React Fundamentals Instructor Guide

Author

Drew Fierst

General course notes

This course is designed to be conducted in a workshop style, where the students are working through demonstrations alongside the instructor during the presentation of each lesson. Most of the actual teaching happens during the coding demos rather than lecturing about a slide.

This approach has been found to keep the students more engaged during the class. It is more manageable in a face-to-face scenario where the instructor is onsite but experience has shown that this approach simply takes too long in a virtual class.

Some students will prefer to watch and take notes instead of working alongside the instructor. They will still get a chance to practice their new skills, as there are exercises at the end of each lesson designed for the students to complete on their own.

If you do not perform the demos during class, this document will at least provide an overview of what changes were implemented (and in which files) for each lesson, indicating what should be reviewed with the students.

Classfiles notes

This course has classfiles associated with it.

The 'demos' directory contains a directory for each lesson with sample application files relevant to that lesson.

The 'exercises' directory contains a directory for each lesson with starting files for a self-paced exercise students should complete at the end of each lesson. Each exercise directory contains an 'instructions.txt' file with step-by-step instructions for the exercise.

The 'finished' directory contains the completed exercises.

The 'node_modules' directory contains the dependencies for all of the applications.

Course Agenda

Day 1

- Introduction
- React Syntax and Basics
- Dynamic Content
- Styling Content
- Debugging

Course Agenda

Day 2

- Components
- Web Server Interactions
- Routing
- Forms

Course Agenda

Day 3

- Managing State with Redux
- Async Redux
- Testing
- Transitions and Animations

Lesson 1 – Introduction

General notes

For a more workshop style class, have the students do the following demos with you, that way they have some experience with the topics before they are tasked with doing the exercise.

Demos can be done on jsbin.com or in local files. If the local file option is used, the files should be run with Node.js to demonstrate them.

Performing each demo provides excellent opportunity to discuss why each step is being taken and what the various options are at each step.

"let" and scope demo

```
//use jsbin.com or create a local file and run with node (01-scope.js)
function testScope() {
  if (true) {
    var msg = 'Hello world';
  }
  console.log(msg);
}
testScope();

//execute
//change var to let and execute again
```

"const" demo

```
//use jsbin.com or create a local file and run with node (02-const.js)
const product = {
  name: 'Widget',
  price: 4.95
};
const rate = 42;

product.price = 5.50;
console.log('Price is:', product.price);

rate = 50;
console.log('Rate is:', rate);
```

Arrow function demo

```
//use jsbin.com or create a local file and run with node (03-arrow-function.js)
//demo all variations of syntax
const doubleIt = (val) => {
  return val * 2;
};
const doubleIt = (val) => val * 2;
const doubleIt = val => val * 2;

console.log('twice 2 = ', doubleIt(2));
console.log('twice 5 = ', doubleIt(5));
```

Arrow function context demo – problem

```
//use jsbin.com or create a local file and run with node (04-context-problem.js)
//if necessary, review how the value of "this" can change based on how a fn is invoked
function doLater(fn) {
  setTimeout(fn, 3000);
}

const person = {
  name: 'Miguel',
  getGreeter: function() {
    return function() { console.log('Hi my name is ', this.name); }
  }
}
doLater(person.getGreeter());
// logs "Hi my name is undefined"
```

Arrow function context demo – solution

```
//use jsbin.com or create a local file and run with node (05-context-solution.js)
function doLater(fn) {
  setTimeout(fn, 3000);
}

const person = {
  name: 'Miguel',
  getGreeter: function() {
    return () => { console.log('Hi my name is ', this.name); }
  }
}
doLater(person.getGreeter());
// logs "Hi my name is Miguel"
```

Exports and Imports Demo

```
//create a local files and run with node (06a-mylib.mjs)
//NOTE: to run, must use --experimental-modules switch (and files must use .mjs extension)
//in one file, create a resource and export it
const logger = {
  logError() {
    console.log('An error happened');
  },
  logMessage() {
    console.log('Something interesting happened');
  }
};

export default logger;
```

Exports and Imports demo continued

```
//in another file, import the resource from the first file (06-imports.mjs)
//NOTE: to run, must use --experimental-modules switch (and files must use .mjs extension)
import logger from './loggerLib';  //or whatever the file name is

logger.logMessage();
logger.logError();
logger.logMessage();

//execute with a line of code like:
//  node --experimental-modules importTest.mjs
```

Class demo

```
//use jsbin.com or create a local file and run with node (07-classes.js)
class Person {
  constructor(name, email) {
    this.name = name;
    this.email = email;
  }
  sayHello() {
    console.log('Hi my name is ' + this.name + ' and you can contact me at '
      + this.email + '.');
  }
}

const p1 = new Person('Felicia', 'felicia@example.com');
p1.sayHello();
```

Inheritance Demo

```
//continue adding to the same file (07-classes.js)
class Employee extends Person {
  constructor(name, email, title) {
    super(name, email);
    this.title = title;
  }
}

const e1 = new Employee('Raj', 'raj@example.com', 'Developer/Analyst');
e1.sayHello();
```

Spread operator demo

```
//use jsbin.com or create a local file and run with node (08-spread.js)
const o1 = {
  size: 'medium',
  color: 'blue',
  price: 12.95
}
const o2 = { ...o1 };
o2.price = 14.95;
const o3 = { ...o1, price: 9.95, category: 'shoes' };

console.log(o1);
console.log(o2);
console.log(o3);
```

Rest operator demo

```
//use jsbin.com or create a local file and run with node (09-rest.js)
function requestVehicle(type, color, ...options) {
  console.log('You want a ' + color + ' ' + type + ' with the following options:');
  console.log(options.join(', '));
}

requestVehicle('car', 'blue', 'alloy wheels', 'premium stereo');
requestVehicle('truck', 'silver', 'cruise control', 'leather seats');
```

Object destructuring demo

```
//use jsbin.com or create a local file and run with node (10-destructuring.js)
function calculateVolume({ radius, height }) {
  const volume = radius * radius * Math.PI * height;
  console.log('The volume of the cylinder is:', volume);
}

const dimensions = { height: 2, radius: 1.3 };
calculateVolume(dimensions);
```

Array methods demo

```
//use jsbin.com or create a local file and run with node (11-arrays.js)
function logAll(msg, arr) {
  console.log();
  console.log(msg);
  arr.forEach(elt => console.log(elt));
}

const nums = [1, 2, 3, 4, 5, 6, 7, 8, 9];
logAll('all numbers', nums);

const odds = nums.filter(elt => elt % 2 === 1);
logAll('odd numbers', odds);
```

React demo – step 1

```
//go to codepen.io, create a new pen
//create simple html and css for a couple of product cards (react-demo/12a-product.html)
<div class="product-card">
  <h3>Widget</h3>
  <dl>
    <dt>Color</dt><dd>Blue</dd>
    <dt>Price</dt><dd>5.95</dd>
  </dl>
</div>
```

React demo – step 1

```
//create simple html and css for a couple of product cards (react-demo/12a-product.css)
.product-card { border: 1px solid #9999FF;
  width: 300px;
  padding: 1em;
  border-radius: 4px;
  background-color: #F0F0F0;
  display: inline-block;
}
dt, dd { display: inline-block;
  margin: 0;
}
dt { width: 20%; font-weight: bold; }
dd { width: 80%; }
```

React demo – step 2

```
//point out the duplication of the content – difficult to maintain!
//in the JS panel click the gear icon and import React and ReactDOM
//also choose Babel as the JavaScript preprocessor (react-demo/12b-component.js)
function Product() {
  return (
    //copy/paste the content for one of the product cards here
    //NOTE: change any "class" attributes to "className"
  );
}
//in the html panel, change one of the cards into a div with an ID
//(react-demo/12b-product.html)

//in the JS panel, use ReactDOM.render() to render the component to the div
ReactDOM.render(<Product />, document.querySelector('#card1');
```

React demo – step 3

```
//add props to the component (react-demo/12c-component.js)
function Product(props) {
  return (
    //replace hard-coded data with {props.name}, etc.
  );
}

ReactDOM.render(<Product name="Widget" . . . >, document.querySelector('#card1'));

// in the HTML, change the second card into a div with an ID
// (react-demo/12c-product.html)

// in the JS, invoke ReactDOM.render() a second time to display the second card
```

React demo – step 4

```
//in the HTML, use just one div with id="app" (react-demo/12d-product.html)

//in the JS, put both product component usages into some JSX in a variable
//and simplify to use only one ReactDOM.render() call (react-demo/12d-component.js)
let app = (
  <div>
    <Product name="Widget" . . . />
    <Product name="Gear" . . . />
  </div>
);
ReactDOM.render(app, document.querySelector('#app'));

//point out that step 3 uses React as a library as you would in a multi-page app
//and step 4 is the approach you would use in a single-page app
```

Lesson 2 – React Syntax and Basics

General notes

For a more workshop style class, have the students do the following demos with you, that way they have some experience with the topics before they are tasked with doing the exercise.

Performing each demo provides excellent opportunity to discuss why each step is being taken and what the various options are at each step.

Install create-react-app demo

```
//execute at a command prompt
npm install –g create-react-app@3.4.1
```

Create application demo

```
//in the demos/lesson02 directory execute:
create-react-app demos

//change the command prompt down into the newly generated application directory
//and run it with:
npm start
```

Tour of application

```
// show and describe package.json

// show and describe node_modules

// show and describe public directory

// show and describe index.html (point out script tags and <div id="root"> )

// show manifest.json and describe its role in PWAs
```

Application tour continued

```
// show and describe the src directory
// point out index.js where it renders the application on the root element
//  note the import of app.js

// show and describe app.js
//  modify the JSX content, replacing the default with a simple hello world message
//  (can then remove the logo file and its import in app.js)
return ( <div className="App">
    <h1>Hello World!</h1>
  </div> );

// show and describe app.css – note that these are global styles, not scoped to app.js
//  (can clean up unused styles, if desired)
// show and describe index.css – also global styles
```

Component demo

```
//in index.js change ReactDOM.render() to render a simple HTML tag instead of the component
ReactDOM.render(<h1>Test</h1>, document.getElementById('root');

// mention ReactDOM can render anything, but HTML content wouldn't be reactive
// reset the code to render the app component

// examine app.js and describe class syntax for components
```

React element demo

```
// in app.js comment out the JSX and replace with React.createElement():
return React.createElement('div', {className: 'App'}, '<h1>Hello World</h1>');
//oops! This doesn't work.
//NOTE: the following will work, but it still uses JSX:
return React.createElement('div', {className: 'App'}, <h1>Hello World</h1>);

//Try:
return React.createElement('div', {className: 'App'},
  React.createElement('h1', null, 'Hello World'));

//refresh in browser, much better!
//return to using JSX
```

Multiple root elements demo

```
//still in App.js add a second root element and demo the error
return (
  <div className="App"> . . . </div>
  <p>Some other content</p>
)

//modify to array syntax and demo success
return ([
  <div className="App"> . . . </div>,
  <p>Some other content</p>
])
```

Multiple root elements demo

```
//modify to <React.Fragment> syntax and demo
return (
  <React.Fragment>
    <div className="App"> . . . </div>
    <p>Some other content</p>
  </React.Fragment>
)

//modify to empty element syntax and demo
return (
  <>
    <div className="App"> . . . </div>
    <p>Some other content</p>
  </>
)
```

Functional component demo

```
//create a new file src/Product.js
import React from 'react';
function Product() {
  return (<p>Each Widget costs $4.95</p>)
}
export default Product;

//import it into App.js and render it after the <h1>
import Product from './Product';

<div className="App">
  <h1>Hello World!</h1>
  <Product></Product>
</div>
```

Dynamic content demo

```
//modify Product.js to use (Math.random() * 15).toFixed(2) to create a dynamic price
return (<p>Each Widget costs $(Math.random() * 15).toFixed(2)</p>);

//demo that it is treated as literal content
//surround with { } and demo that it is now dynamic content
return (<p>Each Widget costs ${(Math.random() * 15).toFixed(2)}</p>);

//add a couple more instances of the component in App.js
<div className="App">
  <h1>Hello World!</h1>
  <Product></Product>
  <Product></Product>
  <Product></Product>
</div>
```

Props demo

```
//in App.js add some attributes/values and child content
<div className="App">
  <h1>Hello World!</h1>
  <Product name="Gear" price="12.98"></Product>
  <Product name="Pulley" price="22.50"></Product>
  <Product name="Sprocket" price="11.49">This is the sprocket description.</Product>
</div>

//in Product.js, modify the JSX content:
function Product(props) {
  return (<p>Each {props.name} costs ${props.price}.</p>);
}
```

Dynamic content demo

```
//in Product.js use props.children to render child content
function(props) {
  return (
    <>
      <p>Each {props.name} costs ${props.price}</p>
      <p>{props.children}</p>
      <hr />
    </>
  )
}
```

State demo part 1

```
//in Product.js convert the Product component to a class
import React, { Component } from 'react';
class Product extends Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (<>
      <p>Each {props.name} costs ${props.price}</p>
      <p>{props.children}</p>
      <hr />
    </>);
  }
}
```

State demo – part 2

```
//add a state property in the constructor and display it in the render method
constructor(props) {
  super(props);
  this.state = { quantity: 0 };
}

render() {
  return (<>
    <p>Each {props.name} costs ${props.price}</p>
    <p>{props.children}</p>
    <div>Quantity Desired: {this.state.quantity}</div>
    <hr />
  </>);
}
```

Event handler context demo

```
//in Product.js add a method
handleClick() {
  console.log(this.props.name);
}

//and a button
<button onClick={this.handleClick}>Click Me</button>

//refresh, click the button and point out the error in the dev tools console
```

Event handler context – fixes

```
//in the constructor, bind the method to "this"
this.handleClick = this.handleClick.bind(this);
//reload and click the button, point out the message in the console

//remove or comment out the bind you just added and use an arrow function for the handler
<button onClick={() => this.handleClick()}
//reload and demo that the message still works

//restore the handler the way it was and change handleClick to an arrow function
handleClick = () => {
  console.log(this.props.name);
};
//reload and demo that message still works
```

Mutating state demo

```
//in Product.js, change the button text to "Buy Now"


//and modify the method to mutate state directly
handleClick = () => {
  this.state.quantity = 1;
}
//refresh and demo that screen is not updated (also show warning in console)
//modify method to correctly mutate state
handleClick = () => {
  this.setState({
    quantity: 1
  });
}
```

Async state mutation demo

```
//in App.js add a unitQuantity attribute to each Product component, with various values
<Product name="Pulley" price="22.50" unitQuantity="5"></Product>

//in Product.js modify the handler to update state asynchronously,
//adding the product's unitQuantity to quantity each time the button is clicked
handleClick = () => {
  this.setState((state, props) => ({
    quantity: state.quantity + parseInt(props.unitQuantity)
  }));
};
//NOTE: you may want to leave out the parseInt() part initially
//to demonstrate that props are strings by default!
```

Form input demo

```
//in Product.js, add another div with an input that displays the quantity
<input type="text" value={this.state.quantity} />
//demo that the value updates when the button is clicked

//update the state's quantity when the text input's value changes
<input type="text" value={this.state.quantity} onChange={this.handleQuantityChange} />

//and
handleQuantityChange = (evt) => {
  this.setState({ quantity: +evt.target.value });
}
```

Lesson 3 – Dynamic Content

General notes

For a more workshop style class, have the students do the following demos with you, that way they have some experience with the topics before they are tasked with doing the exercise.

Perform the demos in the sample application called "demos" located in this chapter's directory. The sample application called "react-fundamentals" already has the demos completed.

Performing each demo provides excellent opportunity to discuss why each step is being taken and what the various options are at each step.

Conditionally showing content demo

```
//in app.js add state to track the visibility of the product components
state = { showProducts: false }

//and add the handler method
toggleProductsHandler = () => {
  this.setState({showProducts: !this.state.showProducts});
}

//then wrap the four <Product> components in a div and add a button above the div
<button onClick={this.toggleProductsHandler}>Toggle Products</button>
{this.state.showProducts && <div>
  //product components here
</div>}
```

Alternative implementation

```
//change the logical expression determining whether to show the products:
{this.state.showProducts ? <div>
  //product components here
</div> : null }
```

Simplify the syntax

```
//still in app.js, modify the render method as follows:
render() {
  let productContent = null;
  if (this.state.showProducts) {
    productContent = ( //paste the entire product component div here );
  }

  return (
    <div className="app">
      //other content
      {productContent}
    </div>
  );
}
```

Collection content demo

```
//use the products array from state to render the product components
//replace the hard-coded components with
productContent = (
  <div>
    {this.state.products.map(product => (
      <Product name={product.name}
            price={product.price}
            unitQuantity={product.unitQuantity}
            >{product.description}</Product>
    ))}
  </div>
);
//demo that it works, but we get a warning about each element needing a unique key
```

Key demo

```
//add a unique key for each element, use the product id property
productContent = (
  <div>
    {this.state.products.map(product => (
      <Product id={product.id}
            name={product.name}
            price={product.price}
            unitQuantity={product.unitQuantity}
            >{product.description}</Product>
    ))}
  </div>
);
```

Updating collection state demo

```
//want a "Remove" button in the Product component that will remove that product
//in Product.js add a button that will execute a props method when clicked
<button onClick={this.props.removeClicked}>Remove</button>

//in app.js add an event handler – will use the index of the element to be removed
removeProductHandler = (idx) => {
  const prods = this.state.products;
  prods.splice(idx, 1);              //this is not best practice – we will fix shortly
  this.setState({products: prods });
}

//modify the map() function to receive the index of the current element
{this.state.products.map((product, idx) =>

//and add the handler as a prop to the Product component (inside the map() function)
removeClicked={() => this.removeProductHandler(idx)}
```

Updating state immutably

```
//need to create a copy of the current state array before making changes
//modify the remove handler:
removeProductHandler = (idx) => {
  const prods = this.state.products.slice();  //create a new array
  prods.splice(idx, 1);
  this.setState({products: prods });
}
```

Updating state of object in array – setup Product

```
//each product needs an option for a custom inscription to be engraved on it
//in Product.js, after the name and cost add a display for the inscription:
<div>Inscription: {this.props.inscription}</div>

//after the Quantity Desired, add a label and text box:
<div>Custom Inscription: <input type="text"
    value={this.props.inscription} onChange={this.props.inscriptionChanged} />
</div>
```

Updating state of object in array demo

```
//in app.js create a handler for the inscriptionChanged event
inscriptionChangedHandler = (evt, id) => {
}

//in app.js pass the inscription as a prop to each Product and hook up the handler
<Product key={product.id}
  . . .
  inscription={product.inscription}
  inscriptionChanged={(evt) => this.inscriptionChangedHandler(evt, product.id)}
>{product.description}</Product>
```

Updating state – continued

```
//finish the inscriptionChanged handler
const productIndex = this.state.products.findIndex(p => p.id === id);

//create a copy of the product to mutate
const product = { ...this.state.products[productIndex] };
product.inscription = evt.target.value;

//create a copy of the array to mutate
const products = [ ...this.state.products ];
products[productIndex] = product;

this.setState({ products: products });
```

Lesson 4 – Styling Content

General notes

For a more workshop style class, have the students do the following demos with you, that way they have some experience with the topics before they are tasked with doing the exercise.

Perform the demos in the sample application called "demos" located in this chapter's directory. The sample application called "react-fundamentals" already has the demos completed.

Performing each demo provides excellent opportunity to discuss why each step is being taken and what the various options are at each step.

Inline style demo

```
//in Product.js, in the render() method add:
const cardStyle = {
  border: '1px solid #CCCCEE',
  borderRadius: '4px',
  padding: '1em',
  margin: '2em 0',
  width: '350px',
  backgroundColor: '#EEEEFF',
  boxShadow: '3px 3px 5px #DDD'
};

//and modify the wrapper div in the JSX
<div style={cardStyle}>
```

Dynamic style demo

```
//in app.js, in the render method add:
const showButtonStyle = {
  backgroundColor: 'green',
  color: 'white',
  padding: '0.5em 0.8em',
  border: '1px solid #888',
  borderRadius: '3px' };

//in the conditional, change the style object
if (this.state.showProducts) {
  showButtonStyle.backgroundColor = 'red';

//and in the returned JSX modify the button
<button style={showButtonStyle} onClick=. . .
```

Classes demo

```
//in app.css add:
.strong { font-weight: bold; }

//in Product.js, in the render() method add:
let qtyStyle = '';
if (this.state.quantity !== 0) qtyStyle = 'strong';

//and in the return JSX, modify the quantity div:
. . .
<div className={qtyStyle}>Quantity Desired: {this.state.quantity}</div>
. . .
```

Classes demo – continued

```
//in app.css add:
.danger { color: #FF3333; }

//in Product.js change the qtyStyle to potentially be multiple classes
const qtyStyles = [];
if (this.state.quantity !== 0) qtyStyles.push('strong');
if (this.state.quantity < 0) qtyStyles.push('danger');

//and modify the quantity div:
<div className={qtyStyles.join(' ')}>Quantity Desired: {this.state.quantity}</div>

//ask the students: how would you implement media queries? Hover effects? :before? :after?
```

Install Radium

```
//suppose we want a hover state for the Show/Hide button
//at a command prompt in the application directory
npm install Radium

//in app.js import Radium and wrap the exported component in a Radium HOC
import Radium from 'radium';



export default Radium(App);
```

Radium pseudo class demo

```
//in app.js modify the showButtonStyle object, adding a property:
':hover': {
  backgroundColor: 'LimeGreen'
}

//reload and demo
//looks good before showing products, but not so much after showing products

//add another line of code inside the conditional for this.state.showProducts:
showButtonStyle[':hover'].backgroundColor = 'OrangeRed';

//reload and demo
```

Radium media query demo

```
//in Product.js, import Radium and wrap the export in the HOC
//modify the cardStyle object so that:
width: '100%',

//and add the media query style property:
'@media (min-width: 27em)': {
  width: '350px',
  display: 'inline-block',
  marginRight: '1em'
}

//reload and demo the error
```

Radium – adding StyleRoot

```
//in app.js import StyleRoot along with Radium
import Radium, { StyleRoot } from 'radium';

//and wrap the returned JSX from render() in the StyleRoot element
return (
  <StyleRoot>
    . . .
  </StyleRoot>
);

//reload and demo the working media query
```

CSS Modules demo

```
//for a more impactful demo, strip away Radium and all cardStyle from the Product.js file
//and demo the lack of formatting before moving forward
//create a file Product.module.css, copy in Product card styles (and edit to CSS syntax)
.productCard {
  border: 1px solid #CCCCEE;
  . . . etc.
}
@media screen and (min-width: 27em) {
  .productCard {
    width: 350px;
    display: inline-block;
    margin-right: 1em;
  }
}
```

CSS Modules demo – continued

```
//in Product.js import the styles
import styles from './Product.module.css';

//comment out all the properties for the cardStyle object in the render() method

//add the productCard class to the wrapper div returned from the JSX
return(
  <div className={styles.productCard}>
    . . .
  </div>
);

//reload and display – point out modified class name in browser dev tools
```

CSS Modules demo – additional example

```
//in app.js strip out all traces of Radium (including the button hover state)
//rename the CSS file and change the style import to:
import styles from './App.module.css';

//modify the JSX returned to use styles.App
<div className={styles.App}>

//reload and demo that the formatting of quantity (when ordered) no longer works
//(App.css styles are no longer global), in Product.js add:
import appStyles from './App.module.css';

//and change the conditional styles to:
qtyStyles.push(appStyles.strong);
qtyStyles.push(appStyles.danger);
```

CSS Modules – restoring hover state

```
//in App.js move showButtonStyle rules into App.module.css and re-write in CSS syntax
//using the selector: .App > button

//remove showButtonStyle from App.js and from returned JSX
//in the render() method add a buttonClass variable with a default value of 'on'
let buttonClass = styles.off;
//and modify its value inside the conditional
if (this.state.showProducts) {
  buttonClass = styles.on;
//use the button class in the returned JSX
<button className={ buttonClass } . . .

//modify App.module.css to set the button colors based on the class names
```

Lesson 5 - Debugging

General notes

For a more workshop style class, have the students do the following demos with you, that way they have some experience with the topics before they are tasked with doing the exercise.

Perform the demos in the sample application called "demos" located in this chapter's directory. The sample application called "react-fundamentals" already has the demos completed.

Performing each demo provides excellent opportunity to discuss why each step is being taken and what the various options are at each step.

Error message demo

```
//in app.js, in inscriptionChangedHandler() remove "target" from "evt.target.value"
//run the application, change an inscription and demo the error messages
//talk about how you would figure out what went wrong and how to figure out
//what the code should be (e.g. check online docs for JS event parameter)
```

Devtools demo

```
//after setting up the error in the previous demo, open up the browser devtools
//and put a breakpoint on the line of code where the error is indicated
//re-run the experiment and use the locals inspector to view the available
//properties of the evt parameter, discovering that the line of code
//should be evt.target.value – fix the error

//set up another error in the same method by changing the findIndex() method
//to look for: p => p.productid === id

//demo no errors, but improper behavior
//use debugger in browser tools to figure out the problem
```

React devtools demo

```
//search the internet for react developer tools
//install the plugin to your browser (may need to restart browser or open a new tab)
//load the site and demo the React devtools – how they let you inspect components
```

Error boundary demo

```
//in Product.js, in the render() method simulate a random error:
if (Math.random() < 0.3) throw new Error('Houston, we have a problem');

//demo the error and how it displays onscreen
//that is nice at dev time, but horrible behavior once deployed live
```

Error boundary component

```
//add a directory called "ErrorBoundary" and in it a file called "ErrorBoundary.js"
import React, { Component } from 'react';

class ErrorBoundary extends Component {
  state = {
    hasError: false,
    errorMessage: ''
  }

}

export default ErrorBoundary;
```

Component continued

```
//add (inside the component)
componentDidCatch = (error, info) => {
  this.setState({
    hasError: true,
    errorMessage: error.message
}

render() {
  if (this.state.hasError) {
    return (<h1>{this.state.errorMessage}</h1>)
  } else {
    return this.props.children;
  }
}
```

Using the Error Boundary

```
//in App.js import the new ErrorBoundary component
import ErrorBoundary from './ErrorBoundary/ErrorBoundary';

//in the render method, when map()-ing the products, wrap the Product component
//with an ErrorBoundary (don't forget to move the key attr to the ErrorBoundary)
{this.state.products.map((product,idx) =>
  (<ErrorBoundary key={product.id}><Product . . .


    >{product.description}</Product></ErrorBoundary>))}

//demo that browser still shows ugly error details (they will only show in dev) –
//but they can be closed to see nice ErrorBoundary content
```

Lesson 6 - Components

General notes

For a more workshop style class, have the students do the following demos with you, that way they have some experience with the topics before they are tasked with doing the exercise.

Perform the demos in the sample application called "demos" located in this chapter's directory. The sample application called "react-fundamentals" already has the demos completed.

Performing each demo provides excellent opportunity to discuss why each step is being taken and what the various options are at each step.

Creating components/restructuring app demo

```
//review the application. Behavior is the same as earlier lessons, but the
//Products component has been refactored to be a stateless functional component

//App component lacks focus. Being the entry point for the application, it should be
//lean, and just delegate to other component(s)

//inside "src" directory create two new directories: "components" and "containers"

//move all three "App" files into containers and fixup the import of App in index.js

//in the new components directory create a Products directory
//move Product.js and Product.module.css into it
```

Creating components demo

```
//create a functional Products component in the Products directory,
//receive props and return a div and the results of looping over props.products,
//mapping each product to an instance of the Product component,
//pass removeClicked, inscriptionChanged and buyClicked to identical methods
//that this new component will receive as props

//refactor the App.js component to use the new Products component:
//change the JSX in the productContent variable to be an instance of the Products
//component, passing the products array and all three methods as props

//demo the app – the behavior should be the same, but the app is more cleanly organized
```

Component creation lifecycle demo

```
//in App.js add a constructor to the component and move the state initialization there:
constructor(props) {
  super(props);
  console.log('(App.js) - in constructor');
  this.state = {
    showProducts: false,
    products: [
      { id: 1 . . . },
      { id: 2 . . . },
      { id: 3 . . . }
    ]
  };
}
//reload and demo message in developer console
```

Creation lifecycle continued…

```
//add:
static getDerivedStateFromProps(props, state) {
  console.log('(App.js) in getDerivedStateFromProps()');
  return null;
}
componentDidMount() {
  console.log('(App.js) in componentDidMount()');
}
//add a log message to the render() method
//if desired, add a log message to Products and Product component functions

//demo – click button a few times and notice multiple messages
```

Component update lifecycle demo

```
//to demo update lifecycle events, convert Products.js and Product.js into class components
//import { Component } from 'react';
//replace the function with a class that extends Component,
//give it a render method that returns the existing JSX
//change all props to this.props and export the class, not the function
//(perhaps convert one with the students and then have them do the other on their own)

//in Products.js start adding lifecycle events:
static getDerivedStateFromProps(props, state) {
  console.log('(Persons.js) getDerivedStateFromProps');
  return state;
}
```

Update lifecycle continued

```
//still in Products.js:
shouldComponentUpdate(nextProps, nextState) {
  console.log('(Persons.js) shouldComponentUpdate');
  return true;
}
getSnapshotBeforeUpdate(prevProps, prevState) {
  console.log('(Persons.js) getSnapshotBeforeUpdate');
  return { message: 'hello world from snapshot' };
}
componentDidUpdate(prevProps, prevState, snapshot) {
  console.log('(Persons.js) componentDidUpdate', snapshot);
}
//load the site, click the button to show products – creation lifecycle executes
//clear the console messages and type in the Custom Inscription for a component to demo
```

Update lifecycle – final demo

```
//go back to the shouldComponentUpdate event for Products.js and return false
//reload, click the button to show products and then type in the Custom Inscription
//text box for any product
//point out lifecycle events are running (messages in console) but no UI changes!

//reset the event to return true
```

Performance enhancement demo – setup

```
//in App.js add a new state property:
showTitle: true,
//and a new method:
hideTitleHandler = () => { this.setState({ showTitle: false }); }

//in the render() method add:
let titleContent = null;
if (this.state.showTitle) { titleContent = (<h1>React Fundamentals</h1>); }

//and in the returned JSX, replace the <h1>React Fundamentals</h1> with:
{ titleContent }
<button onClick={this.hideTitleHandler}>Hide Title</button>

//reload, show the products, clear the console and then hide the title – Products are re-rendered!!
```

Performance enhancement demo – implemented

```
//in Products.js modify shouldComponentUpdate to perhaps return false:
if (nextProps.products !== this.props.products) {
  return true;
} else {
  return false;
}
//reload and demo that the re-rendering of Products and the three Product do not happen
//point out that we are testing equality of reference and this only works because we are
//updating state immutably (e.g. refer to inscriptionChangedHandler in App.js)

//ALSO: click Hide Title multiple times – multiple re-renderings of App happen!
//modify hideTitleHandler (in App.js):
if (this.state.showTitle) this.setState({ showTitle: false });
//reload and demo re-rendering of App only happens on the first click of Hide Title
```

Pure components demo

```
//examine Products.js and shouldComponentUpdate – we're only testing this.props.products
//look at returned JSX from render() – we also depend upon other props
//(namely: productRemoved, inscriptionChanged and buyClicked – handlers)
//handlers are not likely to change, but they could and currently those changes are ignored!

//modify Products.js to be a Pure Component:
import React, { PureComponent } from 'react';

class Products extends PureComponent {

  //comment out shouldComponentUpdate()
}

//reload and demo that rendering doesn't happen when hiding the title
```

Adjacent elements demo

```
//in Products.js, in the render() method, add an adjacent element to the existing content:
return (
  <div>Hello World</div>
  <div>
    {this.props.products.map((product, idx) =>
      . . .
    ))}
  </div>
);

//demo the error in the browser
```

Adjacent element fix #1 - array

```
//modify the returned JSX in the render method of Products.js to return an array:
return [
  <div key="thing1">Hello world!</div>,   //don't forget the comma here!
  <div key="thing2">
    {this.props.products.map((product, idx) =>
      . . .
    ))}
  </div>
];

//perhaps demo it as an array without the keys, view the error message and then add the keys
//remove the array to prepare for next demo
```

Adjacent element fix #2 - Fragments

```
//React 16.2 provides a non-rendering wrapper element – modify Products.js:
import React, { PureComponent, Fragment } from 'react';
//and wrap the adjacent elements in:
<Fragment>
  <div>Hello World!</div>
  <div> .  . . </div>
</Fragment>
//demo that all works well, inspect the DOM and see no wrapping element!
//Variation: empty tags as an alias for Fragment:
<>
  <div>Hello World!</div>
  <div> . . .</div>
</>
//demo that all works well (can remove Fragment import, if desired)
```

Adjacent elements fix #3 – HOC

```
//a homemade Higher Order Component will also do the trick
//create a directory called "hoc" and in it a file called wrapper.js:
const wrapper = props => props.children;
export default wrapper;

//in Products.js import the new component and use it instead of <Fragment></Fragment>
import Wrapper from '../../hoc/wrapper';

return (<Wrapper>
  <div>Hello World!</div>
  <div>. . . </div>
</Wrapper>);
```

Higher-Order Component demo

```
//create another file in the "hoc" directory – withGatewayToken.js
import React, { Component } from 'react';

const withGatewayToken = (Wrapped) => {
  class HOC extends Component {
    render() {
      return (<Wrapped {...this.props} gatewayToken = {456456456} />);
    }
  }
  return HOC
};

export default withGatewayToken;
```

Higher-Order Component demo - continued

```
//in Products.js import the new HOC
import WithGatewayToken from '../../hoc/withGatewayToken';

//modify the JSX output
return (<Wrapper>
  <div>Hello world! – token is {this.props.gatewayToken}</div>
  . . .
</Wrapper>);

//and modify the export:
export default WithGatewayToken(Products);
```

Validating Prop Types demo

```
//install prop-types
npm install prop-types

//import into Product.js
import PropTypes from 'prop-types';
//add the propTypes before exporting:
Product.propTypes = {
  name: PropTypes.string,
  price: PropTypes.number,
  inscription: PropTypes.string,
  quantity: PropTypes.number,
  ...
};
//execute – should be OK – in App.js change a price to be a string, execute – warning!
```

refs demo

```
//e.g. to set the focus to the custom inscription when "Buy" button is clicked:
//in Product.js add a ref to the inscription input:
<input type="text" ref="inscriptionBox" value={this.props.inscription} . . . />

//modify the Buy Now button onClick:
<button onClick={() => {
  this.props.buyClicked();
  this.refs.inscriptionBox.focus();
}}>Buy Now</button>
```

Context API demo

```
//want to control discounts in App.js but apply them in Product.js.
//create a directory "context" and a file in it "discountContext.js":
import React from 'react';

const discountContext = React.createContext({
  enabled: false,
  applyDiscount: () => {}
});

export default discountContext;
```

Context API demo – part 2

```
//in app.js:
import DiscountContext from '../context/discountContext';
//add to state:
discountEnabled: false
//add a method:
applyDiscount = (amount) => this.state.discountEnabled ? amount * 0.8 : amount;

//and in the returned JSX wrap the productContent with the provider:
<DiscountContext.Provider value={{
    enabled: this.state.discountEnabled,
    applyDiscount: this.applyDiscount
  }}>
  {productContent}
</DiscountContext.Provider>
```

Context API demo – part 3

```
//still in app.js add an event handler, a method and a button to toggle discount
toggleDiscountHandler = () => {
  this.setState((prevState, props) => {
    return { discountEnabled: !prevState.discountEnabled };
  });
};

//and inside the context provider, before {productContent}:
<button onClick={this.toggleDiscountHandler}>Toggle Discount</button>
```

Context API demo – part 4

```
//in Product.js
import DiscountContext from '../../context/discountContext';

//and change the render() return:
return (<DiscountContext.Consumer>
  {(context) => (<div className={styles.productCard}>
    <div>Each {this.props.name} costs ${context.applyDiscount(this.props.price)}</div>
    ...
  </div>)}
</DiscountContext.Consumer>)

//Product.js now behaves like a functional component which cannot use ref directly,
//so comment out (or delete) the ref portion of the component – demo and apply discount
```

Alternative Access to Context API demo

```
//in Product.js undo to remove the <DiscountContext.Consumer>
//and the function receiving context (but leave the import)

//add a static property
static contextType = DiscountContext;

//and apply the discount to the price output:
<div>Each {this.props.name} costs ${this.context.applyDiscount(this.props.price)}</div>

//reload and demo
```

Lesson 7 – Web Server Interactions

General notes

For a more workshop style class, have the students do the following demos with you, that way they have some experience with the topics before they are tasked with doing the exercise.

Perform the demos in the sample application called "demos" located in this chapter's directory. The sample application called "react-fundamentals" already has the demos completed.

Performing each demo provides excellent opportunity to discuss why each step is being taken and what the various options are at each step.

Axios demo

```
//want to dynamically load data – use http://www.kazoopromotions.com/api/boardmembers
//install Axios to the application
npm install axios

//in Board.js add:
import axios from 'axios';

componentDidMount() {
  axios.get('https://www.kazoopromotions.com/api/boardmembers')
    .then(resp => {
      console.log(resp);
    });
}
//load and investigate the response object in the console
```

Axios demo – continued

```
//still in Board.js put the received data into state – add a state property
state = {
  members: []
}

//in the .then() callback set state
this.setState({ members: resp.data });

//prepare MemberTile.js to receive data – modify its JSX
<h1>{props.firstName} {props.lastName}</h1>
<div className={styles.jobTitle}>{props.title}</div>
```

Axios demo – part 3

```
//back in Board.js create the member content in the render() method:
const members = this.state.members.map(member => {
  return <MemberTile firstName={member.FirstName} lastName={member.LastName}
    title={member.Title} />
});

//and in the returned JSX
<section className={styles.memberList}>
  {members}
</section>

//demo – get warning about key, so modify the map() method
  return <MemberTile . . . key={member.Id} />
```

Interactions demo

```
//goal: click a member to view details – so in MemberTile.js call a function when clicked
<article className={styles.memberTile} onClick={props.clicked}>

//in Board.js add to state
selectedMemberId: null
//create a method
memberSelectedHandler = (id) => {
  this.setState({ selectedMemberId: id });
}
//give the method to each MemberTile (in the map() method inside render() )
return <MemberTile . . . clicked={() => this.memberSelectedHandler(member.Id)} />
//and pass the ID to the MemberDetail component
<MemberDetail id={this.state.selectedMemberId} />
```

Interactions demo continued

```
//in MemberDetail.js utilize the id now being passed:
//in the render() method, only change the content if props.id exists
if (this.props.id) {
  content = ( . . .);
}

//now, setup to fetch the full member data
import axios from 'axios';

//and add some state
state = {
  loadedMember: null
};
```

Interactions demo – part 3

```
// and add the lifecycle event:
componentDidUpdate() {
  if (this.props.id) {
    axios.get('https://www.kazoopromotions.com/api/boardmembers/' + this.props.id)
      .then(resp => {
        this.setState({ loadedMember: resp.data });
      });
  }
}
//finally render firstName, lastName, title and bio from this.state.loadedMember
//in the JSX content

//demo – Error! b/c props.id is set before we have data for loadedMember
//and we're trying to render loadedMember right away
```

Interactions demo – fixing the display error

```
//in the render() method modify the existing conditional and add another:
if (this.props.id) {
  post = <p className={styles.instructions}>Loading the Board Member!</p>
}
if (this.state.loadedMember) {
  post = ( . . . );
}

//note that bio is HTML content so use:
<div dangerouslySetInnerHTML={ {__html: this.state.loadedMember.bio} }></div>

//Now, open the network tab of the developer tools and point out
//the continuous loop of requests – our setState() starts the update lifecycle again!
```

Interactions demo – fixing the infinite loop

```
//we only need to fetch data if we have no loadedMember or if it is the wrong member
//modify the conditional in componentDidUpdate():
if (this.props.id &&
  (!this.state.loadedMember || this.state.loadedMember.id !== this.props.id))

//demo and check the network tab – no infinite loop!
```

POST demo

```
//modify NewMember.js to import axios and have a method that will send data
import axios from 'axios';
postDataHandler = () => {
  const data = {
    firstName: this.state.firstName,
    lastName: this.state.lastName,
    title: this.state.title,
    bio: this.state.bio
  };
  axios.post('https://www.kazoopromotions.com/api/boardmembers', data)
    .then(resp => {
      console.log(resp);
    });
};
```

POST demo – continued

```
//connect the new method to the click event of the button
<button onClick={this.postDataHandler}>Add Member</button>

//load and test with some dummy data – examine the response in the console
```

DELETE demo – optional

```
// in MemberDetail.js add
deleteMemberHandler = () => {
  axios.delete('https://www.kazoopromotions.com/api/boardmembers/' + this.props.id)
    .then(resp => {
      console.log(resp);
    });
}

//and connect it to the delete button
<button className="Delete" onClick={this.deleteMemberHandler}>Delete</button>

//demo by selecting a member and then clicking delete
//view the response in the console
```

Errors demo

```
//create an error by changing the URL where the request is sent – in Board.js
//modify the URL for axios in componentDidMount and demo the error in the console

//handle the error with a catch()
componentDidMount() {
  axios.get(. . . )
    .then( . . . )
    .catch(err => {
      console.log(err);
    });

//demo – console no longer reports "unhandled error" and note error details
```

Toaster demo

```
//install react-toastify
npm install react-toastify
//in App.js:
import { ToastContainer } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
//in the returned JSX:
<div className={styles.app}>
  <Board />
  <ToastContainer />
</div>
```

Toaster demo – continued

```
//in Board.js import the toast object
import { toast } from 'react-toastify';

//in the catch() handler in componentDidMount:
toast.error(err.ToString());

//Demo the error message

//Finally, fix the error so data loads correctly
```

Request Interceptor demo

```
// set up a request interceptor to examine the request configuration
// and create a global error handler – in index.js - before ReactDOM.render()
import axios from 'axios';

axios.interceptors.request.use(req => {
  console.log('(request interceptor', req);
  return req;
}, err => {
  console.log('(request handler)', err);
  return Promise.reject(err);
});
//demo and show the request config object in the console
//modify the URL in Board.js to get an error
//NOTE: the handler does not execute (b/c request actually worked)
```

Response Interceptor demo

```
//back in index.js:
axios.interceptors.response.use(resp => {
  console.log('(response interceptor)', resp);
  return resp;
}, err => {
  console.log('(response handler)', err);
  return Promise.reject(err);
});

//demo the error messages – this time the Response handler caught the error

//finally restore the URL in Board.js for proper functionality
```

Axios Configuration demo

```
//in index.js add
axios.defaults.baseURL = 'https://www.kazoopromotions.com/api';
axios.defaults.headers.common['Authorization'] = 'MY AUTH TOKEN';

//demo – still works b/c baseURL is only added to relative URLs
//in Board.js modify request to:
axios.get('/boardmembers')

//in NewMember.js modify the one URL to be relative

//in MemberDetail.js modify both URLs to be relative

//demo all functionality still works and custom header value is present
```

Custom Instance demo

```
//in the src directory create a new file "boardApi.js" and enter:
import axios from 'axios';

const instance = axios.create({
  baseURL: 'https://www.kazoopromotions.com/api'
});
instance.defaults.headers.common['Authorization'] = 'INSTANCE TOKEN';

export default instance;

//in MemberDetail.js change axios import to:
import axios from '../../boardApi';

//demo – app still works but interceptors are not called for MemberDetail GET and DELETE
```

Lesson 8 – Routing

General notes

For a more workshop style class, have the students do the following demos with you, that way they have some experience with the topics before they are tasked with doing the exercise.

Perform the demos in the sample application called "demos" located in this chapter's directory. The sample application called "react-fundamentals" already has the demos completed.

Performing each demo provides excellent opportunity to discuss why each step is being taken and what the various options are at each step.

Prep for Routing demo

```
//add react-router to the application
npm install react-router react-router-dom

//in app.js import BrowserRouter
import { BrowserRouter } from 'react-router-dom';

//and wrap the <header> and <Board /> content in BrowserRouter
<BrowserRouter>
  <header> . . . </header>
  <Board />
</BrowserRouter>
```

Routing Demo – static content and Home

```
//still in App.js import Route as well
import { BrowserRouter, Route } from 'react-router-dom';

//replace <Board /> in the JSX with (illustrate that we can return any JSX)
<Route path="/" render={() => <h1>Home</h1> } />

//demo that this content shows up for all menu items
// - router tests path for "starts with" matches rather than "equals" matches with the route
//change the Route to:
<Route path="/" exact render={() => <h1>Home</h1> } />

//now it uses "equals"
//make a copy of that Route component with different content and demo they both show up
```

Rendering Components demo

```
//still in App.js import the Home component
import Home from './Home';

//then get rid of the second Route and modify the first:
<Route path="/" exact component={Home} />
<Route path="/" component={Board} />

//reload and demo the links

//perhaps have students wire up "/about-us" to the AboutUs component
//                          and "contact-us" to the ContactUs component
```

Navigating demo

```
//currently links are posting back (which reloads the page, loses all state data!)
//in App.js import Link
import { BrowserRouter, Route, Link } from 'react-router-dom';

//and modify the <a> to:
<Link to="/">Home</Link>
...etc.

//perhaps demo one of them with the object syntax:
<Link to={{pathname: '/about-us', hash: '#ceo', search: '?q=elvis'}}>About Us</Link>

//reload and demo – no postbacks now!
```

Routing-related props demo

```
//in Board.js, in the componentDidMount() method log the component's props:
componentDidMount() {
  console.log(this.props);
  axios.get . . .
}

//reload the Board Members page and examine the props in the console
//do the same for the AboutUs component and point out the hash and query params
```

Router props in child components demo

```
//suppose we need router props in a child component (not directly loaded by router)
//in MemberTile.js add code inside the function:
const memberTile = (props) => {
  console.log('(tile)', props);
  return ( . . . );
}

//load and demo this.props has no router props – b/c it was not directly loaded by router
//back in MemberTile.js add:
import { withRouter } from 'react-router-dom';

export default withRouter(memberTile);

//demo that router props are now there!
```

Styling the Active Route demo

```
//in the browser, inspect the navigation links – no special class for the "current" link
//in app.js import NavLink instead of Link and change all <Link> to <NavLink>

//refresh browser and inspect classes on "current" link
//go to index.css and add:
a.active {
  background-color: #FFC;
}

//view the page – almost perfect:
//inspect the "Home" link – it is "active" even away from the home page
//in app.js add the "exact" attribute to the Home NavLink and refresh the page
```

Another need for exact demo

```
//want to show the NewMember content on its own screen at the URL "board/add".
//Remove it and its import from Board.js

//in App.js import NewMember and add a Route for it:
import NewMember from './Board/NewMember';

<Route path="/board/add" component={NewMember} />

//in Board.js add a link to get there:
import { Link } from 'react-router-dom';

return ( <div>
    <p><Link to="/board/add">Add New</Link></p>
    <section> . . .
```

Another need for exact demo – continued

```
//try the button – the form appears, but the tiles still display!
// b/c the "/board" route still matches – routing uses "starts with" as default matcher

//go to App.js and add "exact" to the "/board" route
<Route path="/board" exact component={Board} />

//now it works properly
//NOTE: probably do NOT want to add "exact" to the NavLink for /board
// b/c adding is still under the general heading of Board of Directors
// and it should be highlighted
```

Route parameters demo – setup

```
//Board.js should not show list and detail on the same screen,
//should be different screens, with urls like "/board", "/board/103"

//in Board.js remove the <section> for MemberDetail (and its import)
//remove the selectedMemberId from state and from the memberSelectedHandler

//in App.js import MemberDetail and set up the dynamic route
import MemberDetail from './Board/MemberDetail';

<Route path="/board/:id" component={MemberDetail} />
//be sure to put this route after "/board/add" so as not to interfere
```

Route Parameters demo – Link

```
//first solution to redirect – in Board.js when using map() to build the members content
//wrap the MemberTile component in a Link (move the key attribute to Link)
<Link to={'/board/' + member.id} key={member.id}>
  <MemberTile . . .  />
</Link>

//demo functionality – now links, but MemberDetail needs to load its data differently now
```

Route Parameters demo – using parms

```
//MemberDetails now needs to load its data when mounting instead of updating
//change the function componentDidUpdate() into componentDidMount()

//in addition, it needs to get the Id from the URL instead of from props
//add as the first line inside componentDidMount()
console.log(this.props);

//note that props.match has "params" with an "id"
//get rid of the console.log, change the conditional to:
if (this.props.match.params.id)

//and use that same value in the URL for axios
```

Route Parameters demo – problem

```
//navigate to the "Add New" board member screen
//notice that the MemberDetails component shows up there as well!
// b/c the route still matches - "board/add" matches "board/:id" – "add" is a valid "id"

//solution: use Switch to load only one route – in App.js:
import { BrowserRouter, Route, NavLink, Switch } from 'react-router-dom';

<Switch>
  <Route path="/board/add" component={NewMember} />
  <Route path="/board/:id" component={NewMember} />
</Switch>

//interesting demo – switch the order of these two routes and see that "Add New"
//loads the wrong component
```

Route Parameters demo – navigate via code

```
//after clicking "delete" on the details, should redirect back to list
//refresh the details page and examine this.props in the console
//the "history" object has a push() method

//in MemberDetail.js, in deleteMemberHandler change the URL and in the then() callback:
axios.delete('/boardmembers/' + this.props.match.params.id)
  .then(resp => {
    console.log(resp);
    this.props.history.push('/board');
  })
  . . .
```

Nested Route demo

```
//in board.js
import { Route } from 'react-router-dom';
import MemberDetail from './MemberDetail';

//remove the detail Route from app.js and put it in Board.js JSX
//after the section containing the members list
<Route path="/board/:id" component={MemberDetail} />

//demo – the content disappears when clicking a tile. Why? main board route has "exact"
//remove "exact" in app.js from the board route

//content now shows, but does not update when clicking other tiles
//also, click on "Add New" – now ALL content shows up!
```

Nested Route demo – fixing it

```
//fix "Add New" problem first – both board and board/add routes match
//in app.js move the board route into the Switch – and put it second

//for the non-update problem, URL is changing, but React is re-using the same component
//instance, for performance purposes, therefore componentDidMount() is not firing again
//in MemberDetails.js move the code from componentDidMount() into a new function
//called loadMember() and invoke it from componentDidMount()
//add componentDidUpate() and invoke the function there

//demo – we have the infinite loop of re-sending AJAX calls once again
//modify the conditional in loadMember()
if (!this.state.loadedMember || this.props.match.params.id != this.state.loadedMember.id) }
//be sure to use "!=" instead of "!==" b/c one is a string and one is a number
```

Lesson 9 – Forms

General notes

For a more workshop style class, have the students do the following demos with you, that way they have some experience with the topics before they are tasked with doing the exercise.

Perform the demos in the sample application called "demos" located in this chapter's directory. The sample application called "react-fundamentals" already has the demos completed.

Performing each demo provides excellent opportunity to discuss why each step is being taken and what the various options are at each step.

Custom Input Component demo

```
//in "components" create a folder "Forms" and in it files "Input.js" and "Input.module.css"
//in Input.js create a functional component and export it:
import React from 'react';
import styles from './Input.module.css';

const input = (props) => {
  let elem = null;

  return ( <div className={styles.input}>
      <label>{props.label}</label>
      {elem}
    </div> );
};
export default input;
```

Custom Input Component – continued

```
//between the variable declaration and the return statement:
switch ( props.kind ) {
  case ('textarea'):
    elem = <textarea {...props} />
    break;
  default:
    elem = <input {...props} />
}
```

Custom Input Component – styling

```
//in Input.module.css add:
.input {
  width: 100%; padding: 0.8em; box-sizing: border-box;
}
.input label {
  font-weight: bold; display: block; margin-bottom: 0.4em;
}
.input input, .input textarea {
  outline: none; border: 1px solid #ccc; font: inherit; padding: 0.4em 0.8em;
  display: block; width: 100%; box-sizing: border-box;
}
.input input:focus, .input textarea:focus {
  outline: none; background-color: #eee;
}
```

Custom Input Component – using

```
//delete or comment out label, input, textarea and select stylings in NewMember.module.css,
//import custom component into NewMember.js and use it:
import Input from '../../components/Forms/Input';

//comment out or delete the label/input pairs and replace with
<Input label="First Name" />
<Input label="Last Name" />
<Input label="Title" />
<Input label="Email" type="email" />
<Input label="Hire Date" type="date" />
<Input label="Bio" kind="textarea" placeholder="summarize professional experience" />

//demo
```

Form Configuration demo – setup

```
//in "src" create a new file "formBuilder.js"
const formBuilder = {
  configInput(kind, type, label, placeholder, value='') {
    const settings = { kind, label, value, attrs: {} };

    if (typeof type !== 'undefined') { settings.attrs.type = type; }

    if (typeof placeholder !== 'undefined') { settings.attrs.placeholder = placeholder; }
    else { settings.attrs.placeholder = 'enter ' + label.toLowerCase(); }

    return settings;
  }
};
export default formBuilder;
```

Form Configuration demo – define form elements

```
//in NewMember.js
import formBuilder from '../../formBuilder';

//change state to be:
state = {
  memberForm: {
    firstName: formBuilder.configInput('input', 'text', 'First Name'),
    lastName: formBuilder.configInput('input', 'text', 'Last Name'),
    title: formBuilder.configInput('input', 'text', 'Title'),
    email: formBuilder.configInput('input', 'email', 'Email'),
    hireDate: formBuilder.configInput('input', 'date', 'Hire Date'),
    bio: formBuilder.configInput('textarea', '', 'Bio', 'summarize professional experience')
  }
}
```

Form Configuration demo – build form

```
//in formBuilder.js
import React from 'react';
import Input from './components/Forms/Input';

//add a method to the formBuilder object:
buildForm(formConfig) {
  const elements = [];
  for (let key in formConfig) {
    elements.push({ id: key, config: formConfig[key] });
  }

  //more to come here
}
```

Form Configuration demo – build form, part 2

```
//continuing the function:
const content = elements.map(elem => (
  <Input key={elem.id}
    label={elem.config.label}
    kind={elem.config.kind}
    value={elem.config.value}
    attrs = {elem.config.attrs}
      />
));
return content;

//in Input.js in the switch statement, change both {...props} to {...props.attrs}
elem = <input {...props.attrs} />
```

Form Configuration demo – using the form content

```
//in NewMember.js, inside the render() method, before the return:
const formContent = formBuilder.buildForm(this.state.memberForm);

//in the return statement, replace the <Input> elements with:
{formContent}

//demo – the form displays, but is readonly (and has warnings about that)
```

Handle changes demo

```
//in NewMember.js create a method to receive changes
inputChangedHandler = (evt, id) => {
  const updatedForm = { ...this.state.memberForm };
  const updatedElement = { ...updatedForm[id]};
  updatedElement.value = evt.target.value;
  updatedForm[id] = updatedElement;
  this.setState({ memberForm: updatedForm });
};

//and in the render method, pass the handler to the buildForm() method:
const formContent = formBuilder.buildForm(this.state.memberForm, this.inputChangedHandler);
```

Handle Changes demo – part 2

```
//in formBuilder.js in the buildForm() method add a parameter
buildForm(settings, handler) {

//and use the handler when rendering the <Input> element in the map() method
<Input key={elem.id} . . .  changed={(event) => handler(event, elem.id)}

//finally, in Input.js in both branches of the switch add an onChange attribute:
onChange={props.changed}
```

Form Submit demo

```
//in NewMember.js, wrap the {formContent} and button in a <form> tag
<form>
  {formContent}
  <button>Add Member</button>
</form>

//and move the event handler to the form's onSubmit event
<form onSubmit={this.postDataHandler}>

//in the post handler add the event parameter and prevent the default action
postDataHandler = (evt) => {
  evt.preventDefault();
```

Form Submit demo – extract data

```
//still in the post data handler, extract the data from its new location:
const data = {
  firstName: this.state.memberForm.firstName.value,
  lastName: this.state.memberForm.lastName.value,
  title: this.state.memberForm.title.value,
  bio: this.state.memberForm.bio.value
}

//demo, submit and it should all work
```

Validation demo – setup

```
//want to now pass a validation hashmap to configInput(), already using a default value
//for "value", so add a validation param before that (in formBuilder.js)
configInput(kind, type, label, placeholder, validation={}, value='') {

//then try modifying just one field (in NewMember.js)
//we still don't want a custom placeholder text, but we have to pass validation after that
firstName: formBuilder.configInput('input', 'text', 'First Name', null, {required: true}),

//demo – we have lost the default placeholder text b/c it's only looking for 'undefined'

//modify the conditional in configInput()
if (placeholder !== null && typeof placeholder !== 'undefined')
```

Validation Demo – requirements

```
//now we can pass validation rules to the input configurator
//in NewMember.js:
lastName: formBuilder.configInput('input', 'text', 'Last Name', null,
  {required: true, minLength: 2}),
Title: formBuilder.configInput('input', 'text', 'Title', null,
  {required: true}),

//and in formBuilder in the configInput method use the new parameter:
const settings = {
  kind, label, value,
  validation,
  attrs: {}
};
```

Validation Demo – implementing checks

```
//in formBuilder.js add a new method to the exported object:
checkValidity(inputConfig) {
  inputConfig.valid = true;

  if (inputConfig.validation.required) {
    if (inputConfig.value.trim() === '') { inputConfig.valid = false; }
  }
  if (inputConfig.validation.minLength) {
    if (inputConfig.value.trim().length < inputConfig.validation.minLength) {
      retVal.valid = false;
    }
  }
}
```

Validation Demo – enforce validation checks

```
//in NewMember.js in the postDataHandler() check the validity each element before processing
let hasError = false;
for (let elt in this.state.memberForm) {
  formBuilder.checkValidity(this.state.memberForm[elt]);
  if (!this.state.memberForm[elt].valid) {
    hasError = true;
  }
}
if (hasError) {
  this.forceUpdate();
  toast.error('you must properly fill out the form');
  return;
}
```

Visual Feedback demo – setup

```
//pass the validity state to the <Input> component – in buildForm() in formBuilder:
<Input key={elem.id} . . . valid={elem.config.valid} . . . />

//in Input.js need to check validity and add a class name to any existing classNames
//before switch() add:
const classes = [];
if (props.attrs.className) classes.push(props.attrs.className.replace('invalid', '').trim());
if (!props.valid) classes.push('invalid');
props.attrs.className = classes.join(' ').trim();

//demo – all fields should now be invalid! Why? "valid" does not yet exist, so it is falsy
//change the line to:
if (props.valid === false) classes.push('invalid');
```

Visual Feedback demo – fix bug

```
//need to re-evaluate validity when each value updates
//in the inputChangedHandler in NewMember.js – before settings state add:
formBuilder.checkValidity(updatedElement);
```

Error Messages demo – setup

```
//when checking validity we can store error messages in the config object
//in checkValidity() in formBuilder add at beginning of function:
inputConfig.errors = {};

//in each validator, if there is an error add a message property to the errors hash
if (inputConfig.value.trim() === '') {
  inputConfig.valid = false;
  inputConfig.errors.required = `You must enter a value for ${inputConfig.label}`);
}

//in buildForm() the error messages need to be passed to the Input component
<Input key={elem.id} . . . errors={elem.config.errors} />
```

Error Messages demo – display messages

```
//in Input.js need to display any error messages that exist – add (before the return)
const errors = [];
if (props.errors) {
  for (let key in props.errors) { errors.push(props.errors[key]); }
}
let errorContent = null;
if (errors.length) {
  const errMsgs = errors.map((err, idx) => (<li key={idx}>{err}</li>));
  errorContent = (<ul className="validation-errors">{errMsgs}</ul>);
}

//and in the return statement, after {elem} add
{errorContent}
```

Lesson 10 – Managing State with Redux

General notes

For a more workshop style class, have the students do the following demos with you, that way they have some experience with the topics before they are tasked with doing the exercise.

Perform the demos in the sample application called "demos" located in this chapter's directory. The sample application called "react-fundamentals" already has the demos completed.

Performing each demo provides excellent opportunity to discuss why each step is being taken and what the various options are at each step.

Redux standalone demo – part 1

```
npm install redux
//create a file in the app root (not in "src") called redux-alone.js
//going to execute using Node.js, so need to use Node's importing syntax:
const redux = require('redux');
const reducer = (state = initialState, action) => {
      return state;
};

const store = redux.createStore(reducer);
console.log(store.getState());
//execute with "node redux-alone.js" and see that state is undefined
```

Redux standalone – part 2

```
//add (before the reducer)
const initialState = {
       counter: 0
};

//modify reducer:
const reducer = (state = initialState, action) => {

//re-execute and see that state is initialized now
```

Redux standalone – part 3

```
//dispatch an action – after existing code:
store.dispatch({type: 'INCREMENT'});
store.dispatch({type: 'ADD', payload: 5);
console.log(store.getState());

//re-execute – no change in state!
//of course not, we haven't done anything in the reducer function
```

Redux standalone – part 4

```
//modify the reducer function:
if (action.type === 'INCREMENT') {
        //do NOT mutate existing state!
        return { ...state, counter: state.counter + 1 };
}
if (action.type === 'ADD') {
        return { ...state, counter: state.counter + action.payload };
}
return state;

//execute again – now we have changed state!
//try dispatching an action called "FOO" and test (should be no changes to state)
```

Redux standalone – part 5

```
//create a subscription to the state – before dispatching the actions
store.subscribe(() => {
      console.log('(subscription)', store.getState());
});

//and comment out existing console.log() invocations
//execute and see the state each time an action is dispatched
```

React Redux demo – setup

```
//create folders "src/store/reducers" and in there a file called "counter.js"
const initialState = {
  counter: 0
};
const reducer = (state = initialState, action) => {
  return state;
}
export default reducer;

//in index.js:
import { createStore } from 'redux';
import reducer from './store/reducers/counter';

const store = createStore(reducer);
```

React Redux demo - connecting

```
//install react-redux
npm install react-redux

//in index.js
import { Provider } from 'react-redux';

//and
ReactDOM.render(<Provider store={store}><App /></Provider>, . . . );
```

React Redux demo – connecting a container component

```
//in Counter.js:
import { connect } from 'react-redux';

//and
const mapStateToProps = state => {
  return { counter: state.counter };
};

export default connect(mapStateToProps)(Counter);

//demo – no errors, but clicking buttons no longer changes the counter
```

React Redux demo – continued

```
//still in counter.js
const mapDispatchToProps = dispatch => {
  return {
    onIncrement: () => dispatch({type: 'INCREMENT'})
  };
};

export default connect(mapStateToProps, mapDispatchToProps)(Counter);

//on CounterControl for increment change to:
<CounterControl label="Increment" clicked={this.props.onIncrement} />

//add the increment action processing to the reducer function and demo
```

React Redux demo – continued

```
//in Counter.js add to mapDispatchToProps:
onAdd: (amount) => dispatch({type: 'ADD', payload: amount});

//modify the counter control:
<CounterControl label="Add 5" clicked={() => this.props.onAdd(5)}

//and modify reducer.js
//demo both buttons

//have students implement "decrement" and "subtract"
//at that point, state and counterChangedHandler can be removed from Counter.js
```

Action type constants demo

```
//create folder "src/store/actions" and in it a file "actionTypes.js"
export const INCREMENT = 'INCREMENT';
export const DECREMENT = 'DECREMENT';
export const ADD = 'ADD';
export const SUBTRACT = 'SUBTRACT';

//in reducers/counter.js import the constants and use them in the switch statement
import * as actionTypes from '../actions/actionTypes';

switch (action.type) {
  case actionTypes.INCREMENT:
  . . .
```

Action creators demo

```
//create file store/actions/counter.js:
import * as actionTypes from './actionTypes';

export const increment = () => {
  return { type: actionTypes.INCREMENT };
};

export const add = (amt) => {
  return { type: actionTypes.ADD, payload: amt };
};

...etc.
```

Action creators demo – continued

```
// in containers/Counter/Counter.js:
import * as actions from '../../store/actions/counter';

//and mapDispatchToProps becomes:
const mapDispatchToProps = dispatch => {
  return {
    onIncrement: () => dispatch(actions.increment()),
    onAdd: (amount) => dispatch(actions.add(amount)),
    onDecrement: () => dispatch(actions.decrement()),
    onSubtract: (amount) => dispatch(actions.subtract(amount))
  };
};
```

After exercise - review

```
//it would be a good idea to review the exercise solution with the students,
//in particular pointing out how it is set up to have multiple reducers
//each in charge of a different portion of the state,
//and how the actions and reducers are gathered together in the "index.js" files
//in their corresponding directories, and how that makes it easy for the rest of
//the application to import them by importing from the directory name.
```

Lesson 11 – Async Redux

General notes

For a more workshop style class, have the students do the following demos with you, that way they have some experience with the topics before they are tasked with doing the exercise.

Perform the demos in the sample application called "demos" located in this chapter's directory. The sample application called "react-fundamentals" already has the demos completed.

Performing each demo provides excellent opportunity to discuss why each step is being taken and what the various options are at each step.

Middleware demo – creating

```
//in redux-alone.js, before creating the store:
const logger = store => {
  return next => {
    return action => {
      console.log('(middleware) – dispatching', action.type);
      const result = next(action);
      console.log('(middleware) – next state', store.getState());
      return result;
    }
  }
}
const store = redux.createStore(reducer, redux.applyMiddleware(logger));
//execute using: node redux-alone.js
```

Using Redux middleware in React demo

```
//copy logger middleware function into index.js, also:
import { createStore, applyMiddleware } from 'redux';

const store = createStore(reducer, applyMiddleware(logger));
//reload and demo
```

Redux devtools demo

```
//search for and install redux devtools in Chrome browser
//load application and open Redux devtools – see message about instructions, click link
//add to index.js:
import { createStore, applyMiddleware, compose } from 'redux';

const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;

const store = createStore(reducer, composeEnhancers(applyMiddleware(logger)));

//demo some of the redux devtools features, especially the timeline
```

Async action demo

```
//in store/actions/counter.js add a new action creator:
export const delayAdd = (amt, dispatch) => {
  setTimeout(() => { dispatch({type: actionTypes.ADD, payload: amt}); }, 3000);
};

//in components/Counter/Counter.js in mapDispatchToProps add:
onDelayAdd: (amount) => actions.delayAdd(amount, dispatch)

//and add a new button:
<CounterControl label="Delay Add 7" clicked={() => this.props.onDelayAdd(7)} />

//demo async functionality
```

redux-thunk example - setup

```
//shutdown the app server, install and then re-start app server
npm install redux-thunk

//then in index.js
import thunk from 'redux-thunk';
import { compose } from 'redux';

//and when creating the store:
const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__
  || compose;
const store = createStore(reducer, composeEnhancers(applyMiddleware(logger, thunk)));
```

redux-thunk example - execution

```
//in components/Counter/Counter.js, change the delayAdd invocation back to standard:
onDelayAdd: (amount) => dispatch(actions.delayAdd(amount))

//in store/actions/counter.js, change the delayAdd action creator:
export const delayAdd = (amt) => {
  return dispatch => {
    setTimeout(() => {
      dispatch({type: actionTypes.ADD, payload: amt });
      }, 3000);
  };
};

//demo and point out that delayAdd never makes it to the store
// it doesn't show in Redux devtools
```

Moving board members to state – defining actions

```
//in store/actions/actionTypes.js add:
export const ADD_MEMBER = 'ADD_MEMBER';
export const DELETE_MEMBER = 'DELETE_MEMBER';
export const STORE_MEMBERS = 'STORE_MEMBERS';
```

Loading members – action creators

```
//create a new file store/actions/board.js
import * as actionTypes from './actionTypes';
import axios from 'axios';

export const storeMembers = (members) => {
  return { type: actionTypes.STORE_MEMBERS, payload: members };
};

export const loadMembersAsync = () => {
  return dispatch => {
    axios.get('/boardmembers').then(resp => dispatch(storeMembers(resp.data)))
  };
};
```

Detour: providing easy access to all actions

```
//components may want to dispatch actions from different groups,
//so create a file store/actions/index.js
export { storeMembers, loadMembersAsync } from './board';
export { increment, decrement, add, subtract, delayAdd } from './counter';

//now, components can just
import * as actionCreators from '../store/actions';

//and use them all as:
dispatch(actionCreators.loadMembers());
```

Loading members – reducer

```
//create a file store/reducers/board.js
import * as actionTypes from '../actions/actionTypes';
const initialState = { board: [] };

const reducer = (state = initialState, action) => {
  switch (action.type) {
    case actionTypes.STORE_MEMBERS:
      return { ...state, board: action.payload };
    default:
      return state;
  }
};

export default reducer;
```

Combining reducers

```
//in index.js:
import { combineReducers } from 'redux';  //add to existing imports from 'redux'
import counterReducer from './store/reducers/counter';   //change the name
import boardReducer from './store/reducers/board';

const rootReducer = combineReducers({
  board: boardReducer,
  counter: counterReducer
});
const store = createStore(rootReducer, composeEnhancers( . . .));   //change reducer name

//in components/Counter/Counter.js modify mapStateToProps()
return { counter: state.counter.counter }
```

Utilizing state for board members

```
//in containers/Board/Board.js
import { connect } from 'react-redux';
import * as actions from '../../store/actions';

const mapStateToProps = state => { return { members: state.board.board }; };
const mapDispatchToProps = dispatch => {
  return {
    onLoad: () => dispatch(actions.loadMembersAsync())
  };
};

export default connect(mapStateToProps, mapDispatchToProps)(Board);
```

Utilizing state for board members – continued

```
//still in Board.js delete state and modify render() to:
render() {
  const members = this.props.members.map(. . .

//modify componentDidMount to:
this.props.onLoad();
```

Add a board member into state – action

```
//in store/actions/board.js
export const addMemberAsync = (newMember) => {
  return dispatch => { axios.post('/boardmembers', newMember)
    .then(resp => dispatch(addMember(resp.data)));
  };
};

export const addMember = (member) => {
  return { type: actionTypes.ADD_MEMBER, payload: member };
};

//add them to the import/export in index.js in the same directory
```

Add board member – reducer

```
//in store/reducers/board.js add
case actionTypes.ADD_MEMBER:
  return { ...state, board: state.board.concat(action.payload) };
```

Using state to add board member

```
//in containers/Board/NewMember.js
import { connect } from 'react-redux';
import * as actions from '../../store/actions';

const mapDispatchToProps = dispatch => {
  return { onSave: (member) => dispatch(actions.addMemberAsync(member)) };
};
export default connect(null, mapDispatchToProps)(NewMember);

//and inside postDataHandler() replace the axios call with
this.props.onSave(data);

//go to Board of Directors to load them and then add a new one
//watch in Redux devtools to see it added to state
```

Problem with our state

```
//in store/actions/board.js modify addMemberAsync to return the axios promise
return axios.post('/boardmembers', . . .

//in containers/Board/NewMember.js the axios promise should be returned from
//onSave() because of the way we built mapDispatchToProps()
//so in postDataHandler() use the promise to navigate after storing the new member
this.props.onSave(data).then(() => {
  this.props.history.push('/board');
}).catch(err => toast.error(err.ToString()));

//reload board, add one, watch it get added to state (in Redux dev tools)
//and then watch state get reset to four board members when Board component loads
// problem is: we're reloading the board state every time that component loads
// instead of only the first time that component loads
```

Fix the problem

```
//in store/actions/board.js modify loadMembersAsync()
export const loadMembersAsync = () => {
  return (dispatch, getState) => {
    const currState = getState();
    if (!currState.board.board.length) {
      axios.get('/boardmembers').then(resp => dispatch(storeMembers(resp.data)))
    }
  };
};
//can now remove the need to visit Board of Directors before Add New
//in NewMember.js add same onLoad() functionality that Board.js has
onLoad: () => dispatch(actions.loadMembersAsync())

componentDidMount() { this.props.onLoad(); }
```

Optional: Deleting members from store

```
//add to store/actions/board.js – and put them in store/actions/index.js as well!
export const deleteMemberAsync = (member) => {
  return dispatch => {
    return axios.delete('/boardmembers/' + member.Id)
      .then(resp => dispatch(deleteMember(member)));
  };
};

export const deleteMember = (member) => {
  return {
    type: actionTypes.DELETE_MEMBER,
    payload: member
  };
};
```

Optional: deleting members from store

```
//in store/reducers/board.js add:
case actionTypes.DELETE_MEMBER:
  return { ...state, board: state.board.filter(elt => elt.Id !== action.payload.Id)};
```

Optional: deleting members from store

```
//in MemberDetail.js
import { connect } from 'react-redux';
import * as actions from '../../store/actions';

//and
const mapDispatchToProps = dispatch => {
  return { onDelete: (member) => dispatch(actions.deleteMemberAsync(member)) };
};
export default connect(null, mapDispatchToProps)(MemberDetail);

//in deleteMemberHandler() replace axios.delete with
this.props.onDelete(this.state.loadedMember)
//leave the .then() and .catch()
```

Lesson 12 – Testing

General notes

For a more workshop style class, have the students do the following demos with you, that way they have some experience with the topics before they are tasked with doing the exercise.

Perform the demos in the sample application called "demos" located in this chapter's directory. The sample application called "react-fundamentals" already has the demos completed.

Performing each demo provides excellent opportunity to discuss why each step is being taken and what the various options are at each step.

Test orientation

```
//examine containers/App.test.js, describe it and then run the tests with:
npm test
```

First test demo – setup

```
// Jest should already be installed, but need to install:
npm install enzyme react-test-renderer enzyme-adapter-react-16

//create file components/Counter/CounterOutput.test.js
import React from 'react';
import CounterOutput from './CounterOutput';
import { configure, shallow } from 'enzyme';
import Adapter from 'enzyme-adapter-react-16;

//connect Enzyme to our React environment
configure({ adapter: new Adapter() });
```

First test demo – the test

```
//still in CounterOutput.test.js
describe('<CounterOutput /> tests', () => {
  it('should render exactly one div', () => {
    const wrapper = shallow(<CounterOutput />);
    expect(wrapper.find('div')).toHaveLength(1);
  });
});

//run tests with:
npm test
```

Second test demo – props

```
//still in CounterOutput.test.js
  it('should render props.value in a div', () => {
    const wrapper = shallow(<CounterOutput value="42" />);

    expect(wrapper.containsMatchingElement(<div>Current Counter: 42</div>)).toEqual(true);
  });

//run tests with:
npm test
```

Setup and teardown demo

```
//refactor the .test.js to use a suite-level variable for the wrapper and a beforeEach()
describe('<CounterOutput /> tests', () => {
  let wrapper;
  beforeEach(() => {
    wrapper = shallow(<CounterOutput />);
  });
  it('should render exactly one div', () => {
    expect(wrapper.find('div')).toHaveLength(1);
  });
  it('should render props.value in a div', () => {
    wrapper.setProps({ value: 42 });
    expect(wrapper.containsMatchingElement(<div>Current Counter: 42</div>)).toEqual(true);
  });
});
```

Container testing demo

```
//in containers/Counter/Counter.js export the class in addition to the existing export
//create a file containers/Counter/Counter.test.js
import React from 'react';
import { configure, shallow } from 'enzyme';
import Adapter from 'enzyme-adapter-react-16';
import { Counter } from './Counter';
import CounterOutput from '../../components/Counter/CounterOutput';
import CounterControl from '../../components/Counter/CounterControl';

configure({ adapter: new Adapter() });

describe('<Counter /> tests', () => {

});
```

Container testing demo – continued

```
//inside the describe() function:
let wrapper;
beforeEach(() => { wrapper = shallow(<Counter />); });

it('should render <CounterOutput /> when rendering', () => {
  expect(wrapper.find(CounterOutput)).toHaveLength(1);
});

it('should render 5 <CounterControl /> when rendering', () => {
  expect(wrapper.find(CounterControl)).toHaveLength(5);
});

//run, all tests should pass
```

Testing events demo

```
//create file components/Counter/CounterControl.test.js import CounterControl and
//copy in the imports and configure code from CounterOutput.test.js
describe('<CounterControl /> tests', () => {
  let wrapper;
  beforeEach(() => { wrapper = shallow(<CounterControl />); });

  it('should execute its clicked fn when clicked', () => {
    let fn = jest.fn();
    wrapper.setProps({ clicked: fn });
    let elt = wrapper.find('div');
    elt.simulate('click');
    expect(fn).toHaveBeenCalled();
  });
});
```

Testing Redux demo

```
//create a file store/reducers/counter.test.js
import reducer from './counter';
import * as actionTypes from '../actions/actionTypes';

describe('counter reducer tests', () => {
  let initial;
  beforeEach(() => { initial = { counter: 42 }; });

  it('should increment correctly', () => {
    const action = { type: actionTypes.INCREMENT };
    const result = reducer(initial, action);
    expect(result.counter).toEqual(43);
  });
});
```

Lesson 13 – Transitions and Animations

General notes

For a more workshop style class, have the students do the following demos with you, that way they have some experience with the topics before they are tasked with doing the exercise.

Perform the demos in the sample application called "demos" located in this chapter's directory. The sample application called "react-fundamentals" already has the demos completed.

Performing each demo provides excellent opportunity to discuss why each step is being taken and what the various options are at each step.

Initial orientation

```
//review changes made to:
//  containers/Board/Board.js
//  components/Board/MemberDetails.js
//  components/Modal/Backdrop.js
//to make the board member details into a popup modal
```

CSS Transition demo

```
//we want the modal to fade in and slide down from the top when it appears
//in MemberDetail.module.css
.modalOpen {
  display: block;
  opacity: 1;
  transform: translateY(0);
}
.modalClosed {
  display: none;
  opacity: 0;
  transform: translate(-100%);
}
//demo that nothing has changed in the app
```

CSS transition demo – the reveal

```
//still in MemberDetail.module.css
.memberDetail {
  . . .
  transition: all 0.4s ease-out;
}
//still nothing. . . display:none is breaking it
//display cannot be animated and display:none tells the browser there's no use animating
//anything on that element since it is not displayed
//remove display from both .modalOpen and .modalClosed
```

CSS transitions demo – fix a problem

```
//now it works, but the top row of directors is broken – click an no modal
//(may depend on browser window size)
//z-index is killing us now – modal is transparent, but still in front of the tiles
//animate z-index from 200 when open to -1 when closed

//demo – now works



//however, the backdrop suddenly appearing when showing is annoying
//have the students do the same animation on the backdrop
```

CSS animations demo

```
//add some bounce to the modal sliding – in MemberDetails.module.css
@keyframes showModal {
  0% {
    opacity: 0;
    transform: translateY(-100%);
    z-index: -1;
  }
  10% { z-index: 200; }
  90% {
    opacity: 1;
    transform: translateY(10%);
  }
  100% { transform: translateY(0); }
}
```

CSS animations demo – applying the animation

```
//replace the .modalOpen style rules:
.modalOpen {
  animation: showModal 0.4s ease-out forwards;
}
//be sure to add the "forwards" so it doesn't revert to starting state when done
//demo the functionality
```

CSS animations demo – animating out

```
//copy/paste the showModal keyframes to create hideModal – simply reverse all the % numbers
//replace modalClosed styles to use the new animation
.modalClosed {
  animation: closeModal 0.4s ease-in forwards;
}

//change the easing function to mirror the animation on the way in
//discuss easing functions
//perhaps demo chrome dev tools easing function editor
```

CSS limitations demo

```
//inspect in browser and show that even while invisible, backdrop and details present
//modify Board.js to conditionally render the details and backdrop:
{this.state.modalIsOpen ? <Backdrop . . . /> : null }
{this.state.modalIsOpen ? <MemberDetail . . . /> : null }

//demo – animates in but cannot animate out because it is removed from the DOM
```

react-transition-group demo - setup

```
//install
npm install react-transition-group

//convert the Home component into a class-based component and add state:
state={
  display: false
}

//and a method
toggleDisplay = () => {
  this.setState((state, props) => { return { display: !state.display } });
}
```

Transition demo – setup continued

```
//add to the render method
let block = null;
if (this.state.display) {
  block = <div style={{width: '200px', height: '200px', backgroundColor: '#33F'}}></div>;
}

//add to the returned JSX (after the <p> )
<button style={{marginBottom: '1em'}} onClick={this.toggleDisplay}>Toggle</button>
{ block }

//demo that the div is actually added to and removed from the DOM
```

Transition demo - component

```
//still in Home.js import Transition
import { Transition } from 'react-transition-group';

//comment out the current content of "block" and instead put:
const block = <Transition in={this.state.display} timeout={400}>
    {state => <p>{state}</p>}
  </Transition>;

//demo, click the button and see the different states that Transition goes through
```

Transition demo – reveal

```
//modify the content of the <Transition> component to be:
{state => <div style={{
  width: '200px',
  height: '200px',
  backgroundColor: '#33F',
  transition: 'all 0.4s ease-out',
  opacity: state === 'exited' ? 0 : 1
}}></div>

//reload and demo – animation!!
```

Transition demo – fix problem

```
//demo that the content is still in the DOM after it disappears
//add props to the <Transition> component
<Transition in={this.state.display} timeout={400} mountOnEnter unmountOnExit>

//content is now added and removed from DOM, but now we lost the animation
//change the opacity ternary expression to use "exiting" instead of "exited"
```

Animating modal with Transition demo

```
//inside Board.js
import { Transition } from 'react-transition-group';

//in render() replace the conditionally-rendered <MemberDetail> with:
<Transition in={this.state.modalIsOpen} timeout={400} mountOnEnter unmountOnExit>
  {state => <MemberDetail member={this.state.selectedMember}
              show={state} clicked={this.hideModal} /> }
</Transition>

//in MemberDetail.js modify the css class array as follows:
const cssClasses = [ styles.memberDetail, props.show === 'entering' ? styles.modalOpen
  : props.show === 'exiting' ? styles.modalClosed : null ];
//demo now – content is removed. Could now remove all z-index styles
//in both MemberDetail and Backdrop css files
```

Wrapping the Transition demo

```
//in Board.js remove transition and return to rendering MemberDetail always
<MemberDetail member={this.state.selectedMember} show={this.state.modalIsOpen}
clicked={this.hideModal} />


//in MemberDetail.js
import { Transition } from 'react-transition-group';


//in the conditional if (props.member) wrap the content with
<Transition in={props.show} timeout={400} mountOnEnter unmountOnExit>
  {state => {
    return ( . . . Put existing <div> here . . . )
  }}
</Transition>
```

Wrapping the Transition demo – continued

```
//move the css classes array inside the function inside Transition, before the return
//and modify as follows:
const cssClasses = [ styles.memberDetail, state === 'entering' ? styles.modalOpen
    : state === 'exiting' ? styles.modalClosed : null];

//demo – all works except for very first entering – no animation on that one
//why? Normally a component is not transitioned if it is shown when the <Transition>
//component first mounts. To override this add "appear" as a property to <Transition>

//add "appear" as a property on Transition in MemberDetail.js
```

Animation timing demo

```
//in MemberDetail.module.css change the duration of the exit animation to 1s
.modalClosed { animation: closeModal 1s ease-in forwards; }

//demo and show that component is removed before animation is over
//we want enter to be quick and leave to be slow,
//so in MemberDetail.js outside the memberDetail constant:
const animationTiming = {
  enter: 400,
  exit: 1000
};

<Transition . . . timeout={animationTiming} . . . >

//demo and now works!
```

CSSTransition demo

```
//in MemberDetail.js change the import to
import { CSSTransition } from 'react-transition-group';

//change the <Transition> element to <CSSTransition> and remove the cssClasses array
//get rid of the function inside <CSSTransition> leaving the <div> content
//and change its className={styles.memberDetail}
//add classNames="slide-fade" to the <CSSTransition> element

//in the css add to the selectors:
.modelOpen, .slide-fade-enter-active, .slide-fade-appear-active
//and
.modalClosed, .slide-face-exit-active

//demo and it doesn't work (because of CSS modules – remove the modules and it would work)
```

CSSTransition demo – fixing for modules

```
//in MemberDetail.module.css change selectors to:
.modalOpen, .enterActive, .appearActive

//and
.modalClosed, .exitActive

//then, in MemberDetail.js change the CSSTransition element to
<CSSTransition . . . classNames={{...styles}} . . . >

//now, the animations will work!
```

List animation demo - setup

```
//demo list functionality in Counter.js, then add:
import { TransitionGroup, CSSTransition } from 'react-transition-group';

//change the <ul> in the returned JSX to:
<TransitionGroup component="ul" className={styles.list}>

//when building the listItems content wrap each <li> in (move key prop):
<CSSTransition key={item.id} timeout={500} classNames="x">
//NOTE: do not need "in" property for Transition or CSSTransition
//b/c TransitionGroup handles it
```

List animation demo – classes

```
//we are using CSS modules, so in Counter.module.css add the styles:
.enter { opacity: 0; }
.enterActive { opacity: 1; transition: opacity 500ms ease-out; }

.exit { opacity: 1; }
.exitActive { opacity: 0; transition: opacity 500ms ease-out; }

//back in Counter.js change the CSSTransition component
<CSSTransition . . . classNames={{...styles}}>

//demo animation now works!
```

Transitioning router content demo – setup

```
//in ContentContainer.js (factored out of app.js b/c it needs to use router props)
import { TransitionGroup, CSSTransition } from 'react-transition-group';

//in the returned JSX wrap the <Switch> with
<TransitionGroup component="div">
  <CSSTransition timeout={800} classNames={{...styles}}>

//demo – no animation! b/c CSSTransition element gets reused w/ different Switch content
//can use TransitionGroup with one Transition child, but need to change its key
//in order to force animation – i.e. we need a different key for each route
```

Transitioning router content – animations

```
//still in ContentContainer.js
import { Route, Switch, withRouter } from 'react-router-dom';

export default withRouter(contentContainer);

//and
<CSSTransition timeout={800} classNames={{...styles}} key={props.location.key}>

//demo – now some animation, but it's not right
//new content is shown twice, stacked and one fades away
//shown twice b/c both the old and new CSSTransition have a Switch that is getting
//content based on the new URL, so:
<Switch location={props.location}>
//tells it to use the URL from when it was loaded, not the new one
```

Transitioning router content – final fix

```
//need both the new and old content positioned absolutely (on top of each other)
//so they fade one into the other. Transition group is already rendering as a <div>
//and we already have a style built (show styles in ContentContainer.module.css) so:
<TransitionGroup component="div" className={styles.transitionWrapper} />


//finally, wrap the <Switch> in
<div className={styles.transitionSection}>

//done!!
```

Lesson 14 – Introduction to Hooks

General notes

For a more workshop style class, have the students do the following demos with you, that way they have some experience with the topics before they are tasked with doing the exercise.

Perform the demos in the sample application called "demos" located in this chapter's directory. The sample application called "react-fundamentals" already has the demos completed.

Performing each demo provides excellent opportunity to discuss why each step is being taken and what the various options are at each step.

Initial orientation

```
//review changes made to:
//  containers/Counter/Counter.js
//  components/ToDo/ToDoList.js
//to separate the to do list into its own (functional) component
```

useState() demo

```
// in components/ToDo/ToDoList.js add an import
import React { useState } from 'react';

// first thing inside the functional component, add (and discuss what useState does):
const [state, setState] = useState({ toDos: initialToDos, nextId: 4 });

// after the handler functions, add:
const listItems = state.toDos.map(item => (
  // uncomment the <CSSTransition> content here
));

// inside the returned JSX, inside the <TransitionGroup> element, add:
{listItems}
// save and view rendered list
```

updating state demo

```
// inside addItemHandler, where "add to do item here" comment is, add:
const newToDo = { id: state.nextId, text: text };
setState({ toDos: [...state.toDos, newToDo], nextId: state.nextId + 1 });

// inside removeItemHandler, add:
setState({ toDos: state.toDos.filter(item => item.id !== id) });

// save and demo that add and remove work
// however, try to add after removing and point out the runtime error

// if React Developer Tools is added to your browser, refresh the page, inspect the
component and go through the same steps. Point out that state loses the nextId property when
removing a to do item (b/c useState() does not merge state the way class-based state does)
```

fixing updating state demo

```
// change the removeItemHandler code to:
setState({
  toDos: state.toDos.filter(item => item.id !== id),
  nextId: state.nextId
});

// save and demo that the issue is now resolved
```

multiple states demo

```
// change the useState() invocation to:
const [toDos, setToDos] = useState(initialToDos);
const [nextId, setNextId] = useState(4);

// fixup display:
const listItems = toDos.map(. . .)   // remove "state."

// fixup addItemHandler by removing "state.", and replacing setState() with:
setToDos([...toDos, NewToDo]);
setNextId(nextId + 1);

//and fixup removeItemHandler by changing to:
setToDos(toDos.filter(item => item.id !== id));
```

best practices for updating state demo

```
// when new state depends upon current state, should always use functional form
// change addItemHandler to use the following form to set state:
setToDos(prevToDos => [...prevToDos, newToDo]);
setNextId(prevId => prevId + 1);

// and change removeItemHandler to use the following form to set state:
setToDos(prevToDos => prevToDos.filter(item => item.id !== id));
```

Lesson 15 – Side Effects

General notes

For a more workshop style class, have the students do the following demos with you, that way they have some experience with the topics before they are tasked with doing the exercise.

Perform the demos in the sample application called "demos" located in this chapter's directory. The sample application called "react-fundamentals" already has the demos completed.

Performing each demo provides excellent opportunity to discuss why each step is being taken and what the various options are at each step.

HTTP requests demo

```
// review the addition of /containers/talent.js
// we want to query the api for the data so add (after call to useState() )
fetch('https://www.kazoopromotions.com/api/talent').then(response => {
  return resp.json();
}).then(data => {
  setArtists(data);
});

// bring this page up in the browser and use the browser dev tools
// to show the infinite loop of XHR requests being sent

// comment out the setArtists(data) to stop the infinite loop
```

useEffect() demo

```
// continuing in containers/Talent/Talent.js
// add an import for useEffect and put the fetch() and its callbacks inside:
useEffect(() => {
  fetch('https://www.kazoopromotions. . .

});

// uncomment the call to setArtists() and demo the infinite loop of requests
// then comment out the call to setArtists to stop the loop

// describe how this form of useEffect behaves like componentDidUpdate
```

fixing the infinite loop

```
// continuing in containers/Talent/Talent.js
// add an empty array as a second parameter to useEffect()
// and uncomment the call to setTalent()
useEffect(() => {
  fetch('...').then({

  });
}, []);

// demo that the AJAX call is made only once – we have data and no infinite loop!
// describe why (the array is a list of dependencies – the effect will only be
// re-evaluated when any of the dependencies change)
// this form of useEffect() behaves like componentDidMount
```

multiple useEffect demo

```
// still inside Talent.js, add a second call to useEffect()
useEffect(() => {
  console.log('second effect is executing');
});

// save, view in browser, show and explain why the message shows up twice
```

effect dependencies demo - setup

```
// in components/ToDo/ToDoList.js, set up state for the filter text box, add:
const [filter, setFilter] = useState('');

// and modify the input type="text", as follows:
<input type="text"
  value={filter}
  onChange={evt => setFilter(evt.target.value)}
  />

// now, consider if we wanted to re-query for data when the filter changes
// typically, you would have to promote the onChange handler to an actual function
// and code the side-effect there. Then code for loading data would be in both the
// componentDidMount handler and the filter onChange handler.
```

effect dependencies demo

```
// instead, add a new value to state
const [displayedToDos, setDisplayedToDos] = useState([]);

// and a new side effect (don't forget to import useEffect)
useEffect(() => {
  setDisplayedToDos(toDos.filter(elt => filter === '' || elt.text.includes(filter)));
}, [toDos, filter]);

// finally, change the listItems JSX content to be generated from displayedToDos
// instead of toDos

// demo the new functionality
```

useRef() demo - setup

```
// need to debounce the filter keystrokes so it doesn't execute on every keypress
// import useRef from react and then add:
const inputRef = useRef(null);

// and on the <input> element in the returned JSX:
<input ref={inputRef} . . .
```

useRef() demo - completion

```
// then use the ref to get the current value after a suitable delay and compare it
// with the original value when the timeout was scheduled to see whether to execute
useEffect(() => {
  setTimeout(() => {
    if (filter === inputRef.current.value) {
      // put original effect code here, i.e.
      setDisplayedToDos(toDos.filter(. . .
    }
  }, 500);
}, [toDos, filter, inputRef]);

// be sure to include inputRef in the dependencies of the effect
```

cleaning up demo

```
// currently, a timeout is set for each keystroke, most of which will not execute
// anything useful – for performance, we should cancel any that are not going to execute
// change the useEffect code, adding:
const timerId = setTimeout(() => {
  . . .

}, 500);

return () => {
  clearTimeout(timerId);
};
```

Lesson 16 – Reducers and Context

General notes

For a more workshop style class, have the students do the following demos with you, that way they have some experience with the topics before they are tasked with doing the exercise.

Perform the demos in the sample application called "demos" located in this chapter's directory. The sample application called "react-fundamentals" already has the demos completed.

Performing each demo provides excellent opportunity to discuss why each step is being taken and what the various options are at each step.

show loading indicator demo – part 1

```
// the ToDo component has been updated to use an API to load and save data
// be sure to start the server from classfiles/server before doing this demo
// review the changes to this component with the students, then add a loading indicator:
import BusySpinner from '../UI/BusySpinner';

// add some state to the component
const [isBusy, setIsBusy] = useState(false);

// add (in the JSX) in the <div> with the "Add Item" button:
{isBusy && <BusySpinner></BusySpinner>}
```

loading indicator part 2

```
// in the effect that loads the to do items upon first render, add before the fetch():
setIsBusy(true);

// and in the second then() callback, before setting the to do data:
setIsBusy(false);

// save and demo the functionality (it may load too quickly to see the spinner)

// break the url (e.g. remove the "s" from "localhost") and demo that the
// spinner does not disappear when an error occurs – and no error message is displayed
```

handling the error demo

```
// add:
import { toast } from 'react-toastify';

// add a catch() for the effect that loads the to do items:
.catch(err => {
  toast.error('Error loading To Do items.');
  setIsBusy(false);
});

// save and demo the new functionality (you may have to wait for the fetch() to timeout)
// restore the URL and demo the (now working) functionality
```

useReducer() demo – part 1

```
// create a reducer function outside the component function:
const toDoReducer = (state, action) => {
  let newItems, newDisplayed;
  const applyFilter = (items, filter) =>
    items.filter(elt => filter === '' || elt.text.includes(filter));
  switch (action.type) {
    case 'SET_ITEMS':
    case 'SET_FILTER':      // SET_FILTER and APPLY_FILTER are separate
    case 'APPLY_FILTER':    // to keep the debounce behavior when typing a filter
    case 'ADD':
    case 'DELETE':
    default:
      return state;
  }};
```

useReducer demo – part 2

```
// the 'SET_ITEMS' case:
newItems = action.payload;
newDisplayed = applyFilter(newItems, state.filter);
return { ...state, toDos: newItems, displayedToDos: newDisplayed };

// the 'SET_FILTER' case:
return { ...state, filter: action.payload };

// the 'APPLY_FILTER' case:
newDisplayed = applyFilter(state.toDos, state.filter);
return { ...state, displayedToDos: newDisplayed };
```

useReducer() demo – part 3

```
// the 'ADD' case:
newItems = [...state.toDos, action.payload];
newDisplayed = applyFilter(newItems, state.filter);
return { ...state, toDos: newItems, displayedToDos: newDisplayed };

// the 'DELETE' case:
newItems = state.toDos.filter(elt => elt.id !== action.payload);
newDisplayed = applyfilter(newItems, state.filter);
return { ...state, toDos: newItems, displayedToDos: newDisplayed };

// remove the useState() for toDos, filter and displayedToDos, and add:
const [state, dispatch] = useReducer(toDoReducer, { toDos: [],
  displayedToDos: [], filter: ''});
```

useReducer demo – part 4

```
// in the effect that loads the toDo items, replace setToDos() with:
dispatch({ type: 'SET_ITEMS', payload: data });

// in the effect that implements the debounce functionality, change to:
if (state.filter === . . . ) {
  dispatch({ type: 'APPLY_FILTER' });
}
// and change the dependencies to include state
```

useReducer demo – part 5

```
// in the addItemHandler, replace the setToDos() with:
dispatch({ type: 'ADD', payload: data.item });

//in the removeItemHandler, replace setToDos() with:
dispatch({ type: 'DELETE', payload: id });

//modify the creation of the listItems JSX:
const listItems = state.displayedToDos.map( . . .

// and in the returned JSX the <input> should have:
value = {state.filter}
onChange = {evt => dispatch({ type: 'SET_FILTER', payload: evt.target.value })}
```

reducer for http state demo

```
// create a new reducer function
const httpReducer = (httpState, action) => {
  switch (action.type) {
    case 'SENT':
      return { loading: true, error: null };
    case 'RESPONSE':
      return { ...httpState, loading: false };
    case 'ERROR':
      return { loading: false, error: action.payload };
    default:
      return httpState;
  }
};
```

using reducer for http state demo

```
// get rid of the useState() for isBusy and instead use the reducer:
const [ httpState, dispatchHttp ] = useReducer(httpReducer, {loading: false, error: null});

// fixup the errors for setIsBusy with calls to dispatchHttp()
setIsBusy(true) becomes dispatchHttp({type: 'SENT' })
setIsBusy(false) depends upon where it is found:
    in .then() it becomes dispatchHttp({type: 'RESPONSE'});
    in .catch() it becomes dispatchHttp({ type: 'ERROR', payload: 'Error loading' });

// in the JSX:
isBusy becoves httpState.loading

// remove the toast error and below the to do item list add:
{httpState.error && <p class="danger">{httpState.error}</p>}
```

useContext demo – part 1

```
// the user must agree to the site terms of use before they can see any other content
// the root App component has been refactored into a functional component,
// so useContext is necessary to let it work with the Context API
// inside the Context folder, create a new file called termsContext.js and enter:
import React, { useState } from 'react';

export const TermsContext = React.createContext({
  accepted: false,
  accept: () => {}
});
```

useContext demo – part 2

```
// continue in the context file
const TermsContextProvider = props => {
  const [accepted, setAccepted] = useState(false);
  const acceptHandler = () => {
    setAccepted(true);
  };

  return (
    <TermsContext.Provider value={{ accept: acceptHandler, accepted: accepted }}>
      {props.children}
    </TermsContext.Provider>
  );
};
export default TermsContextProvider;
```

useContext demo – part 3

```
// inside index.js add:
import TermsContextProvider from './context/termsContext';

// and change the returned JSX:
ReactDOM.render(
  <TermsContextProvider>
    <Provider store={store}><App /></Provider>
  </TermsContextProvider>, document.getElementById('root')
);
```

useContext demo – part 4

```
// inside app.js:
import React, { useContext } from 'react';
import { TermsContext } from '../context/termsContext';
import SiteTerms from '../components/UI/SiteTerms';

// inside the component function:
const termsContext = useContext(TermsContext);

let content = <SiteTerms></SiteTerms>;
if (termsContext.accepted) {
  content = <<paste currently-returned content here>>
}
return content;
```

useContext demo – part 5

```
// inside SiteTerms.js:
import React, { useContext } from 'react';
import { TermsContext } from '../../context/termsContext';

// inside the class
const termsContext = useContext(TermsContext);

const acceptHandler = () => {
  termsContext.accept();
};

// save and demo the functionality – every time you refresh the page,
// you will have to agree to the site terms
```

Lesson 17 – Custom Hooks

General notes

For a more workshop style class, have the students do the following demos with you, that way they have some experience with the topics before they are tasked with doing the exercise.

Perform the demos in the sample application called "demos" located in this chapter's directory. The sample application called "react-fundamentals" already has the demos completed.

Performing each demo provides excellent opportunity to discuss why each step is being taken and what the various options are at each step.

first hook demo

```
// create a new directory src/hooks and in it create a file toggle.js and add:
import { useReducer } from 'react';

const useToggle = (initial = false) => {
  return useReducer((state) => !state, initial);
};

export default useToggle;
```

using first hook demo

```
// inside containers/home.js import the new hook
import useToggle from '../hooks/toggle';

// change the component into a functional component, remove the state and toggleDisplay()
const Home = props => {
  const [display, toggleDisplay] = useToggle();
  // remove the render() method but leave the JSX content of the method
  // modify the JSX content – instead of this.state.display just use display
  // instead of this.toggleDisplay just use toggleDisplay

};
```

localStorage hook demo – part 1

```
// create a new file hooks/localStorage.js and add:
import { useState, useEffect } from 'react';

const useLocalStorage = (key, def) => {
  // content on next slide
};

export default useLocalStorage;
```

localStorage hook demo – part 2

```
const [state, setState] = useState(() => {
  let value;
  try {
    value = JSON.parse(window.localStorage.getItem(key) || String(def));
  } catch (e) {
    value = def;
  }
  return value;
});
useEffect(() => {
  window.localStorage.setItem(key, state);
}, [state]);

return [state, setState];
```

using localStorage hook demo

```
// in containers/Counter/Counter.js, convert it into a functional component,
// strip out all mapStateToProps and mapDispatchToProps and add:
import useLocalStorage from '../../hooks/localStorage';

// in the component:
const [counter, setCounter] = useLocalStorage('kazoo-counter', 0);

//modify the clicked={} attributes of each <CounterControl> as follows:
() => setCounter(counter + 1)
() => setCounter(counter - 1)
() => setCounter(counter + 5)
() => setTimeout(() => setCounter(counter + 7), 2000)
() => setCounter(counter - 5)
```

problem with usage of localStorage hook demo

```
// test out the counter component. it should work (mostly)
// try clicking the Delay Add 7 button and then quickly the Subtract 5 button
// 5 gets subtracted, but then after 2 seconds, 7 is added to the original number,
// not the number - 5
// modify the clicked code for the Delay Add 7 button to:
() => setTimeout(() => setCounter(c => c + 7), 2000)

// now, it should work properly
```

whyDidYouUpdate demo

```
// review the whyDidYouUpdate.js hook
// in containers/Counter/Counter.js (or any other functional component) add:
import useWhyDidYouUpdate from '../../hooks/whyDidYouUpdate';

// inside the functional component add:
useWhyDidYouUpdate('Counter', props);

// click a few menu items, leaving the counter and coming back to demo the output
```

useArray convenience hook example – part 1

```
// create a new file hooks/array.js and add
import { useCallback, useState } from 'react';

const useArray = initialValues => {
  const [value, setValue] = useState(initialValues);

  return {
    // content on next slide
  }
};

export default useArray;
```

useArray convenience hook example – part 2

```
return {
  value,
  setValue,
  add: useCallback(item => setValue(arr => [...arr, item])),
  clear: useCallback(() => setValue(() => [])),
  removeById: useCallback(id => setValue(arr => arr.filter(v => v && v.id !== id))),
  removeByIndex: useCallback(idx => setValue(arr => {
    arr.splice(idx, 1);
    return [...arr];
  }))
}
```

useArray convenience hook example – part 3

```
// create a new components/ShoppingList.js and enter:
import React from 'react';
import useArray from '../hooks/array';

const ShoppingList = props => {
  const list = useArray(['Eggs', 'Milk', 'Bread']);

  return (
    // content on next slide
  );
};

export default ShoppingList;
```

useArray convenience hook demo – part 4

```
<div>
  <h1>Shopping List</h1>
  <button onClick={() => list.add('something')}>Add</button>
  <ul>
    {list.value.map((item, i) => (
      <li key={i}>
        <button onClick={() => list.removeByIndex(i)}>Delete</button>
         {item}
      </li>
    ))}
  </ul>
  <button onClick={list.clear}>Clear</button>
</div>
```

useArray convenience hook demo – part 5

```
// open containers/Home.js and import and add the new component:
import ShoppingList from '../components/ShoppingList';

// and in the returned JSX:
return (
  <div>
    // other content
    <ShoppingList></ShoppingList>
  </div>
)
```