

LAB 20b

INTERMEDIATE REACT

What You Will Learn

- How to use the create-react-app template.
- How to work with React components and use props and state
- How to add behaviors to your React components
- How to implement program flow with React Router

Approximate Time

The exercises in this lab should take approximately 100 minutes to complete.

Fundamentals of Web Development, 3rd Ed

Randy Connolly and Ricardo Hoar

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

Date Last Revised: Feb 2, 2020

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.

PREPARING DIRECTORIES

- 1 If you haven't done so already, create a folder in your personal drive for all the labs for this book.
- 2 From the main labs folder (either downloaded from the textbook's web site using the code provided with the textbook or in a common location provided by your instructor), copy the folder titled `Lab20b` to your course folder created in step one.

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 20b.1 — INSTALLING NODE

- 1 The mechanisms for installing Node vary based on the operating system.

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 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 in the Node lab.

Exercise 20b.2 — INSTALLING CREATE-REACT-APP

- 1 Visit the main page for create-react-app at <https://github.com/facebook/create-react-app>. Take a quick look at the readme document.
- 2 If you don't have npx, you will have to install create-react-app first via the following.

```
npm install -g create-react-app
```

This will globally install the create-react-app on your computer (or development workspace if using something like cloud9).

- 3 Using your terminal, ensure you are in your lab20b folder.
- 4 Now that create-react-app is installed (if using npm only), you can use it to create a starting project.

Run the following command if you have npx:

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

Run the following command if you do **not** have npx:

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

It will take a few minutes to finish this process. On my slower desktop computer it took almost 10 minutes!

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

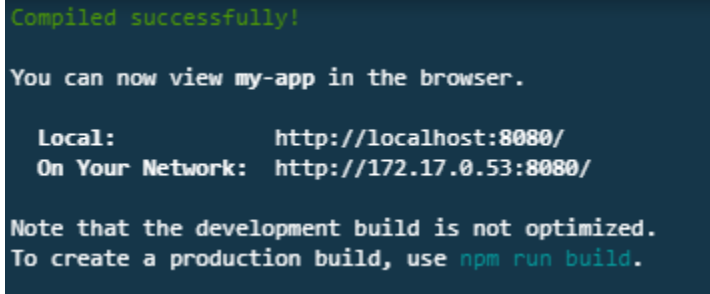
- 7 Examine the `index.js` file in the `src` folder.

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 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 20b-1). It might also automatically open a browser tab with the sample starting react app visible.



```
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 20b.1 – Running the create-react-app application

- 10 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 perform additional steps.

When everything works correctly, you should see the something similar to that shown in Figure 20b.2.

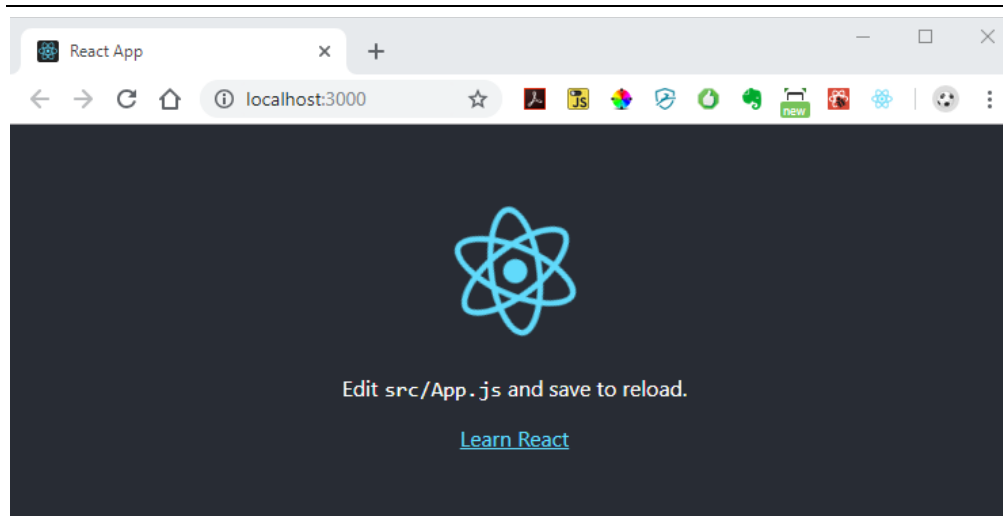


Figure 20b.2 – Finished Exercise 20b.02

Exercise 20b.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 executing `npm start`.

- 2 Type and run the following command:

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

This will download and install the React Router libraries. 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 Examine the file `package.json` in the `my-app` folder.

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 “watches” on files within its folders; Webpack automatically compiles (converts from JSX to JS) any changed files.

CREATING COMPONENTS

Exercise 20c.4 — CREATING FUNCTIONAL COMPONENTS

- 1 In the `src` folder, create a folder named `components`.
In this folder, you will be placing your stateless functional components.

- 2 Create a new file in this new folder named `HeaderBar.js`.

- 3 Add the following code to this file and save:

```
import React from 'react';

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 first line. This adds a reference to the React libraries. This particular component is just presentational; it contains no state or methods so making it a function simplifies things. Notice also the last line. This is needed for making this component available to our other components.

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

```
import React from 'react';

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 React from 'react';
import HeaderBar from './HeaderBar.js';
import HeaderMenu from './HeaderMenu.js';

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

export default HeaderApp;
```

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

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

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

export default App;
```

To use another component, you have to provide the appropriate import statement. Notice here that you have added an import reference to the component you just created. You can then make use of that component in the App's render method.

- 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 20b.3.

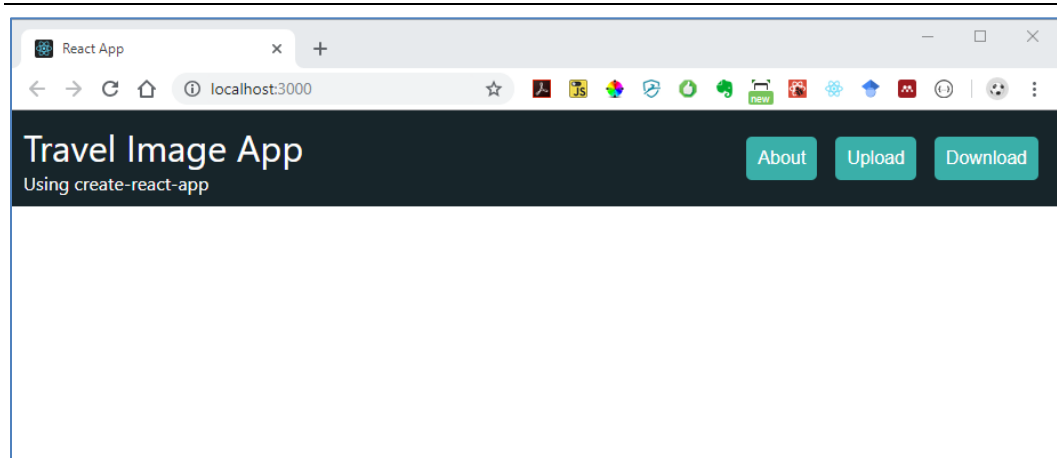


Figure 20b.3 – Finished Exercise 20b.4

Exercise 20b.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.

```
import React from "react";
class PhotoThumb extends React.Component {
  render() {
    const imgURL = `https://storage.googleapis.com/funwebdev-3rd-travel/square-medium/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
```


- 3 Test in browser. Nothing has changed. Why?
- 4 Modify `App.js` by adding the following reference to the new component and test.

```
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 20b.4 illustrates the hierarchical relationships between the different components in this lab's exercise.



Figure 20b.4 – Application Structure

Exercise 20b.6 — CREATING THE APPLICATION STRUCTURE

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

```
import React from "react";
import PhotoThumb from './PhotoThumb.js';
class PhotoList extends React.Component {
  render() {
    return (
      <article className="photos">
        <PhotoThumb />
        <PhotoThumb />
        <PhotoThumb />
        <PhotoThumb />
        <PhotoThumb />
      </article>
    );
  }
}
export default PhotoList
```

```
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
```

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

```
import React from "react";
import PhotoList from './PhotoList.js';
import EditPhotoDetails from './EditPhotoDetails.js';

class PhotoBrowser extends React.Component {
  render() {
    return (
      <section className="container">
        <PhotoList />
        <EditPhotoDetails />
      </section>
    );
  }
}

export default PhotoBrowser
```

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

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

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

The result should look similar to that shown in Figure 20b.5

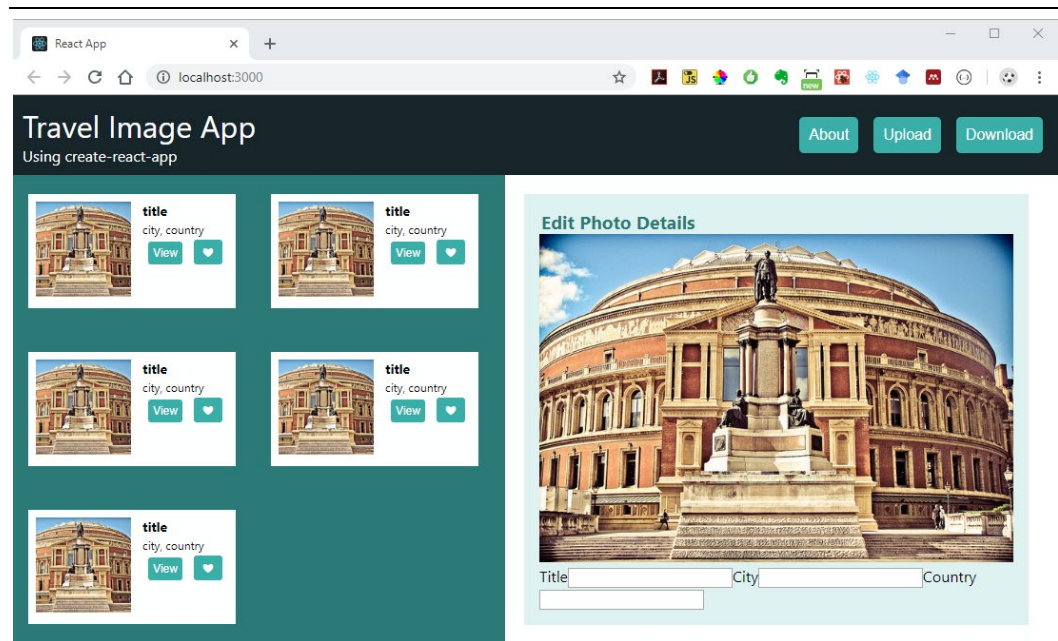


Figure 20b.5 – Finished Exercise 20b.6

Exercise 20b.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 when building.

- 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. This is typically done during the `componentDidMount` event, which will require using a class component instead of a functional component.

Exercise 20b.8 — FETCHING DATA

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

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

- 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.

- 2 Add the following method to the `App` component.

```
async componentDidMount() {  
  try {  
    const url = "http://randyconnolly.com/funwebdev/  
3rd/api/travel/images.php";  
    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 `render()` method of the `PhotoBrowser` component as follows.

```

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

```

- 5 Modify the return of the `render()` method of the `PhotoList` component as follows.

```

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

```

- 6 Modify the `render()` method of the `PhotoThumb` component as follows.

```

render() {
  const imgUrl = `http://www.randyconnolly.com/funwebdev/3rd/images/
travel/square150/${this.props.photo.filename}`;
  return (
    <div className="photoBox" >
      <figure>
        <img src={imgUrl} className="photoThumb"
          title={this.props.photo.title}
          alt={this.props.photo.title} />
      </figure>
      <div>
        <h3>{this.props.photo.title}</h3>
        <p>{this.props.photo.location.city},
          {this.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 20b.6).

- 8 Fix the warning by modify the return of the `render()` method 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 20b.7.

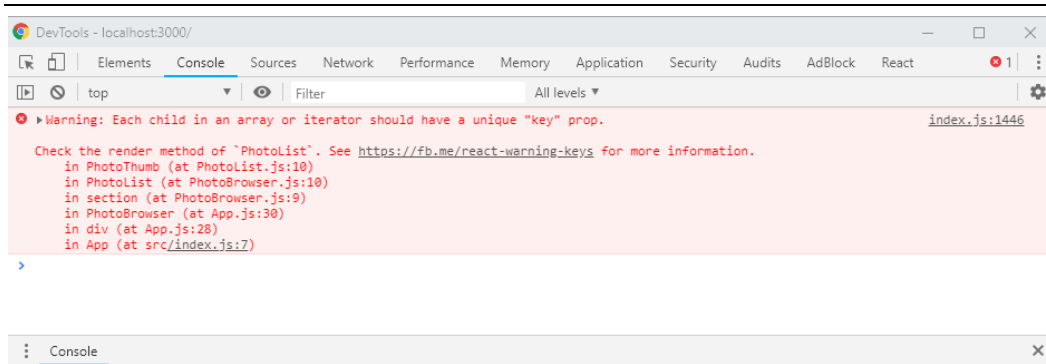


Figure 20b.6 – React warning message

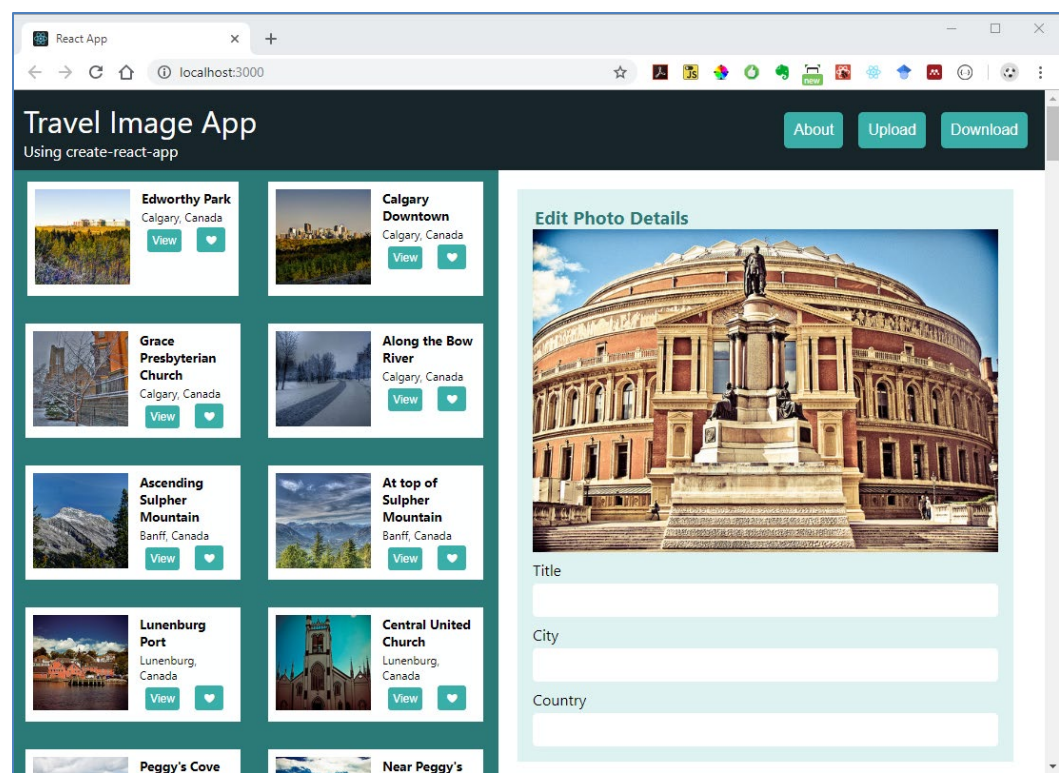


Figure 20b.7 – Finished Exercise 20b.8

Exercise 20b.9 — USING REACT DEVELOPER TOOLS EXTENSION

- 1 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 20b.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.

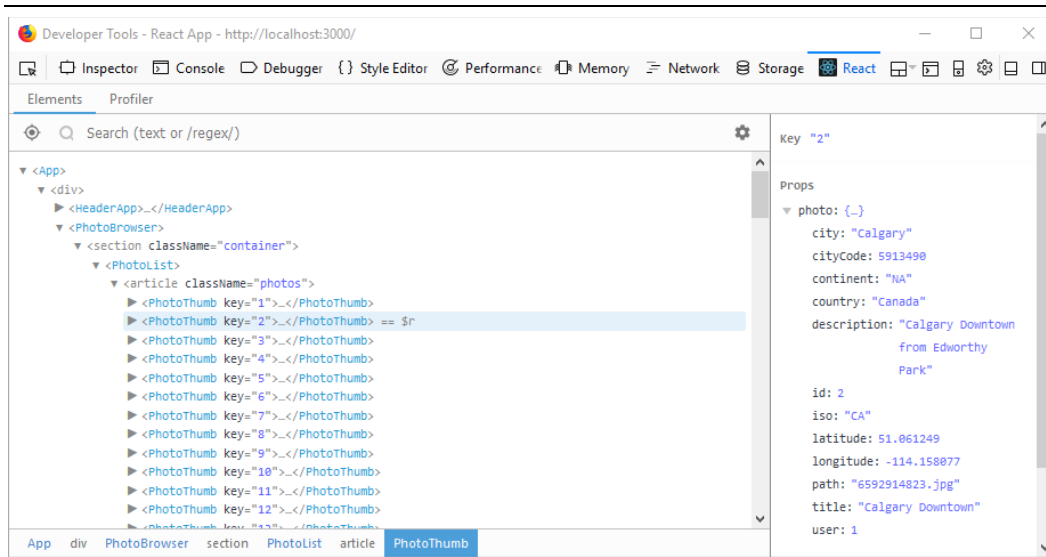


Figure 20b.8 – Using the React Developer Tools in FireFox

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 20b.10 — CONFIGURING EDIT FORM COMPONENT

- 1 Add the following constructor to the `PhotoBrowser` component.

```
constructor(props) {
  super(props);
  this.state = { currentPhoto: 1 };
}
```

Remember that every component can have its own state which can be shared with its children.

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

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

      <EditPhotoDetails photos={this.props.photos}
        currentPhoto={this.state.currentPhoto} />
    </section>
  );
}
```

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

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

```

render() {
  const id = this.props.currentPhoto;
  const imgURL =
    `http://www.randyconnolly.com/funwebdev/3rd/images/travel/medium640/`;
  // just in case, handle missing photos in the props
  if (this.props.photos.length > 0) {
    // find the photo object with this id
    const photo = this.props.photos.find( p => p.id === id);
    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 20b.11 — HANDLE THE VIEW EVENT

- 1 Modify the `render()` method of the `PhotoThumb` component as follows (some markup omitted).

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

- 2 Add this method to the `PhotoThumb` component.

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

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 to have `PhotoBrowser` “tell” `EditPhotoDetails` that a new photo has been selected by a `PhotoThumb` component (see Figure 20b.9).

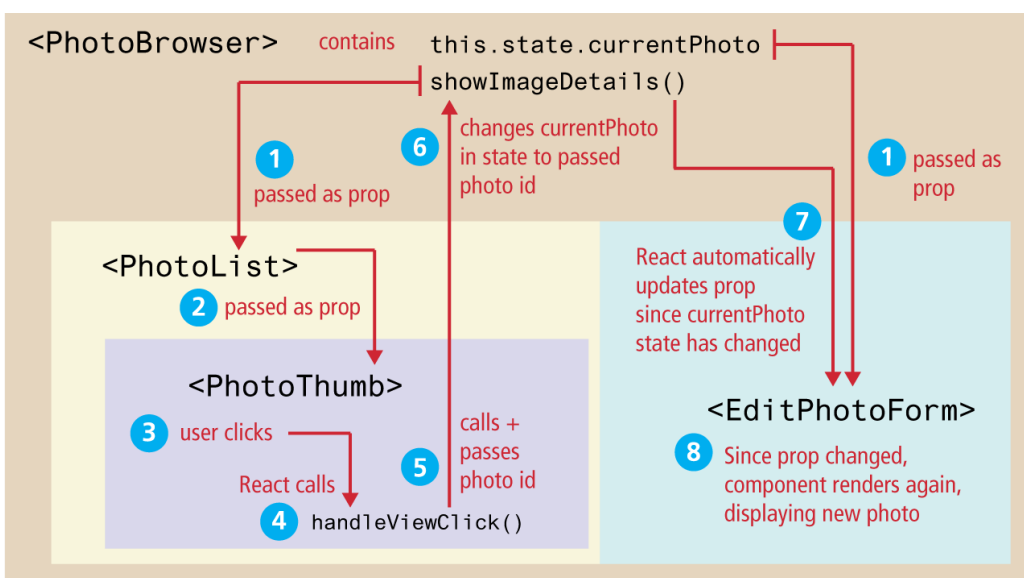


Figure 20b.9 – Event Flow

- 3 Edit the `render()` method in `PhotoList` as follows.

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

We're adding the conditional logic since `render()` happens before we get data from the API.

- 4 Edit the `render()` method in `PhotoBrowser` as follows.

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

- 5 Add the `showImageDetails()` method to `PhotoBrowser`:

```
showImageDetails = (id) => {
  this.setState({ currentPhoto: id });
}
```

- 6 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 20b.12 — 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>
```

- 3 Add this method to the `App` component.

```
updatePhoto = (id, photo) => {
  // Create deep clone of photo array from state.
  // We will use a lodash function for that task.
  const copyPhotos = cloneDeep(this.state.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
    this.setState( {photos: 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 20b.3? That's where you installed the lodash package. You can learn more about lodash at <https://lodash.com>.

- 5 Update the render() method of App component as follows.

```

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

```

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

- 6 Update the render() method of PhotoBrowser component as follows.

```

render() {
  return (
    <section className="container">
      ...
      <EditPhotoDetails
        photos={this.props.photos}
        currentPhoto={this.state.currentPhoto}
        updatePhoto={this.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 20b.13 — ADDING THE REACT ROUTER

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

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

```
ReactDOM.render(<BrowserRouter><App /></BrowserRouter>,
  document.getElementById('root'));
```

You installed the React Router back in Exercise 20b.3.

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

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

class Home extends React.Component {
  render() {
    let imgUrl =
      "http://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>
            <Link to='/about'>
              <button>About</button>
            </Link>
          </p>
        </div>
      </div>
    );
  }
}
```

```

        </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 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 { Route } from 'react-router-dom';
class App extends Component {
  ...
  render() {
    return (
      <main>
        <HeaderApp />
        <Route path='/' exact component={Home} />
        <Route path='/home' exact component={Home} />
        <Route path='/browse' exact
          render={ (props) =>
            <PhotoBrowser
              photos={this.state.photos}
              updatePhoto={this.updatePhoto} />
          } />
      </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. The About link doesn't do anything since it doesn't exist yet.

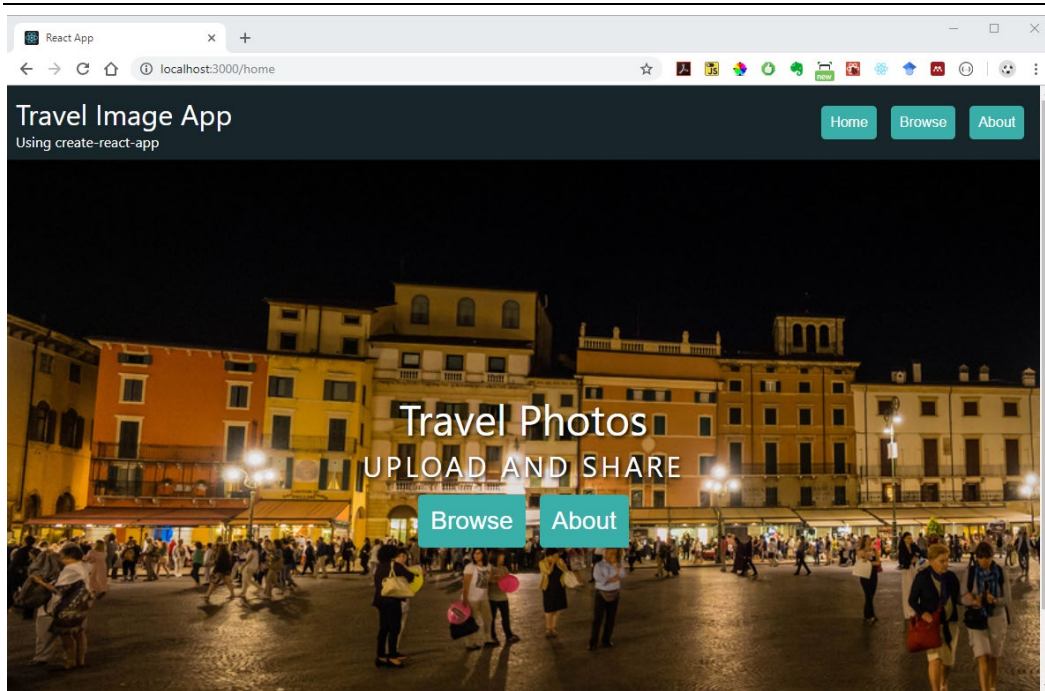


Figure 20b.10 – Finished Exercise 20b.13

Test Your Knowledge

- 1 Create an `About` component and update the routing and navigation buttons so it is displayed when user clicks on one of the `About` buttons.
- 2 The `PhotoThumb` component already has an `Add Favorite` button (the heart icon). You are going to implement its functionality. When the user clicks on this button, it will add a new favorite to an array stored in state, which should update the display in the `Favorites` component (it will be blank row at first between the header and the photo browser).

This will require creating two new components: one called `Favorites`, the other called `FavoriteItem` (there is already a CSS class defined in `index.css` named `favorites`). `Favorites` should display a list of `FavoriteItem` components. Since the `favorites` CSS class uses grid layout, your `FavoriteItem` component will need to wrap the image in a block-level item, such as a `<div>`.

You will also need to add the `Favorites` component to the `render()` method of `App`. Figure 20b.10 illustrates the finished version.

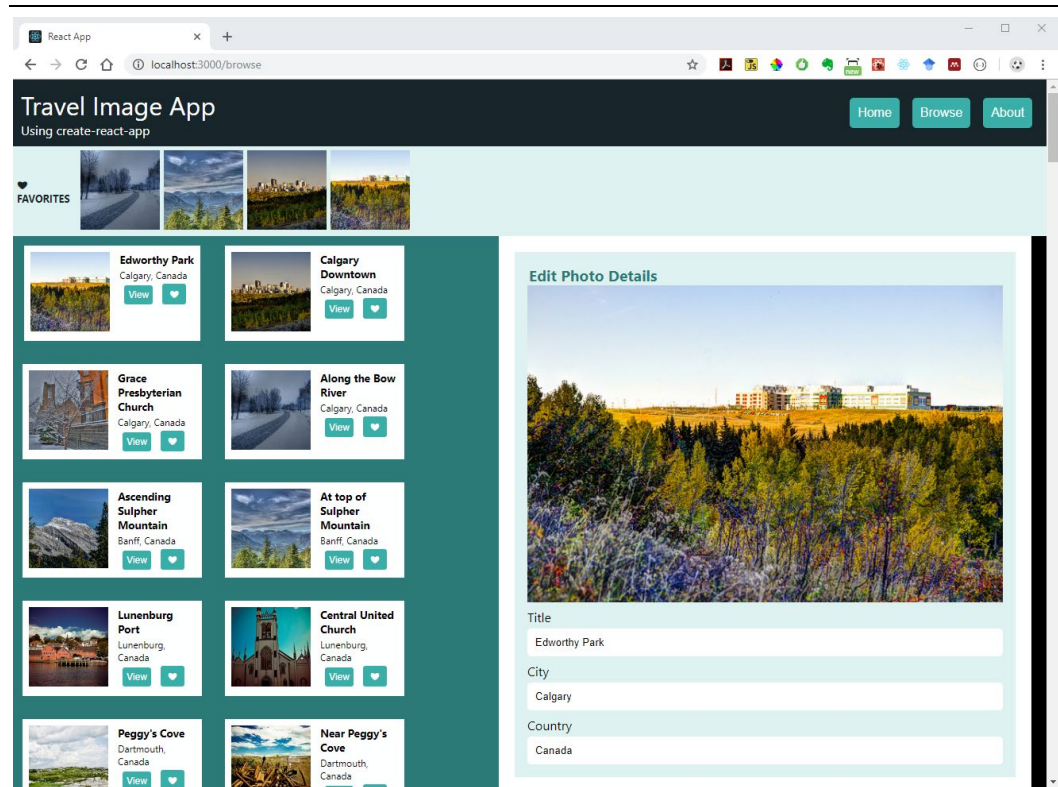


Figure 20b.11 – Test Your Knowledge