

LAB 11b

INTERMEDIATE REACT

What You Will Learn

- How to use the create-react-app template.
- How to interact with the React lifecycle methods
- How to implement data flow with functional and class components
- How to implement program flow with React Router

Note

This chapter's content has been split into three labs: Lab11a, Lab11b, Lab11c.

Approximate Time

The exercises in this lab should take approximately 120 minutes to complete.

Fundamentals of Web Development, 3rd Ed

Randy Connolly and Ricardo Hoar

Textbook by Pearson
<http://www.funwebdev.com>

Date Last Revised: February 16, 2023

PREPARING DIRECTORIES

- 1 The starting `lab11b` folder has been provided for you (within the zip folder downloaded from Gumroad).

*Note: these labs use the convention of `blue background` text to indicate filenames or folder names and **`bold red`** for content to be typed in by the student.*

USING NODE, NPM, AND CREATE-REACT-APP

In this lab, we will make use of the popular `create-react-app` application and template to construct a more complicated React application. It will apply some of the same React concepts as the previous lab, but add routing and extracting data from an external API.

The `create-react-app` application makes use of `npm` and `node`, both of which you will have to install or use an environment that already has it installed.

Exercise 11b.1 — INSTALLING NODE

- 1 The mechanisms for installing Node vary based on the operating system. You can find installers at <https://nodejs.org/>.

If you wish to run Node locally on a Windows-based development machine, you will need to download and run the Windows installer from the Node.js website.

If you want to run Node locally on a Mac, then you will have to download and run the install package.

If you want to run Node on a Linux-based environment, you will likely have to run `curl` and `sudo` commands to do so. The Node website provides instructions for most Linux environments.

If you are using a cloud-based development environment, Node may be already installed in your workspace.

- 2 To run Node, you will need to use Terminal/Bash/Command Window. Verify it is working after you have finished the installation by typing the following commands:

```
node -v
npm -v
npm -v
```

The second command will display the version number of `npm`, the Node Package Manager which is part of the Node install. The third command (`npm`) is newer and might not be on your system: it is a tool for executing Node packages (though you can do so also via `npm`).

- 3 Navigate to the folder you are going to use for your source files in this lab.

- 4 Create a simple file from the command line via the following command:

```
echo "console.log('hello world')" > hello.js
```

- 5 Verify your node install works by running this file in node via the following command:

```
node hello.js
```

Not the most amazing program but you will do more exciting things later (in lab 13 on Node).

Exercise 11b.2 — INSTALLING CREATE-REACT-APP

- 1 Visit the main page for create-react-app (CRA) at <https://github.com/facebook/create-react-app>. Take a quick look at the documentation.

- 2 Using your terminal, ensure you are in your `lab11b` folder.

- 3 Run the following command:

```
npx create-react-app my-app
```

If you had an older global version of `create-react-app` installed on your computer, you will have to run the following two commands first.

```
npm uninstall -g create-react-app  
npm clear-npx-cache
```

It will take a few minutes for the `create-react-app` process to finish. On my slow university computer, it took almost 10 minutes, but only took three minutes on my newer laptop.

- 5 Switch to the `my-app` folder via:

```
cd my-app
```

- 6 Examine the folders created by `create-react-app`. Let's spend a few moments examining them.

Examine the `index.html` file in the `public` folder.

The projects created with `create-react-app` are SPA (single-page applications). This file is that one page, and contains the root element that all React components will be rendered into. You can add any global CSS, JS, or font libraries here. To better understand the folder structure of `create-react-app`, see Figure 11.15 on page 581 of Chapter 11.

- 7 Examine the `index.js` file in the `src` folder.

The code you will write for this project will be within this `src` folder. The `index.js` file renders the `<App>` element to the root element that we just looked at in `index.html`.

- 8 Examine the `App.js` file in the `src` folder. We will be replacing this content in the next exercise, but for now, we will simply look at it. Notice that it uses `import` statements to create references to other functions or classes that are part of the React infrastructure. In this lab, you will be creating a number of new components; each one will exist, like this one, in its own file.

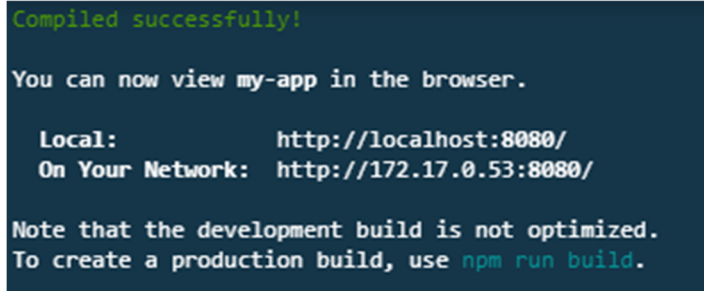
- 9 Examine the `node_modules` folder.

There will be dozens of folders, each containing JavaScript files. This folder is used to contain source code downloaded from the npm site. Some of this code is used to run the create-react-app infrastructure; some of it is used by React. The builder will combine and include any JavaScript in this folder needed for production.

- 10 To run and test our project, you need to switch to the `my-app` folder and run the project. This will start its own development web server (using node). To do this, run the following command from the terminal:

```
npm start
```

This should display a message saying compilation was successful (see Figure 11b-1). It should also automatically open a browser tab with the sample starting react app visible. You may also see warning messages about features that will be deprecated in the future. You can ignore these.



```
Compiled successfully!

You can now view my-app in the browser.

Local:      http://localhost:8080/
On Your Network: http://172.17.0.53:8080/

Note that the development build is not optimized.
To create a production build, use npm run build.
```

Figure 11b.1 – Running the create-react-app application

- 11 If a browser tab didn't automatically open with the page, then click on the Local link/url to view page in browser or copy and paste the local URL into your browser if the previous step didn't already do so. Depending on your environment, you may need to perform additional steps.

When everything works correctly, you should see the something similar to that shown in Figure 11b.2.

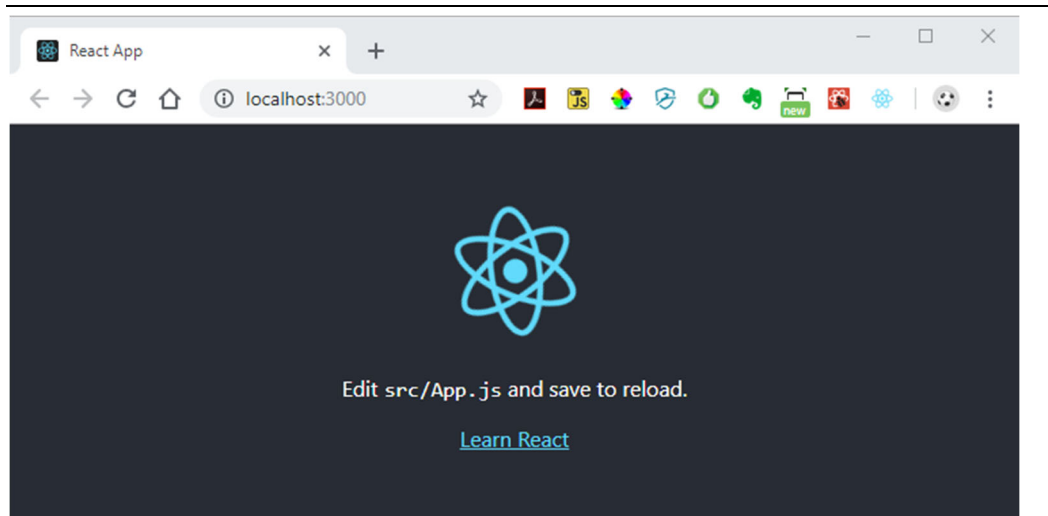


Figure 11b.2 – Finished Exercise 11b.02

Exercise 11b.3 — INSTALLING ADDITIONAL MODULES

- 1 In the Bash/Terminal, press Ctrl-C. This should stop the server and return you to the terminal prompt. We are doing this because we are going to install some additional npm modules.

We can later, at any time, restart the server by running `npm start`.

- 2 Type and run the following command:

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

This will download and install the React Router libraries in our `node_modules` folder. We will use routers to construct hyperlinks that display components rather than make http requests. You can ignore any warnings displayed in this step and the next.

- 3 Type and run the following command:

```
npm install --save lodash
```

This will download and install the Lodash array manipulation library. We will use one of its functions to make deep copies of arrays.

- 4 Open and examine the file `package.json` in the `my-app` folder within your text editor.

This file is modified when you install packages with npm and use the `--save` flag. Notice that our project has dependencies to a wide variety of packages.

- 5 Rerun the server via the command:

```
npm start
```

Every time we save a file in our project, we should see some type of message about project being “Compiled successfully”. The create-react-app has set up Webpack (another command line tool) with “listeners” on files within its folders; Webpack automatically compiles (converts from JSX to JS) any changed files.

CREATING COMPONENTS

Exercise 11b.4 — CREATING FUNCTIONAL COMPONENTS

- 1 In the `src` folder, create a folder named `components`.
In this folder, you will be placing the components you create.
- 2 Create a new file in this new folder named `HeaderBar.js`.
- 3 Add the following code to this file and save:

```
const HeaderBar = function (props) {
  return (
    <div className="header-titles">
      <h1>Travel Image App</h1>
      <p>Using create-react-app</p>
    </div>
  );
}
```

```
export default HeaderBar;
```

Notice the last line. This is needed for making this component available to our other components. What you are creating here then is a JavaScript module; you may want to review pages 493-7 in Chapter 10.

- 4 Create a new file in this same folder named `HeaderMenu.js` as follows:

```
const HeaderMenu = function (props) {
  return (
    <nav>
      <button>About</button>
      <button>Upload</button>
      <button>Download</button>
    </nav>
  );
}
export default HeaderMenu;
```

- 5 Create a new file in this same folder named `HeaderApp.js` as follows:

```
import HeaderBar from './HeaderBar.js';
import HeaderMenu from './HeaderMenu.js';

const HeaderApp = function (props) {
  return (
    <header className="header">
      <HeaderBar />
      <HeaderMenu />
    </header>
  );
}

export default HeaderApp;
```

To use another component, you have to provide the appropriate import statement. Notice here that you have added import references to the two components you just created.

- 6 Modify the `App.js` file as follows (notice some lines in the original are to be deleted):

```
import logo from './logo.svg';
import './App.css';
import HeaderApp from './components/HeaderApp.js';

function App() {
  return (
    <main>
      <HeaderApp />
    </main>
  );
}

export default App;
```

- 7 Save this file. It should compile correctly (but there may be a variety of warnings), and you should be able to view it in your browser.

- 8 If you wanted to make use of an external CSS library such as Bootstrap, you can add it to the `<head>` of the file `index.html` in the `public` folder as shown below.

```
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1,
  shrink-to-fit=no">
<meta name="theme-color" content="#000000" />
<link rel="stylesheet" href="https:// some external css library">
```

We're not using an external library, so this step is strictly informative.

- 9 Locate the `index-styles-to-copy.css` file in the starting files that have been provided to you as part of this lab. This file contains a few dozen already-created styles. Copy the content in this file and replace the content of the `index.css` file in the `src` folder with it. Test in browser.

The result should look similar to that shown in Figure 11b.3.

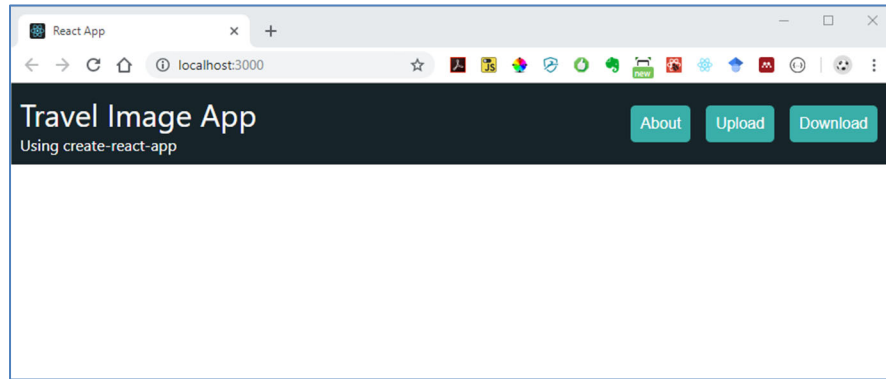


Figure 11b.3 – Finished Exercise 11b.4

Exercise 11b.5 — CREATING CLASS COMPONENTS

- 1 In the `component` folder, create a new file named `PhotoThumb.js`.
This component will display a single thumbnail version of one of the photos.
- 2 In the new file, add the following content.

```
const PhotoThumb = props => {
  const imgURL =
    `https://www.randyconnolly.com/funwebdev/3rd/images/travel/square150/5855174537.jpg`;

  return (
    <div className="photoBox">
      <figure>
        <img src={imgURL} className="photoThumb" alt="later" />
      </figure>
      <div>
        <h3>title</h3>
        <p>city, country</p>
        <button>View</button><button>♥</button>
      </div>
    </div>
  );
}
export default PhotoThumb;
```

I'm sometimes tardy in renewing my yearly https certificate; if this url doesn't work, try changing from https to http.

- 3 Test in browser. Nothing has changed. Why?

It didn't work because we haven't referenced (used) the new PhotoThumb component.

- 4 Modify `App.js` by adding the following reference to the new component and test (it won't work).

```
class App extends Component {
  render() {
    return (
      <main>
        <HeaderApp />
        <PhotoThumb />
      </main>
    );
  }
}
```

Can you see why it doesn't work? You need to import every component you access within a component.

- 5 Add the following to the top of the component and test.

```
import PhotoThumb from './components/PhotoThumb.js';
```

The component should show up. We will add in real data in a few more exercises.

You are ready to create the remaining components in this application. Figure 11b.4 illustrates the hierarchical relationships between the different components in this lab's exercise. You may also want to look at Figure 11.19 on page 595 in Chapter 11.



Figure 11b.4 – Application Structure

Exercise 11b.6 — CREATING THE APPLICATION STRUCTURE

- 1 In the `component` folder, create a component named `PhotoList.js` as follows.

```
import PhotoThumb from './PhotoThumb.js';
const PhotoList = (props) => {
  return (
    <article className="photos">
      <PhotoThumb />
      <PhotoThumb />
      <PhotoThumb />
      <PhotoThumb />
      <PhotoThumb />
      <PhotoThumb />
      <PhotoThumb />
    </article>
  );
}
export default PhotoList;
```

- 2 In the `component` folder, create a component named `PhotoBrowser.js` as follows.

```
import PhotoList from './PhotoList.js';
import EditPhotoDetails from './EditPhotoDetails.js';

const PhotoBrowser = props => {
  return (
    <section className="container">
      <PhotoList />
      <EditPhotoDetails />
    </section>
  );
}
export default PhotoBrowser;
```

- 3 In the `component` folder, create a class component named `EditPhotoDetails.js` as follows.

```
import React from "react";

class EditPhotoDetails extends React.Component {
  render() {
    const imgURL = `https://www.randyconnolly.com/funwebdev/3rd/
images/travel/medium640/5855174537.jpg`;
    return (
      <article className="details">
        <div className="detailsPhotoBox">
          <form className="photoForm">
            <legend>Edit Photo Details</legend>
            <img src={imgURL} alt="later" />

            <label>Title</label>
            <input type='text' name='title' />

            <label>City</label>
            <input type='text' name='city' />

            <label>Country</label>
            <input type='text' name='country' />
          </form>
        </div>
      </article>
    );
  }
}

export default EditPhotoDetails;
```

Notice that this component is a class component rather than a functional component. Why? Just to show/remind you that you can use either when creating components. Notice also the first line, which adds a reference to the React libraries.

- 4 Modify `App.js` as follows and then test in browser.

```
import './App.css';
import HeaderApp from './components/HeaderApp.js';
import PhotoBrowser from './components/PhotoBrowser.js';

function App() {
  return (
    <main>
      <HeaderApp />
      <PhotoBrowser />
    </main>
  );
}
```

The result should look similar to that shown in Figure 11b.5

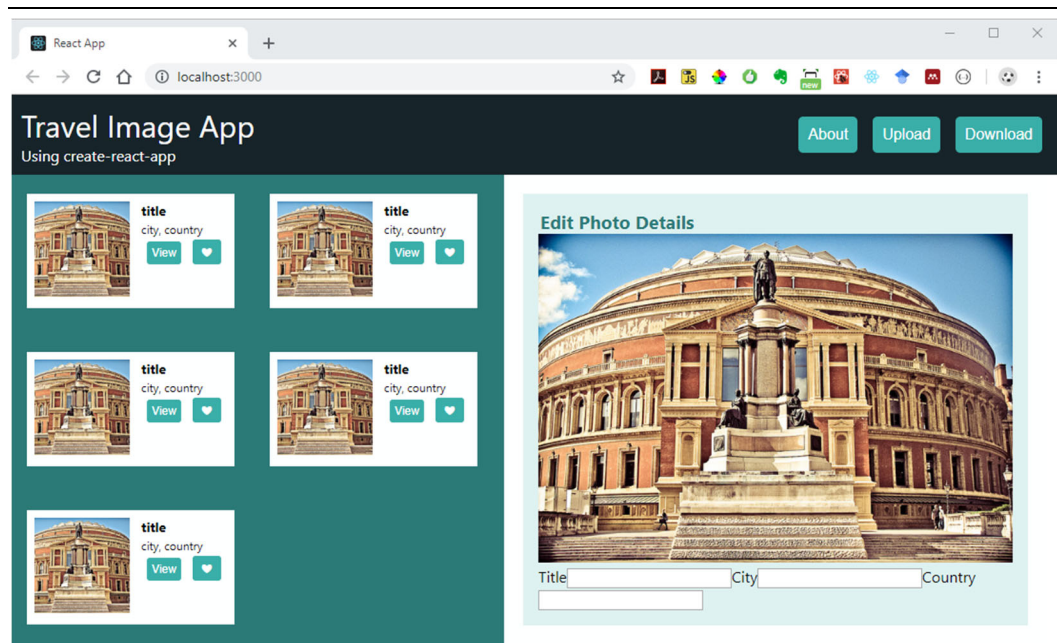


Figure 11b.5 – Finished Exercise 11b.6

Exercise 11b.7 — COMPONENT STYLES

- 1 In the `component` folder, create a file named `EditPhotoDetails.css` and add the following styles.

```
form.photoForm {
  display: grid;
}
form.photoForm label {
  display: block;
  margin-top: 0.5rem;
  margin-bottom: 0.25rem;
}
form.photoForm input {
  padding: 0.75em;
  border: 0;
  border-radius: 5px;
}
form.photoForm input:focus {
  box-shadow: 3px 3px 8px 0px rgba(0,0,0,0.49);
}
```

Each component can have its own styles. The create-react-app build tools will merge all the CSS styles into a single file when building. While style conflicts can thus still occur, it does help modularize your applications styling.

- 2 Add the following to `EditPhotoDetails.js` and test.

```
import React from "react";
import './EditPhotoDetails.css';
```

INTEGRATING EXTERNAL DATA

Most React applications need data. This data is usually fetched from an external API. In React, we have to retrieve data at a specific point in the component's life cycle. Prior to React Hooks, this was done during the `componentDidMount` event, which required using a class component instead of a functional component.

Exercise 11b.8 — FETCHING DATA IN CLASS COMPONENT

- 1 Modify the App component as follows and test to verify still works.

```
import React from "react";

class App extends React.Component {
  render() {
    return (
      <main>
        <HeaderApp />
        <PhotoBrowser />
      </main>
    );
  }
}
```

A component in React can be either a class or a function. This first exercise will show how to fetch data within a class component, which until the release of Hooks in 2019, was the only way to fetch data in React. This approach is still commonly used in legacy code or in online examples.

- 2 Add the following constructor to the App component.

```
constructor(props) {
  super(props);
  this.state = { photos: [] };
}
```

The parent component is normally the owner of the state data used by its child components.

- 3 Test the following URL in a new browser tab.

<https://randyconnolly.com/funwebdev/3rd/api/travel/images.php?iso=gb>

You will be fetching from this URL in the next step. Due to word-wrapping within the word processor, this URL below is split across several lines but in your code editor it should be a single line with no internal spaces.

- 4 Add the following method to the App component.

```
async componentDidMount() {
  try {
    const url = "https://randyconnolly.com/funwebdev/
3rd/api/travel/images.php?iso=gb";
    const response = await fetch(url);
    const jsonData = await response.json();
    this.setState( {photos: jsonData } );
  }
  catch (error) {
    console.error(error);
  }
}
```

This populates the state array with the image data retrieved from the API.

- 3 Modify the render() method of the App component as follows.

```
render() {
  return (
    <main>
      <HeaderApp />
      <PhotoBrowser photos={this.state.photos} />
    </main>
  );
}
```

In order for the state to be available to any child components, you must pass them via props.

- 4 Modify the PhotoBrowser component as follows.

```
return (
  <section className="container">
    <PhotoList photos={props.photos} />
    <EditPhotoDetails photos={props.photos} />
  </section>
);
```

- 5 Modify the PhotoList component as follows.

```
return (
  <article className="photos">
    {props.photos.map( (p) => <PhotoThumb photo={p} /> )}
  </article>
);
```

6 Modify the PhotoThumb component as follows.

```
render() {
  const imgURL = `https://www.randyconnolly.com/funwebdev/3rd/images/
travel/square150/${props.photo.filename}`;
  return (
    <div className="photoBox" >
      <figure>
        <img src={imgURL} className="photoThumb"
          title={props.photo.title}
          alt={props.photo.title} />
      </figure>
      <div>
        <h3>{props.photo.title}</h3>
        <p>{props.photo.location.city},
          {props.photo.location.country}</p>
        <button>View</button> <button>♥</button>
      </div>
    </div>
  );
}
```

7 Test.

It should work, though there will be a React warning message in the console about a missing key prop (see Figure 11b.6).

8 Fix the warning by modify the return of the PhotoList component as follows and test.

```
render() {
  return (
    <article className="photos">
      { this.props.photos.map( (p) =>
        <PhotoThumb photo={p} key={p.id} /> )}
    </article>
  );
}
```

The message should be gone, and the result should look similar to Figure 11b.7.

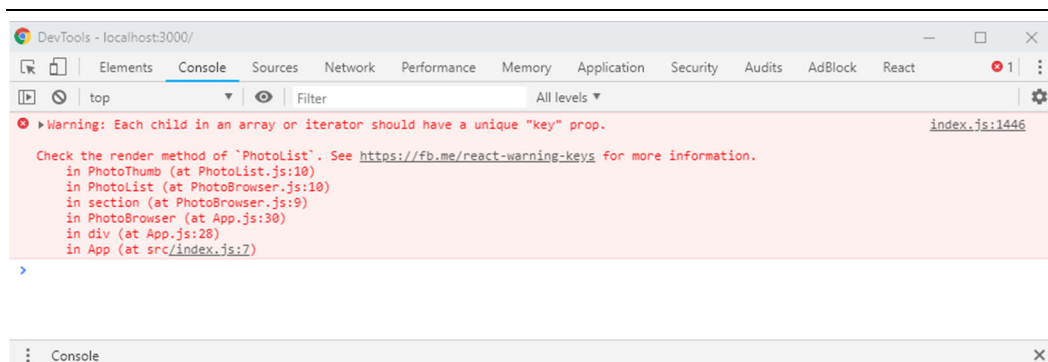


Figure 11b.6 – React warning message

Exercise 11b.9 — FETCHING DATA IN FUNCTIONAL COMPONENT

- 1 Comment out the class component in `App.js` created in the last exercise.
- 2 Replace the class component with the following functional component.

```
import React, { useEffect, useState } from 'react';
import HeaderApp from './components/HeaderApp.js';
import PhotoBrowser from './components/PhotoBrowser.js';

function App() {
  const [photos, setPhotos] = useState([]);

  useEffect( () => {
    const url =
      "https://www.randyconnolly.com/funwebdev/3rd/api/travel/
      images.php?iso=gb";
    fetch(url)
      .then( resp => resp.json() )
      .then( data => setPhotos(data))
      .catch( err => console.error(err));

  });

  return (
    <main>
      <HeaderApp />
      <PhotoBrowser photos={photos} />
    </main>
  );
}
```

- 3 Test. It should work.
- 4 Comment out the previous `useEffect()` call and replace it with the following that uses `async...await` keywords.

```
useEffect( () => {
  const getData = async () => {
    try {
      const url =
        "https://www.randyconnolly.com/funwebdev/3rd/api/travel/
        images.php?iso=gb";
      const response = await fetch(url);
      const data = await response.json();
      setPhotos(data);
    }
    catch (err) {
      console.error(err);
    }
  };
  // invoke the async function
  getData();
}, [] );
```

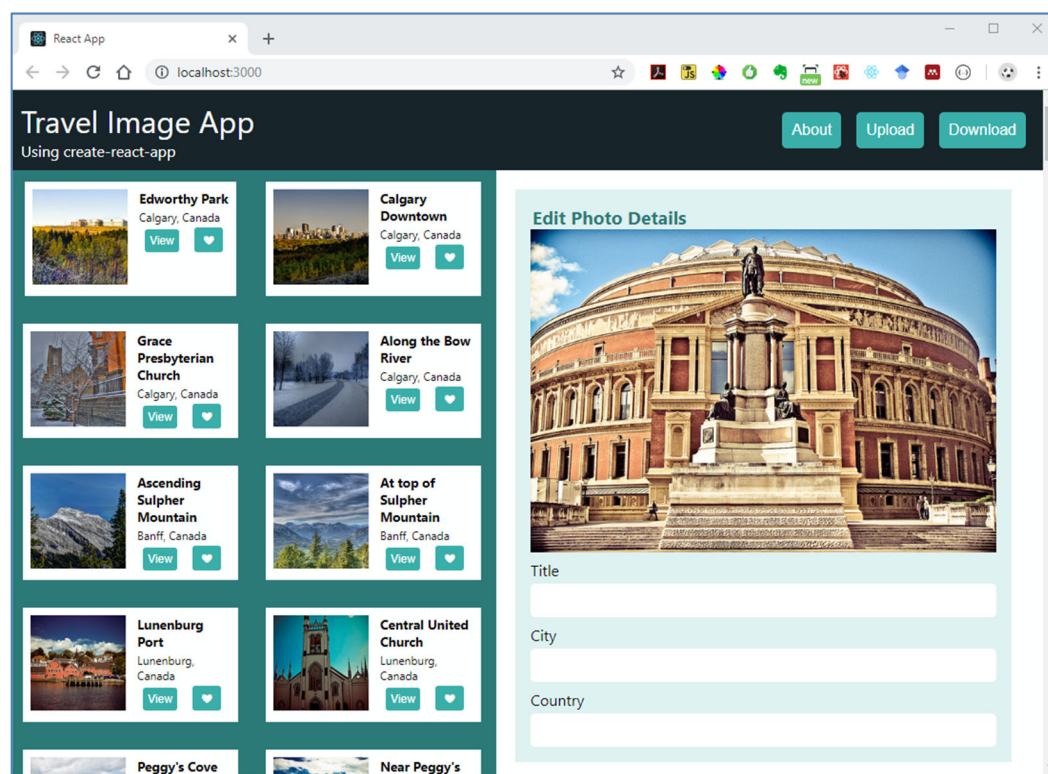



Figure 11b.7 – Finished Exercise 11b.8

Exercise 11b.10 — USING REACT DEVELOPER TOOLS EXTENSION

- 1 This exercise is optional. While you don't have to install and use the React Developer Tools to your browser, you may find it help for debugging React applications.

If you are using Chrome or Firefox, install the React Developer Tools extension/add-on.

This extension will allow you to view and examine the React components at run-time within the browser.

- 2 After you have installed, reload the previous exercise and switch to the Inspector. Click on the React or Components tab. You should be able to see inside any component and view its state and props (see Figure 11b.8).

- 3 This `$r` is a temporary variable that you can now access in the console. Switch to the console and enter `$r` and then enter `$r.props`.

This `$r` can be an easy way to examine the properties of any React component.

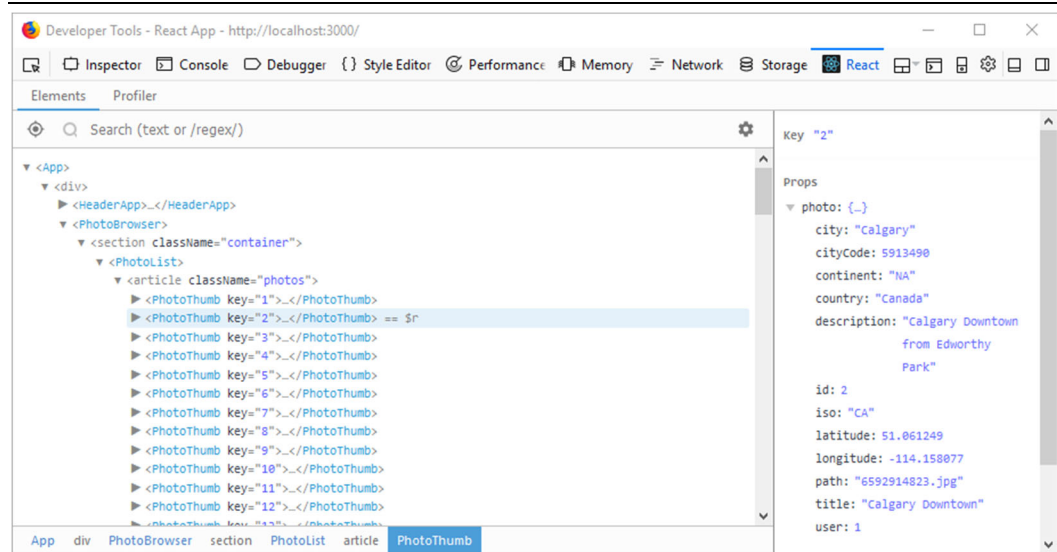


Figure 11b.8 – Using the React Developer Tools in Firefox

We are ready to continue with our app by implementing the `EditPhotoDetails` component. It needs to “know” which photo to display. The photo to display can be passed in via props.

Exercise 11b.11 — CONFIGURING EDIT FORM COMPONENT

- 1 Edit the `PhotoBrowser` component as follows.

```
import { useEffect, useState } from "react";

const PhotoBrowser = props => {
  // the first photo in fetched data has an id of 15
  const [currentPhoto, setCurrentPhoto] = React.useState(15);

  return (
    <section className="container">
      <PhotoList photos={props.photos} />
      <EditPhotoDetails photos={props.photos}
        currentPhoto={currentPhoto} />
    </section>
  );
}
```

Remember that every component can have its own state which can be shared with its children. This example uses the React Hook's approach for state. Also, in order for this state to be available to its child components, it must be passed to them via props.

- 2 Modify the `render()` method of the `EditPhotoDetails` component as follows.

```
render() {
  const id = this.props.currentPhoto;
  const imgURL =
    `https://www.randyconnolly.com/funwebdev/3rd/images/travel/medium640/`;

  // just in case, handle case of a missing photos collection
  if (this.props.photos.length > 0) {
    // find the photo object with this id
    const photo = this.props.photos.find( p => p.id === id);
    // just in case there is no match, then exit
    if (! photo) return null;
    // we have found a photo for this id, so display edit form
    return (
      <article className="details">
        <div className="detailsPhotoBox">
          <form className="photoForm">
            <legend>Edit Photo Details</legend>
            <img src={imgURL+photo.filename} alt={photo.title} />

            <label>Title</label>
            <input type='text' name='title' value={photo.title} />

            <label>City</label>
            <input type='text' name='city'
              value={photo.location.city} />

            <label>Country</label>
            <input type='text' name='country'
              value={photo.location.country} />
          </form>
        </div>
      </article>
    );
  } else {
    return null;
  }
}
```

- 4 Test in browser.

Notice the warning message in the console. Also notice that you can't change the data in the form because you haven't defined a change event handler.

Now you are ready to start implementing some event handlers. First, you will implement a click handler in the `PhotoThumb` component which will ultimately change the photo displayed in the `EditPhotoDetails` component.

Exercise 11b.12 — HANDLE THE VIEW EVENT

- 1 Modify the `PhotoThumb` component as follows (some code omitted).

```
const handleViewClick = () => {
  props.showImageDetails(props.photo.id);
}

return (
  <div className="photoBox" onClick={ handleViewClick } >
    ...
    <button onClick={ handleViewClick }>View</button>
    ...
  );
```

Why is this handler calling another method? What do we want to happen when we click `View`? We want to display the clicked photo in the `EditPhotoDetails` component. This means we have to have `PhotoBrowser` “tell” `EditPhotoDetails` that a new photo has been selected by a `PhotoThumb` component (see Figure 11b.9).

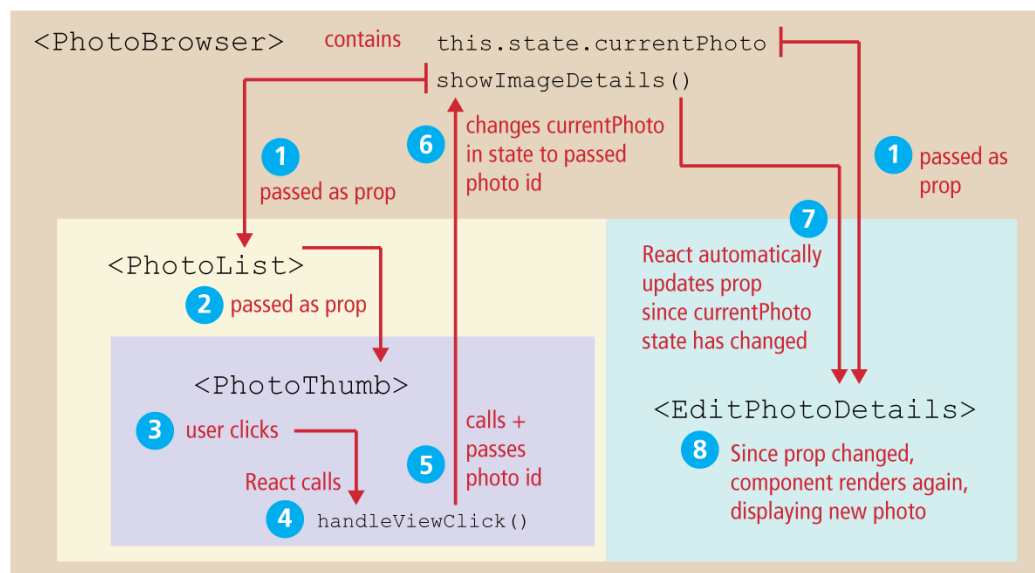


Figure 11b.9 – Event Flow

2 Edit PhotoList component as follows.

```
const PhotoList = (props) => {
  if (props.photos.length > 1) {
    return (
      <article className="photos">
        {props.photos.map( (p) =>
          <PhotoThumb photo={p} key={p.id}
            showImageDetails={props.showImageDetails} /> )}
        </article>
      );
    } else
    return null;
  }
}
```

The conditional logic is necessary since the first render will happen before we get data from the API; once the data is received, another render will happen.

4 Edit the PhotoBrowser component as follows.

```
const PhotoBrowser = props => {
  // the first photo in fetched data has an id of 15
  const [currentPhoto, setCurrentPhoto] = React.useState(15);

  const showImageDetails = (id) => {
    // change the state based on passed id
    setCurrentPhoto(id);
  }

  return (
    <section className="container">
      <PhotoList photos={props.photos}
        showImageDetails={showImageDetails} />
      <EditPhotoDetails photos={props.photos}
        currentPhoto={currentPhoto} />
    </section>
  );
}
```

5 Test.

The EditPhotoDetails component should now update whenever a user clicks in the PhotoThumb component.

Right now, the form elements don't change any of the data since EditPhotoDetails doesn't have a handle for the change event. The next exercise uses the same techniques as the previous exercise to solve this problem.

Exercise 11b.13 — ALLOW FORM TO EDIT

- 1 Add this method to the `EditPhotoDetails` component.

```

handleChange = e => {
  // find the current photo in our photo array
  const id = this.props.currentPhoto;
  const photo = this.props.photos.find( p => p.id === id);

  // update the photo using these 3 steps ...

  // 1. make a clone of the current photo object
  const clonedPhoto = { ...photo };

  // 2. update value of field that just changed
  if (e.currentTarget.name === 'title')
    clonedPhoto[e.currentTarget.name] = e.currentTarget.value;
  else
    clonedPhoto.location[e.currentTarget.name] =
      e.currentTarget.value;

  // 3. tell parent (or above) to update the state for this photo
  this.props.updatePhoto(this.props.currentPhoto, clonedPhoto);
}

```

Right now, `updatePhoto` doesn't exist yet, so we will need to add it to the App

- 2 Add the following code to the `render()` method of the `EditPhotoDetails` component.

```

<form className="photoForm">
  ...
  <label>Title</label>
  <input type='text' name='title'
    onChange={this.handleChange}
    value={photo.title} />

  <label>City</label>
  <input type='text' name='city'
    onChange={this.handleChange}
    value={photo.location.city} />

  <label>Country</label>
  <input type='text' name='country'
    onChange={this.handleChange}
    value={photo.location.country} />
</form>

```

This is a class component, so we need to add “*this*” before the function name.

- 3 Add this method to the App component.

```
const updatePhoto = (id, photo) => {
  // Create deep clone of photo array from state.
  // We will use a lodash function for that task.
  const copyPhotos = cloneDeep(photos);
  // find photo to update in cloned array
  const photoToReplace = copyPhotos.find( p => p.id === id);
  // replace photo fields with edited values
  photoToReplace.title = photo.title;
  photoToReplace.location.city = photo.location.city;
  photoToReplace.location.country = photo.location.country;
  // update state
  setPhotos(copyPhotos);
}
```

Notice how you need to make a deep copy of the photo array. By deep copy, we mean create a clone, not just of the array, but of the objects inside the array. This is necessary because we need to replace the entirety of the object in state with a completely new object.

- 4 Add this import to the top of your App component.

```
import * as cloneDeep from 'lodash/cloneDeep';
```

Remember Step 3 of Exercise 11b.3? That's where you installed the lodash package. You can learn more about lodash at <https://lodash.com>.

- 5 Update the return of App component as follows.

```
return (
  <main>
    <HeaderApp />
    <PhotoBrowser
      photos={photos}
      updatePhoto={updatePhoto} />
  </main>
);
```

Just as with the previous exercise, you need to pass the updatePhoto method down into the child components.

- 6 Update the return of PhotoBrowser component as follows.

```
return (
  <section className="container">
    ...
    <EditPhotoDetails
      photos={props.photos}
      currentPhoto={currentPhoto}
      updatePhoto={props.updatePhoto} />
  </section>
);
```

- 7 Test by clicking on a photo and then edit its title, city, and country. Notice how it changes in the PhotoThumb component as well, since the updatePhoto() method changes the one copy of the data in state.

USING REACT ROUTER

React Router is a package that allows a developer to configure routes. What is a route? In normal HTML, you use hyperlinks (anchor tags) to jump from page to page in your application. But in a single-page application, there is only one page. Using React Router, you can turn links to different pages into links to display React components (on the same page).

The React Router package includes quite a few components. In this lab, we will use just three: `<BrowserRouter>`, `<Route>`, and `<Link>`.

Exercise 11b.14 — ADDING THE REACT ROUTER

- 1 In the `src` folder, modify the `index.js` file as follows.

```
import { BrowserRouter } from 'react-router-dom';
...
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>,
  document.getElementById('root')
);
```

You installed the React Router back in Exercise 11b.3.

- 2 Create a new component named `Home.js` in the `components` folder with the following content.

```
import { Link } from 'react-router-dom';

const Home = props => {
  let imgUrl =
    "https://www.randyconnolly.com/funwebdev/3rd/images/travel/large1600/9496792166.jpg";
  return (
    <div className = 'banner'
      style = {{ backgroundImage: `url(${imgUrl})`,
        height: '800px',
        backgroundSize: 'cover',
        backgroundPosition: 'center center',
        backgroundRepeat: 'no-repeat',
      }}>
      <div>
        <h1>Travel Photos</h1>
        <h3>Upload and Share</h3>
        <p>
          <Link to='/browse'>
            <button>Browse</button>
          </Link>
        </p>
      </div>
    </div>
  )
};
```



```

        <Link to='/about'>
          <button>About</button>
        </Link>
      </p>
    </div>
  </div>
);
};

```

```
export default Home
```

Notice the `<Link>` elements. They are the React equivalent of the `<a>` tag. Instead of specifying a URL, they indicate which component to display. Notice also that it sets the various background image properties within the React component rather than in the CSS.

- 3 Create a new component named `About.js` in the `components` folder with the following content.

```

const About = function (props) {
  return (
    <div className="header-titles">
      <h1>About</h1>
      <p>You can make this more fancy</p>
    </div>
  );
}
export default About;

```

- 4 Edit `HeaderMenu` component as follows (some code omitted).

```

import { Link } from 'react-router-dom';

const HeaderMenu = function (props) {
  return (
    <nav>
      <Link to='/home'>
        <button>Home</button>
      </Link>
      <Link to='/browse'>
        <button>Browse</button>
      </Link>
      <Link to='/about'>
        <button>About</button>
      </Link>
    </nav>
  );
}

```

- 4 Edit App component as follows (some code omitted).

```
import { Routes, Route } from 'react-router-dom';
import Home from './components/Home.js';
import About from './components/About.js';
...
return (
  <main>
    <Routes>
      <HeaderApp />
      <Route path="/" element={<Home />} />
      <Route path="/home" element={<Home />} />
      <Route path="/about" element={<About />} />
      <Route path="/browse"
        element={<PhotoBrowser photos={photos}
          updatePhoto={updatePhoto} />} />
    </Routes>
  </main>
);
...
```

These routes describe which components to display based on the path request. In the third Route, since we have to display the complex PhotoBrowser component, it requires passing the component with its props via a function in the render property.

- 5 Test. It should default to the Home component, and only go to the PhotoBrowser when you click on one of the Browse buttons.

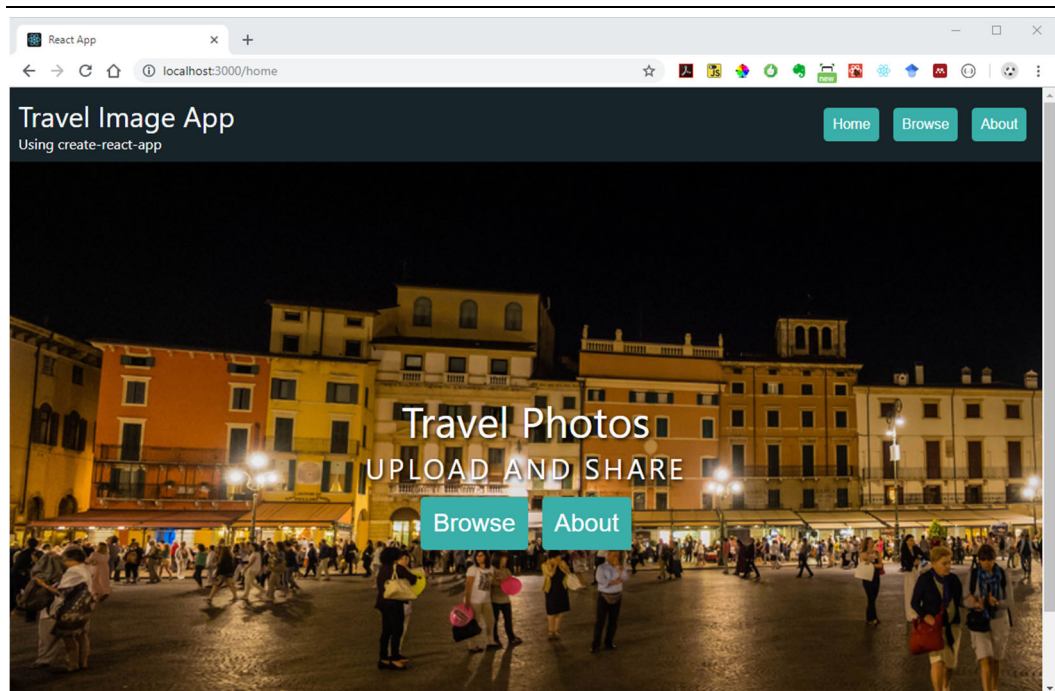


Figure 11b.10 – Finished Exercise 11b.13