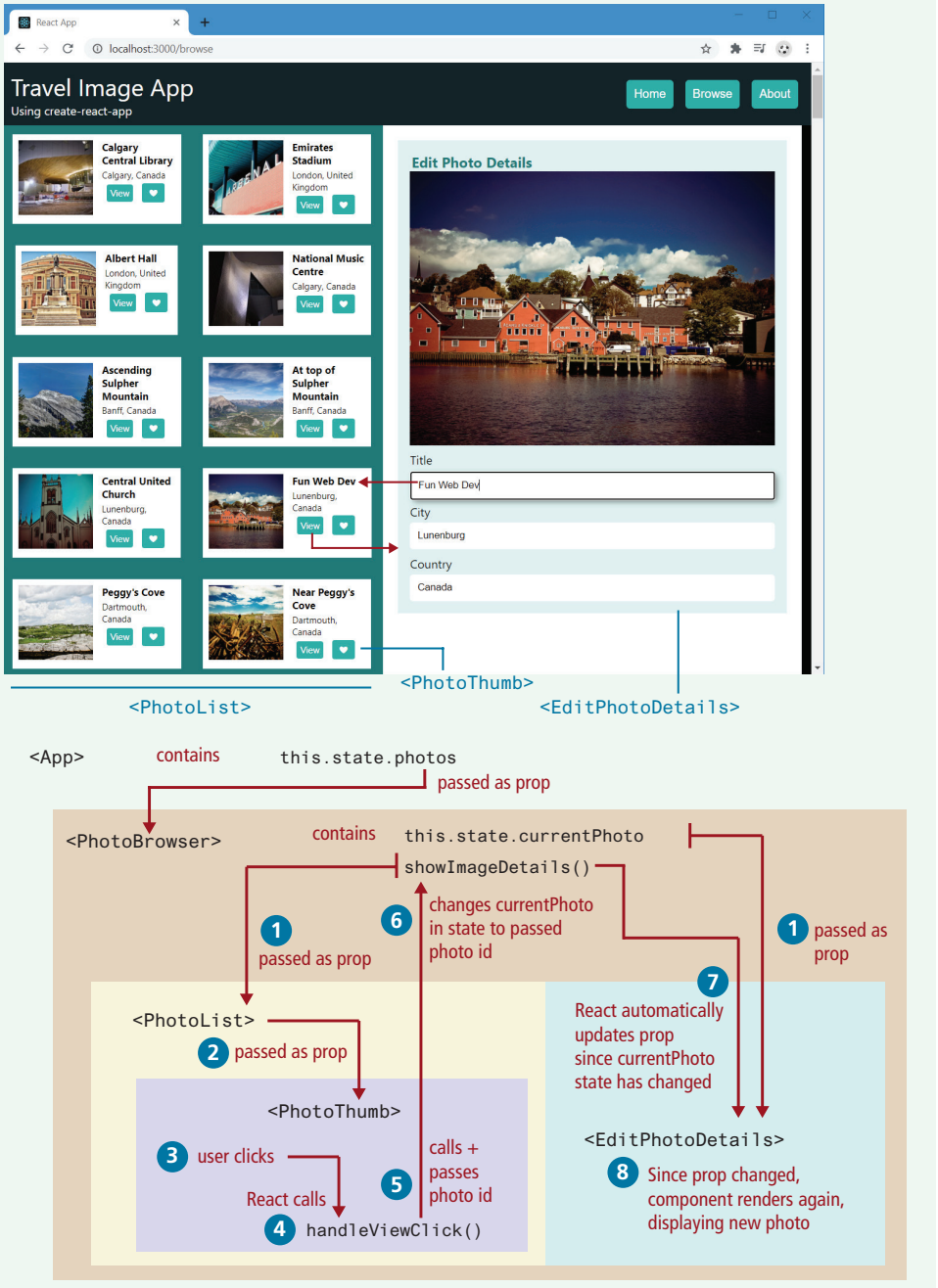

FIGURE 11.19 Completed Test Your Knowledge #2

not using Hooks) or the `useEffect()` handler (if using Hooks) of the App component, as shown in Section 11.5.1. The supplied **single-image.json** file can be used to examine the content of a single image returned from this API.

4. As shown in Figure 11.19, the fetched data will need to be passed down via an attribute to the `PhotoBrowser` component, which in turn will need to pass it to the `PhotoList` component. Your `PhotoList` component should display `PhotoThumb` components by looping through the passed photos array data (using the `map` function); be sure to pass a photo object to `PhotoThumb`. Finally, your `PhotoThumb` component can display the title, city, and country data in the photo object that has been provided to it via props. The URL for the thumbnail image is detailed in the readme file. You will append the value of the photo object's `filename` property to that URL to display the thumbnail image.
5. While the styling is in the provided CSS file, use styled-components so that each component has its own encapsulated style definitions. Remember that this will require installing styled-components using npm.
6. Add a click event handler to the `PhotoThumb` component. As shown in Figure 11.19, you will need to implement a handler named `showImageDetails` in the `PhotoBrowser` and then pass it via props down through `PhotoList` and `PhotoThumb`.
7. Using controlled form components (see Section 11.3.4), implement the form in `EditPhotoForm`. It should populate the fields using the passed photo object. As covered in 11.3.5, the handler that will actually modify the data will have to reside in the `App` component.
8. Install the react-router-dom package using npm. Use it to implement the navigation in the header. Implement new components named Home and About (using the provided markup). Modify the `HeaderMenu` component to set up the appropriate routing. This will require making each button a child within a `<Link>` element. The render function of the `App` component will need to use `Route` elements, and the **index.js** file will need to use `BrowserRouter`, as shown in Section 11.6.1.

11.7 Chapter Summary

This chapter has provided an overview of the most popular JavaScript framework: the React library, which was originally created by Facebook. It discussed the role of JavaScript frameworks in general and demonstrated how the component-based approach of React ultimately makes for a more composable system for creating and maintaining web applications. React certainly has a learning curve, which required learning how to use props, state, and behaviors with both functional and class components. Because React uses its own JSX language, build tools are typically used to transform JSX into JavaScript: this chapter made use of the build process provided by the create-react-app application. Finally, the chapter briefly looked at the React lifecycle and how to extend React with components libraries.