LEARNING REACT

KIRUPA CHINNATHAMBI

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Learning React

Kirupa Chinnathambi

Addison-Wesley

Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi
Mexico City • Sao Paulo • Sidney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Learning React

Copyright © 2017 Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-134-54631-5

ISBN-10: 0-134-54631-8

Library of Congress Control Number: 2016917161

Printed in the United States of America

First printing: November 2016

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Pearson cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the United States, please contact intlcs@pearsoned.com.

Acquisitions Editor
Mark Taber

Development Editor
Chris Zahn

Copy Editor
Abigail Manheim

Indexer
Erika Millen

Technical Reviewers
Trevor McCauley
Kyle Murray

Cover Designer
Chuti Prasertsith

❖

To my dad!

(Who always believed in me—even if what I was often doing made no sense to him...or to me for that matter! :P)

❖

Contents

1 Introducing React 1

Old School Multi-Page Design 2

New School Single-Page Apps 3

Meet React 6

- Automatic UI State Management 7
- Lightning-fast DOM Manipulation 8
- APIs to Create Truly Composable UIs 9
- Visuals Defined Entirely in JavaScript 9
- Just the V in an MVC Architecture 11

Conclusion 11

2 Building Your First React App 13

Dealing with JSX 14

Getting Your React On 15

Displaying Your Name 16

It's All Still Familiar 18

- Changing the Destination 18
- Styling It Up! 19

Conclusion 21

3 Components in React 23

A Quick Review of Functions 24

Changing How We Deal with the UI 26

Meet the React Component 29

- Creating a Hello, World! Component 30
- Specifying Properties 32
- Dealing with Children 34

Conclusion 36

4 Styling in React 37

Displaying Some Vowels 37

Styling React Content Using CSS 40

- Understand the Generated HTML 40
- Just Style It Already! 41

Styling Content the React Way 42

Creating a Style Object 43

Actually Styling Our Content 43

You Can Omit the “px” Suffix 45

Making the Background Color Customizable 45

Conclusion 47

5 Creating Complex Components 49

From Visuals to Components 49

Identifying the Major Visual Elements 51

Identifying the Components 53

Creating the Components 56

The Card Component 58

The Square Component 60

The Label Component 61

Passing Properties, Again! 63

Why Component Composability Rocks 66

Conclusion 66

6 Transferring Properties (Props) 69

Problem Overview 69

Detailed Look at the Problem 74

Meet the Spread Operator 78

Properly Transferring Properties 78

Conclusion 80

7 Meet JSX—Again! 81

What Happens with JSX? 81

JSX Quirks to Remember 83

You Can Only Return A Single Root Node 83

You Can’t Specify CSS Inline 84

Reserved Keywords and `className` 85

Comments 86

Capitalization, HTML Elements, and Components 87

Your JSX Can Be Anywhere 88

Conclusion 88

8 Dealing with State 89

Using State 89

Our Starting Point 90

Getting Our Counter On 93

Setting the Initial State Value 93

Starting Our Timer and Setting State 94

Rendering the State Change 96

Optional: The Full Code 96

Conclusion 98

9 Going from Data to UI 99

The Example 99

Your JSX Can Be Anywhere—Part II 102

Dealing with Arrays in the Context of JSX 103

Conclusion 105

10 Working with Events 107

Listening and Reacting to Events 107

Starting Point 108

Making the Button Click Do Something 110

Event Properties 112

Doing Stuff With Event Properties 114

More Eventing Shenanigans 115

Listening to Regular DOM Events 117

The Meaning of `this` Inside the Event Handler 119

React…Why? Why?! 120

Browser Compatibility 120

Improved Performance 120

Conclusion 121

11 The Component Lifecycle 123

Meet the Lifecycle Methods 123

See the Lifecycle Methods in Action 124

The Initial Rendering Phase 127

The Updating Phase 129

The Unmounting Phase 132

Conclusion 133

12 Accessing DOM Elements 135
Meet Refs 137
Conclusion 142

13 Creating a Single-Page App Using React Router 143
The Example 144
Building the App 146
- Displaying the Initial Frame 147
- Displaying the Home Page 149
- Interim Cleanup Time 151
- Displaying the Home Page Correctly 154
- Creating the Navigation Links 155
- Adding the Stuff and Contact Views 157
- Creating Active Links 159

Conclusion 161

14 Building a Todo List App 163
Getting Started 164
Creating the UI 165
Creating the Functionality 168
- Initializing our State Object 169
- Handling the Form Submit 169
- Populating Our State 171
- Displaying the Tasks 173
- Adding the Finishing Touches 176

Conclusion 178

15 Setting Up Your React Development Environment 179
Meet the Tools 182
- Node.js 182
- Babel 182
- webpack 183
- Your Code Editor 183

It Is Environment Setup Time! 184
    Setting up our Initial Project Structure 184
    Installing and Initializing Node.js 187
    Installing the React Dependencies 190
    Adding our JSX File 191
    Going from JSX to JavaScript 193
    Building and Testing Our App 197
Conclusion 198

16 The End 199

Index 201

Acknowledgments

First, none of this would be possible without the support and encouragement of my awesome wife, **Meena**. If she didn't put her goals on hold to allow me to spend six months designing, writing, and re-writing everything you see here, me writing this book would have been a distant dream.

Next, I'd like to thank **my parents** for always encouraging me to aimlessly wander and enjoy free time to do what I liked—such as teaching complete strangers over the internet in the late 1990s how to do cool things with programming. I wouldn't be half the rugged indoorsman/scholar/warrior I am today without them both :P

On the publishing side, writing the words you see here is the easy part. Getting the book into your hands is an amazingly complex process. The more I learn about all the moving pieces involved, the more impressed I am at all the individuals who work tirelessly behind the scenes to keep this amazing machinery running. **To everyone at Pearson** who made this possible, thank you! There are a few people I'd like to explicitly call out, though. First, I'd like to thank **Mark Taber** for continuing to give me opportunities to work together, **Chris Zahn** for patiently addressing my numerous questions/concerns, **Abby Manheim** for turning my version of English into something human-understandable, and **Loretta Yates** for helping make the connections a long time ago that made all of this happen. The technical content of this book has been reviewed in great detail by my long-time friends and online collaborators, **Kyle Murray (aka Krilnon)** and **Trevor McCauley (aka senocular)**. I can't thank them enough for their thorough (and frequently, humorous!) feedback.

About the Author

Kirupa Chinnathambi has spent most of his life trying to teach others to love web development as much as he does.

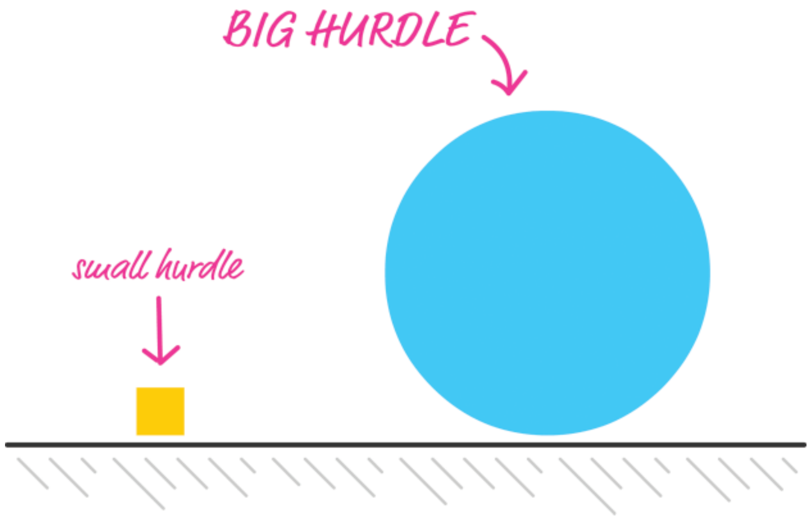
In 1999, before blogging was even a word, he started posting tutorials on kirupa.com. In the years since then, he has written hundreds of articles, written a few books (none as good as this one, of course!), and recorded a bunch of videos you can find on YouTube. When he isn't writing or talking about web development, he spends his waking hours helping make the Web more awesome as a Program Manager in Microsoft. In his non-waking hours, he is probably sleeping...or writing about himself in the third person.

You can find him on Twitter (twitter.com/kirupa), Facebook (facebook.com/kirupa), or e-mail (kirupa@kirupa.com). Feel free to contact him anytime.

2

Building Your First React App

By now, thanks to the previous chapter, you probably know all about the backstory of React and how it helps even your most complex user interfaces sing performantly. For all the awesomeness that React brings to the table, getting started with it (kinda like this sentence) is not the most straightforward thing. It has a steep learning curve filled with many small and big hurdles:



In this chapter, we start at the very beginning and get our hands dirty by building a simple React app. We encounter some of these hurdles head-on, and some of these hurdles we skip over—for now. By the end of this chapter, not only will we have built something you can proudly show off to your friends and family, we'll have set ourselves up nicely for diving deeper into all that React offers in future chapters.

Dealing with JSX

Before we start building our app, there is an important thing we should cover first. React isn't like many JavaScript libraries you may have used. It isn't very happy when you simply refer to code you've written for it using a script tag. React is annoyingly special that way, and it has to do with how React apps are built.

As you know, your web apps (and everything else your browser displays) are made up of HTML, CSS, and JavaScript:

It doesn't matter if your web app was written using React or some other library like Angular, Knockout, or jQuery. *The end result* has to be some combination of HTML, CSS, and JavaScript. Otherwise, your browser really won't know what to do.

Now, here is where the special nature of React comes in. *Besides normal HTML, CSS, and JavaScript, the bulk of your React code will be written in something known as JSX*. JSX, as I mentioned in Chapter 1, is a language that allows you to easily mix JavaScript and HTML-like tags to define user interface (UI) elements and their functionality. That sounds cool and all (and we will see JSX in action in just a few moments), but there is a slight problem. Your browser has no idea what to do with JSX.

To build a web app using React, we need a way to take our JSX and convert it into plain old JavaScript that your browser can understand.

If we didn't do this, our React app simply wouldn't work. That's not cool. Fortunately, there are two solutions to this:

- **Set up a development environment around Node and a handful of build-tools.** In this environment, every time you perform a build, all of your JSX is automatically converted into JS and placed on disk for you to reference like any plain JavaScript file.
- **Let your browser rely on a JavaScript library to automatically convert JSX to something it understands.** You specify your JSX directly just like you would any old piece of JavaScript, and your browser takes care of the rest.

Both of these solutions have a place in our world, but let's talk about the impact of each.

The first solution, while a bit complicated and time-consuming at first, is *the way* modern web development is done these days. Besides compiling (transpiling to be more accurate) your JSX to JS, this approach enables you to take advantage of modules, better build tools, and a bunch of other features that make building complex web apps somewhat manageable.

The second solution provides a quick and direct path where you initially spend more time writing code and less time fiddling with your development environment. To use this solution, all you do is reference a script file. This script file takes care of turning the JSX into JS on page load, and your React app comes to life without you having to do anything special to your development environment.

For our introductory look at React, we are going to use the second solution. You may be wondering why we don't use the second solution always. The reason is that your browser takes a performance hit each time it spends time translating JSX into JS. That is totally acceptable when learning how to use React, but that is totally not acceptable when deploying your app for real-life use. Because of that un-acceptableness, we will revisit all of this and look at the first solution and how to set up your development environment later, once you've gotten your feet comfortably wet in React.

Getting Your React On

In the previous section, we looked at the two ways you have for ensuring your React app ends up as something your browser understands. In this section, we are going to put all of those words into practice. First, we will need a blank HTML page that will act as our starting point.

If you don't have a blank HTML page handy, feel free to use the following:

```
<!DOCTYPE html>
<html>

<head>
  <title>React! React! React!</title>
</head>

<body>
  <script>

  </script>
</body>

</html>
```

This page has nothing interesting or exciting going for it, but let's fix that by adding a reference to the React library. Just below the `title`, add these two lines:

```
<script src="https://unpkg.com/react@15.3.2/dist/react.js"></script>
<script src="https://unpkg.com/react-dom@15.3.2/dist/react-dom.js"></script>
```

These two lines bring in both the core React library as well as the various things React needs to work with the DOM. Without them, you aren't building a React app at all. Now, we aren't done yet. There is one more library we need to reference. Just below these two `script` tags, add the following line:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.23/browser.min.
js"></script>
```

What we are doing here is adding a reference to the Babel JavaScript compiler (http://babeljs.io/). Babel does many cool things, but the one we care about is its capability to turn JSX into JavaScript.

At this point, our HTML page should look as follows:

```
<!DOCTYPE html>
<html>

<head>
  <title>React! React! React!</title>
  <script src="https://unpkg.com/react@15.3.2/dist/react.js"></script>
  <script src="https://unpkg.com/react-dom@15.3.2/dist/react-dom.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.23/browser.min.
js"></script>
</head>

<body>
  <script>

  </script>
</body>

</html>
```

If you preview your page right now, you'll notice that this page is still blank with nothing visible going on. That's OK. We are going to fix that next.

Displaying Your Name

The first thing we are going to do is use React to display our name on screen. The way we do that is by using a method called `render`. Inside your `script` tag, add the following:

```
ReactDOM.render(
  <h1>Sherlock Holmes</h1>,
  document.body
);
```

Don't worry if none of this makes sense at this point. Our goal is to get something to display on screen first, and we'll make sense of what we did shortly afterwards. Now, before we preview this in our page to see what happens, we need to designate this `script` block as something that Babel can do its magic on. The way we do that is by setting the `type` attribute on the `script` tag to a value of `text/babel`:

```
<script type="text/babel">
  ReactDOM.render(
    <h1>Sherlock Holmes</h1>,
    document.body
  );
</script>
```

Once you've made that change, now preview what you have in your browser. What you'll see are the words **Sherlock Holmes** printed in giant letters. Congratulations! You just built an app using React.

As apps go, this isn't all that exciting. Chances are your name isn't even Sherlock Holmes. While this app doesn't have much going for it, it does introduce you to one of the most frequently used methods you'll use in the React universe—the `ReactDOM.render` method.

The `render` method takes two arguments:

- The HTML-like elements (aka JSX) you wish to output
- The location in the DOM that React will render the JSX into

Here is what our render method looks like:

```
ReactDOM.render(
  <h1>Sherlock Holmes</h1>,
  document.body
);
```

Our first argument is the text **Sherlock Holmes** wrapped inside some `h1` tags. This HTML-like syntax inside your JavaScript is what JSX is all about. While we will spend a lot more time drilling into JSX a bit later, I should mention this up front—***It is every bit as crazy as it looks***. Whenever I see brackets and slashes in JavaScript, a part of me dies on the inside because of all the string escaping and quotation mark gibberish I will need to do. With JSX, you do none of that. You just place your HTML-like content as-is just like what we've done here. Magically (like the super-awesome kind involving dragons and laser beams), it all works.

The second argument is `document.body`. There is nothing crazy or bizarre about this argument. It simply specifies where the converted markup from the JSX will end up living in our DOM. In our example, when the `render` method runs, the `h1` tag (and everything inside it) is placed in our document's `body` element.

Now, the goal of this exercise wasn't to display *a* name on the screen. It was to display *your* name. Go ahead and modify your code to do that. In my case, the `render` method will look as follows:

```
ReactDOM.render(
  <h1>Batman</h1>,
  document.body
);
```

Well—it would look like that if my name was Batman! Anyway, if you preview your page now, you will see your name displayed instead of Sherlock Holmes.

It's All Still Familiar

While the JavaScript looks new and shiny thanks to JSX, the end result your browser sees is nice, clean HTML, CSS, and JavaScript. To see this for yourself, let's make a few alterations to how our app behaves and looks.

Changing the Destination

The first thing we'll do is change where our JSX gets output. Using JavaScript to place things directly in your `body` element is never a good idea. A lot can go wrong—especially if you are going to be mixing React with other JS libraries and frameworks. The recommended path is to create a separate element that you will treat as a new root element. This element will serve as the destination our `render` method will use. To make this happen, go back to the HTML and add a `div` element with an `id` value of `container`.

Instead of showing you the full HTML for this one minor change, here is what just our `body` element looks like:

```
<body>
  <div id="container"></div>
  <script type="text/babel">
    ReactDOM.render(
      <h1>Batman</h1>,
      document.body
    );
  </script>
</body>
```

With our **container** `div` element safely defined, let's modify the `render` method to use it instead of `document.body`. Here is one way of doing this:

```
ReactDOM.render(
  <h1>Batman</h1>,
  document.querySelector("#container")
);
```

Another way of doing this is by doing some things outside of the `render` method itself:

```
var destination = document.querySelector("#container");

ReactDOM.render(
  <h1>Batman</h1>,
  destination
);
```

Notice that the destination variable stores the reference to our container DOM element. Inside the render method, we simply reference the same destination variable instead of writing the full element-finding syntax as part of the argument itself. The reason I want to do this is simple. I want to show you that you are still writing JavaScript and render is just another boring old method that happens to take two arguments.

Styling It Up!

Time for our last change before we call it a day. Right now, our names show up in whatever default h1 styling our browser provides. That is just terrible, so let's fix it by adding some CSS. Inside your head tag, add a style block with the following CSS:

```
#container {
  padding: 50px;
  background-color: #EEE;
}
#container h1 {
  font-size: 48px;
  font-family: sans-serif;
  color: #0080A8;
}
```

After you have added all of this, preview your page. Notice that our text appears with a little more purpose than it did earlier when it relied entirely on the browser's default styling (see Figure 2-1).

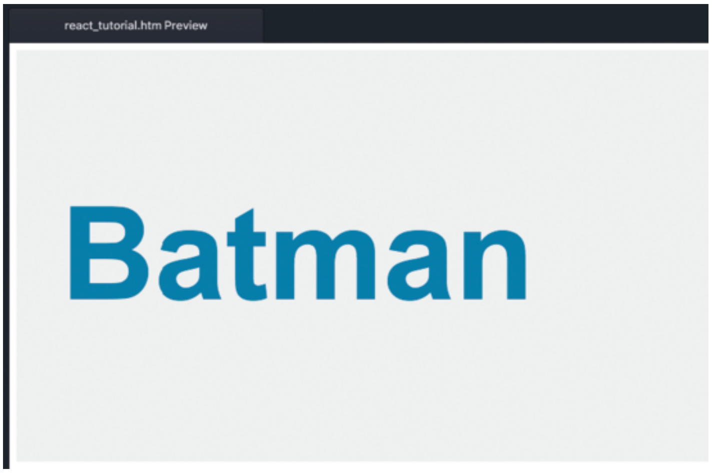


Figure 2-1 The result of adding the CSS.

The reason this works is that our DOM's `body`, after running all of the React code, contains our **container** element with an `h1` tag inside it. It doesn't matter that the `h1` tag was defined entirely inside JavaScript in this JSX syntax or that your CSS was defined well outside of the `render` method. The end result is that your React app is still going to be made up of some 100% organic (and cage-free!) HTML, CSS, and JavaScript:

```
<!DOCTYPE html>
<html>

<head>
  <title>React! React! React!</title>
  <script src="https://unpkg.com/react@15.3.2/dist/react.js"></script>
  <script src="https://unpkg.com/react-dom@15.3.2/dist/react-dom.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.23/browser.min.
js"></script>

  <style>
    #container {
      padding: 50px;
      background-color: #EEE;
    }
    #container h1 {
      font-size: 144px;
      font-family: sans-serif;
      color: #0080a8;
    }
  </style>
</head>

<body>
  <div id="container"></div>
  <script type="text/babel">
    var destination = document.querySelector("#container");

    ReactDOM.render(React.createElement(
      "h1",
      null,
      "Batman"
    ), destination);
  </script>
</body>

</html>
```

Notice that there is nary a trace of React-like code in sight. Also, we should use the word *nary* more often in everyday conversation!

Conclusion

If this is your first time building a React app, we covered a lot of ground here. One of the biggest takeaways is that React is different than other libraries because it uses a whole new language called JSX to define what the visuals will look like. We got a very small glimpse of that here when we defined the `h1` tag inside the `render` method.

JSX's impact goes beyond how you define your UI elements. It also alters how you build your app as a whole. Because your browser can't understand JSX in its native representation, you need to use an intermediate step to convert that JSX into JavaScript. One approach is to build your app to generate the transpiled JavaScript output to correspond to the JSX source. Another approach (aka the one we used here) is to use the Babel library to translate the JSX into JavaScript on the browser itself. While the performance hit of doing that is not recommended for live/production apps, when familiarizing yourself with React, you can't beat the convenience.

In future chapters, we'll spend some time diving deeper into JSX and going beyond the render method as we look at all the important things that make React tick.

This page intentionally left blank

Index

Symbols

{ } (curly brackets), 33, 86–87

... (spread) operator
- explained, 78
- transferring properties with, 78–80

A

accessing
- DOM elements
  - ES6 arrow functions, 141
  - references, 137–141
  - when to use, 135–137
- properties, 33

active links, creating, 159–160

activeClassName property, 159–161

addEventListener function, 118–119

addItem event handler, 170, 171, 177–178

Alpert, Ben, 86

Animaniacs Good Idea / Bad Idea sketches, 36

APIs (application programming interfaces), 9

app frames, 147–149

application programming interfaces (APIs), 9

apps
- catalog browser app
  - multi-page design, 2–3
  - SPA (single-page app) model, 3–6
- creating with React Router

app frame, 147–149
boilerplate markup and code, 146–147
Contact component, 157–158
CSS (Cascading Style Sheets), 151–153
example, 144–145
Home component, 149–151
home page, displaying, 149–151, 154–155
Link component, 156
navigation links, 155–156, 159–160
overview, 143–144
ReactRouter prefix, 153–154
render method, 148–149, 154
Route component, 149
route matching, 158
Router component, 149
Stuff component, 157–158

first React app
blank HTML page, creating, 15–16
destination, changing, 18–19
name, displaying, 16–18
styles, 19–20

MyTotallyAwesomeApp
index.jsx file, 191–192
initial project structure, 184–186
Node.js initialization, 187–189
React dependencies, 190–191
testing, 197–198

Todo List app
addItem event handler, 171, 177–178
app functionality, 168
form submission, 169–170
functionality, 168
initial code listing, 164–165
overview, 163–164
render method, 166, 171, 173–174
state object initialization, 169
state object population, 171–172
styles, 167, 176–178
task display, 173–176
TodoItems component, 173–176
UI (user interface), 165–168

architecture (MVC), 11
arguments. *See* properties
arrays of components, 103–105
arrow functions (ES6), 141
attributes. *See* properties
autobinding, 95
automatic UI state management, 7–8
avoiding ReactRouter prefix, 153–154

B

Babel
overview, 182
referencing, 16, 86
website, 198

background color, customizing, 45–47
backgroundColor property (letterStyle object), 46
bgcolor attribute, 46
blank HTML pages, creating, 15–16
Bower, 198
browser compatibility, 120
building apps. *See* apps
button counter
event handler, 110–112
event listening
regular DOM events, 117–119
SyntheticEvent type, 116–117
event properties, 114–115
initial code listing, 108–110

lifecycle methods
componentDidMount method, 129
componentDidUpdate method, 131
componentWillMount method, 129
componentWillReceiveProps method, 132
componentWillUnmount method, 132
componentWillUpdate method, 131
getDefaultProps method, 128
getInitialState method, 128
initial code listing, 124–127
initial rendering phase, 127–129
render method, 129, 131
shouldComponentUpdate method, 130–131
unmounting phase, 132
updating phase, 129–132
overview, 107–108
SyntheticEvent type, 112–114
this keyword, 119–120
Buttonify component, 35–36

C

calling functions, 25
camelcase, 86
capitalization (JSX), 87–88
Card component, 58–59
Cascading Style Sheets. *See* CSS (Cascading Style Sheets)
catalog browser app
multi-page design, 2–3
SPA (single-page app) model, 3–6
changing destination, 18–19
child components
overview, 34–36
passing properties to, 63–65
transferring properties to
component hierarchy and, 69–74
problems with, 69–77
spread operator (…), 78–80
Circle component
array of, 103–105
circleStyle object, 101
initial code listing, 99–102
render method, 102
showCircle function, 102–103
theCircle variable, 102
circleStyle object, 101
class names, 85–86
CleverComponent, 34
code editors, 183–184
color of background, customizing, 45–47
color palette card
Card component, 58–59
component definitions, 56–58
component identification, 53–56
generated HTML, 66
Label component, 61–63
overview, 49–51
properties, passing to child components, 63–65
Square component, 60–61
visual element identification, 51–53
Colorizer component
behavior of, 135–137
references, 137–141
comments, 86–87
compiler. *See* Babel
component hierarchy, 55–56, 69–70
componentDidMount method, 93, 94–95, 118, 129

componentDidUpdate method, 131
components
  arrays of, 103–105
  Buttonify, 35–36
  capitalization of, 87–88
  child components, 34–36
  Circle
    array of, 103–105
    circleStyle object, 101
    initial code listing, 99–102
    render method, 102
    showCircle function, 102–103
    theCircle variable, 102
  CleverComponent, 34
  color palette card
    Card component, 58–59
    component definitions, 56–58
    component identification, 53–56
    generated HTML, 66
    Label component, 61–63
    overview, 49–51
    properties, passing to child components, 63–65
    Square component, 60–61
    visual element identification, 51–53
  Colorizer
    behavior of, 135–137
    references, 137–141
  component hierarchy, 55–56, 69–70
  composability. *See also* color palette card
    advantages of, 66
    definition of, 49
    overview, 49–51
  Contact, 157–158
  CounterParent
    event handler, 110–112
    event listening, 116–119
    event properties, 114–115
    initial code listing, 108–110
    overview, 107–108
    this keyword, 119–120
  creating
    Card component, 58–59
    component definitions, 56–58
    component identification, 53–56
    generated HTML, 66
    Label component, 61–63
    overview, 49–51
    properties, 63–65
    Square component, 60–61
    visual element identification, 51–53
  definition of, 23–24, 29
  events. *See* events
  generated HTML, 40–41
  HelloWorld component
    creating, 30–32
    properties, 32–34
  Home, 149–151
  Label, 77
  Letter, 37–39
  lifecycle methods
    button counter example, 124–127
    componentDidMount method, 129
    componentDidUpdate method, 131
    componentWillMount method, 129
    componentWillReceiveProps method, 132
    componentWillUnmount method, 132
    componentWillUpdate method, 131
    definition of, 123
    getDefaultProps method, 128

getInitialState method, 128
initial rendering phase, 127–129
overview, 123–124
render method, 129, 131
shouldComponentUpdate method, 130–131
unmounting phase, 132
updating phase, 129–132
LightningCounterDisplay
componentDidMount method, 93, 94–95
full code listing, 96–98
getInitialState method, 93–94
initial code, 90–92
initial state value, setting, 93–94
overview, 89–90
setInterval function, 93
setState method, 93, 95
state change, rendering, 96
strikes variable, 93–94
timerTick function, 94–95
Link, 156
multiple components, displaying, 103–105
PlusButton, 116–117
properties
specifying, 33–34
transferring. *See* transferring properties
Route, 149
Router, 149
Shirt, 74–77
Stuff, 157–158
styling with CSS, 41–42
styling with React
customizable background color, 45–47
overview, 42
px suffix, omitting, 45
style object, creating, 43
styles, applying, 43–45
TodoItems, 173–176
TodoList
addItem event handler, 171, 177–178
app functionality, 168
form submission, 169–170
functionality, 168
initial code listing, 164–165
overview, 163–164
render method, 166, 171, 173–174
state object initialization, 169
state object population, 171–172
styles, 167, 176–178
task display, 173–176
UI (user interface), 165–168
componentWillMount method, 129
componentWillReceiveProps method, 132
componentWillUnmount method, 132
componentWillUpdate method, 131
composability of components
advantages of, 66
color palette card example
Card component, 58–59
component definitions, 56–58
component identification, 53–56
generated HTML, 66
Label component, 61–63
overview, 49–51
properties, passing to child components, 63–65
Square component, 60–61
visual element identification, 51–53
definition of, 49
overview, 49–51

configuring development environment
index.jsx file, 191–192
initial project structure, 184–186
Node.js, 187–189
React dependencies, 190–191
webpack, 193–195
console warnings, 105
Contact component, 157–158
container elements, 18
CounterParent component
event handler, 110–112
event listening, 116–119
event properties, 114–115
initial code listing, 108–110
overview, 107–108
this keyword, 119–120
counters
button counter
event handler, 110–112
event listening, 116–119
event properties, 114–115
initial code listing, 108–110
lifecycle methods, 124–127
overview, 107–108
SyntheticEvent type, 112–114
this keyword, 119–120
LightningCounterDisplay
componentDidMount method, 93, 94–95
full code listing, 96–98
getInitialState method, 93–94
initial code, 90–92
initial state value, setting, 93–94
overview, 89–90
setInterval function, 93
setState method, 93, 95
state change, rendering, 96
strikes variable, 93–94
timerTick function, 94–95
createClass method, 30
createElement function, 82–84
CSS (Cascading Style Sheets)
applying, 40–42
first React app, 19–20
HTML versus JSX, 84–85
SPA (single-page app) example, 151–153
Todo List app, 167, 176–178
curly brackets ({ }), 33, 86–87
customizing background color, 45–47

D

DE. *See* development environment
deep links, 144
dependencies, installing, 190–191
design
multi-page design, 2–3
SPA (single-page app) model, 3–6
destination, changing, 18–19
destination variable, 18–19
dev folder, 184–185
development environment
advantages of, 179–181
Babel, 182
building apps, 197–198
code editors, 183–184
Node.js
installing, 187–189
overview, 182
setup
index.jsx file, 191–192
initial project structure, 184–186
Node.js, 187–189
React dependencies, 190–191

webpack, 193–195
webpack, 183
displaying
home page, 149–151, 154–155
multiple components, 103–105
name, 16–18
tasks in Todo List app, 173–176
div element
first React app, 18
HelloWorld component, 32
styling with CSS, 41–42
documentation
npm documentation, 198
React Router, 161
document.body argument (render method), 17
DOM elements, accessing
ES6 arrow functions, 141
references, 137–141
when to use, 135–137
DOM manipulation, 8

E

editors (code), 183–184
elements
capitalization in JSX, 87–88
div
first React app, 18
HelloWorld component, 32
styling with CSS, 41–42
DOM elements, accessing
ES6 arrow functions, 141
references, 137–141
when to use, 135–137
form, 169–170
inefficiencies with UI elements, 26–29
outputting multiple, 27
transferring properties with, 138–139
ellipses (…) operator
explained, 78
transferring properties with, 78–80
EmberJS templates, 10
environment. ***See*** **development environment**
errors in ranges, 105
ES6 arrow functions, 141
event handlers
addItem, 170
overview, 110–112
this keyword, 119–120
events
browser compatibility, 120
button counter
event handler, 110–112
event listening, 116–119
event properties, 114–115
initial code listing, 108–110
overview, 107–108
this keyword, 119–120
definition of, 107
event handlers
addItem, 170
overview, 110–112
this keyword, 119–120
KeyboardEvent type, 112
listening to, 116–119
MouseEvent type, 112
onSubmit, 169–170
performance, 120–121
SyntheticEvent type, 112–114

F

files, index.jsx, 191–192
first React app
blank HTML page, creating, 14–15

destination, changing, 18–19
name, displaying, 16–18
styles, 19–20
folders
dev, 184–185
node_modules, 190
output, 184–185
form submission, 169–170
formatDistance function, 25
forum.kirupa.com, 199
frames (app), 147–149
functions. *See also* methods
addEventListener, 118–119
calling, 25
createElement, 82–84
ES6 arrow functions, 141
explained, 24–26
formatDistance, 25
getDistance, 32–33
increase, 110–112, 115
printStuff, 78
removeEventListener, 119
setInterval, 93
showCircle, 102–103
timerTick, 94–95

G

generated HTML
color palette card, 66
Letter component, 40–41
getDefaultProps method, 128
getDistance function, 32–33
getInitialState method, 93–94, 128, 169

H

HelloWorld component
creating, 30–32
properties, 32–34
help, forum.kirupa.com, 199
hierarchies
component hierarchy, 55–56, 69–70
visual hierarchy, 52
Home component, 149–151
home page, displaying, 149–151, 154–155
HTML elements, capitalization in JSX, 87–88
HTML templates, 6

I

identifying
components, 53–56
visual elements, 51–53
increase function, 110–112, 115
index.html file, 185–186
index.jsx file, 191–192
initial project structure, 184–186
initial rendering phase (lifecycle methods), 127–129
initial state value, 93–94
initializing
Node.js, 187–189
state object, 169
inline styles
applying, 43–45
Circle component, 101
customizable background color, 45–47
overview, 42
px suffix, omitting, 45
style object, creating, 43
input element, 138–139
_input property, 139–140
installing
Babel, 195–196
Node.js, 187–189
React dependencies, 190–191
webpack, 193–195
itemArray, creating, 172

J

JavaScript
- JSX-to-JavaScript transformation. *See also* development environment
  - Babel, 182
  - overview, 81–83
  - test app, 197–198
  - webpack, 183, 193–196
- visuals defined in, 9–11

JSX
- arrays, 103–105
- capitalization, 87–88
- class names, 85–86
- comments, 86–87
- CSS (Cascading Style Sheets) and, 84–85
- explained, 10–11, 14–15
- first React app
  - blank HTML page, creating, 14–15
  - destination, changing, 18–19
  - name, displaying, 16–18
  - styles, 19–20
- index.jsx file, 191–192
- JSX-to-JavaScript transformation. *See also* development environment
  - Babel, 182
  - overview, 81–83
  - test app, 197–198
  - webpack, 183, 193–196
- location in code, 88
- methods. *See* methods
- reserved keywords, 85–86
- root nodes, returning, 83–84
- style attribute, 84–85

JSX-to-JavaScript transformation. *See also* development environment
- Babel, 182
- test app, 197–198
- webpack, 183, 193–196

K

KeyboardEvent type, 112

keywords
- table of, 85–86
- this, 119–120

L

Label component, 61–63, 77

Letter component
- generated HTML, 40–41
- overview, 37–39
- styling with CSS, 41–42
- styling with React
  - customizable background color, 45–47
  - letterStyle object, creating, 43
  - overview, 42
  - px suffix, omitting, 45
  - styles, applying, 43–45

letterStyle object
- creating, 43
- customizable background color, 45–47
- overview, 85
- px suffix, omitting, 45
- style attribute, 43–45

lifecycle methods
- button counter example, 124–127
- componentDidMount method, 129
- componentDidUpdate method, 131
- componentWillMount method, 129
- componentWillReceiveProps method, 132
- componentWillUnmount method, 132
- componentWillUpdate method, 131
- definition of, 123
- getDefaultProps method, 128
- getInitialState method, 128

initial rendering phase, 127–129
overview, 123–124
render method, 129, 131
shouldComponentUpdate method, 130–131
unmounting phase, 132
updating phase
prop changes, 131–132
state changes, 129–131
LightningCounterDisplay component
componentDidMount method, 93
full code listing, 96–98
getInitialState method, 93–94
initial code, 90–92
initial state value, setting, 93–94
overview, 89–90
setInterval function, 93
setState method, 93
state change, rendering, 96
strikes variable, 93–94
timerTick function, 94–95
Link component, 156
links
active links, 159–160
deep links, 144
navigation links, 155–156
listening to events
regular DOM events, 117–119
SyntheticEvent type, 116–117
listItems variable, 175–176
logo (React), 7

M

matching routes, 158
Matryoshka dolls analogy, 9
messages, console warnings, 105
methods. *See also* functions
addItem, 170, 171, 177–178
button counter example, 124–127
componentDidMount, 93, 94–95, 118, 129
componentDidUpdate, 131
componentWillMount, 129
componentWillReceiveProps, 132
componentWillUnmount, 132
componentWillUpdate, 131
createClass, 30
definition of, 123
getDefaultProps, 128
getDefaultProps method, 128
getInitialState, 93–94, 128, 169
initial rendering phase, 127–129
overview, 123–124
preventDefault, 172
render
Card component, 59
Circle component, 102, 104–106
Colorizer component, 138
first React app, 16–18
HelloWorld component, 30–32
initial rendering phase, 129
Label component, 62
LightningCounterDisplay component, 92
overview, 26, 129, 131
SPA (single-page app) example, 148–149, 154
Square component, 60–61
Todo List app, 166
TodoList component, 171, 173–174
updating phase, 131
setNewColor, 141
setState, 93, 95
shouldComponentUpdate, 130–131
unmounting phase, 132

updating phase, 129–132

MouseEvent type, 112

multi-page design, 2–3

multiple components, displaying, 103–105

multiple elements, outputting, 27

MVC architecture, 11

MyTotallyAwesomeApp

index.jsx file, 191–192

initial project structure, 184–186

Node.js initialization, 187–189

React dependencies, 190–191

testing, 197–198

N

names

class names, 85–86

displaying, 16–18

navigation links, creating, 155–156

Node Package Manager (NPM). *See* Node.js

node_modules folder, 190

Node.js

initializing, 187–189

installing, 187–189

overview, 182

nodes (root), returning, 83–84

NPM (Node Package Manager). *See* Node.js

npm documentation, 198

O

objects

circleStyle object, 101

letterStyle

creating, 43

customizable background color, 45–47

overview, 85

px suffix, omitting, 45

style attribute, 43–45

props, 79

state object

initializing, 169

populating, 171–172

onClick event handler, 110–112

onSubmit event, 169–170

operators, spread (…)

explained, 78

transferring properties with, 78–80

output folder, 184–185

outputting multiple elements, 27

P

palette card. *See* color palette card

passing properties

color palette card example, 63–65

component hierarchy and, 69–74

problems with, 74–77

spread operator (…)

example, 78–80

explained, 78

performance, events and, 120–121

PlusButton component, 116–117

populating state object, 171–172

preventDefault method, 172

printStuff function, 78

properties

accessing, 33

activeClassName, 159–161

bgcolor, 46

event properties, 114–115

HelloWorld component, 32–34

prop changes, 131–132

ref, 138–139
specifying
component call, 34
component definition, 33
style, 43–45, 84–85
SyntheticEvent type, 113–114
transferring
color palette card example, 63–65
component hierarchy and, 69–74
problems with, 74–77
spread operator (...), 78–80
vendor-prefixed properties, 59
props object, 79
px suffix, omitting, 45

Q-R

React dependencies, installing, 190–191
React Event System document, 114
React Router
creating SPAs (single-page apps) with
active links, 159–160
app frame, 147–149
boilerplate markup and code, 146–147
Contact component, 157–158
CSS (Cascading Style Sheets), 151–153
displaying, 149–151
example, 144–145
Home component, 149–151
home page, displaying, 154–155
Link component, 156
navigation links, 155–156
overview, 143–144
ReactRouter prefix, 153–154
render method, 148–149, 154
Route component, 149
route matching, 158
Router component, 149
Stuff component, 157–158
documentation, 161
referencing, 147
React Tooling Integration, 198
ReactRouter prefix, 153–154
reconciliation, 8
ref attribute, 138–139
references, 137–141
referencing
Babel JavaScript compiler, 16
React library, 15
React Router, 147
regular DOM events, listening to, 117–119
removeEventListener function, 119
render method
Card component, 59
Circle component, 102, 104–106
Colorizer component, 138
first React app, 16–18
HelloWorld component, 30–32
initial rendering phase, 129
Label component, 62
LightningCounterDisplay component, 92
overview, 26
SPA (single-page app) example, 148–149, 154
Square component, 60–61
Todo List app, 166
TodoList component, 171, 173–174
updating phase, 131
renderData array, 104
rendering state change, 96
reserved keywords, 85–86
root nodes, returning, 83–84

Route component, 149
route matching, 158
Router component, 149
routing, 144, 149. *See also* React Router
Russian Matryoshka dolls analogy, 9

S

setInterval function, 93
setNewColor method, 141
setState method, 93, 95
setting up development environment
  index.jsx file, 191–192
  initial project structure, 184–186
  Node.js, 187–189
  React dependencies, 190–191
  webpack, 193–195
shiftKey property (SyntheticEvent), 114–115
Shirt component, 74–77
shouldComponentUpdate method, 130–131
showCircle function, 102–103
simple catalog browser app
  multi-page design, 2–3
  SPA (single-page app) model, 3–6
single-page app (SPA). *See* SPA (single-page app)
SPA (single-page app)
  creating with React Router
    active links, 159–160
    app frame, 147–149
    boilerplate markup and code, 146–147
    Contact component, 157–158
    CSS (Cascading Style Sheets), 151–153
    example, 144–145
    Home component, 149–151
    home page, displaying, 149–151, 154–155
    Link component, 156
    navigation links, 155–156
    overview, 143–144
    ReactRouter prefix, 153–154
    render method, 148–149, 154
    Route component, 149
    route matching, 158
    Router component, 149
    Stuff component, 157–158
  model, 3–6
specifying properties
  component call, 34
  component definition, 33
spread operator (...)
  explained, 78
  transferring properties with, 78–80
Square component, 60–61
state change, rendering, 96
state management
  lifecycle methods, 129–131
  LightningCounterDisplay
    componentDidMount method, 93, 94–95
    getInitialState method, 93
    initial code, 90–92
    initial state value, setting, 93–94
    overview, 89–90
    setInterval function, 93
    setState method, 93, 95
    strikes variable, 93–94
  LightningCounterDisplay component
    full code listing, 96–98
    initial code, 90–92
    overview, 89–90
    setInterval function, 93
    state change, rendering, 96

timerTick function, 94–95
UI (user interface), 7–8
state object. ***See also*** **state management**
initializing, 169
populating, 171–172
strikes variable, 93–94
Stuff component, 157–158
style attribute, 43–45, 84–85
styles
CSS (Cascading Style Sheets)
applying, 40–42
HTML versus JSX, 84–85
SPA (single-page app) example, 151–153
Todo List app, 167, 176–178
first React app, 19–20
inline approach
Circle component, 101
customizable background color, 45–47
overview, 42
px suffix, omitting, 45
style object, creating, 43
styles, applying, 43–45
overview, 37
submitting forms, 169–170
support, forum.kirupa.com, 199
swatchComponent variable, 88
SyntheticEvent type, 112–114

T

tasks, displaying in Todo List app, 173–176
templates
EmberJS templates, 10
HTML templates, 6
theCircle variable, 102
this keyword, 119–120
timerTick function, 94–95
todoEntries variable, 174
TodoItems component, 173–176
TodoList component
addItem event handler, 171, 177–178
app functionality, 168
form submission, 169–170
functionality, 168
initial code listing, 164–165
overview, 163–164
render method, 166, 171, 173–174
state object initialization, 169
state object population, 171–172
styles, 167, 176–178
task display, 173–176
TodoItems component, 173–176
UI (user interface), 165–168
tools
Babel, 182
code editors, 183–184
Node.js
installing, 187–189
overview, 182
webpack
installing, 193–195
overview, 183
transferring properties
color palette card example, 63–65
component hierarchy and, 69–74
problems with, 74–77
spread operator (…)
example, 78–80
explained, 78
transpilation from JSX to JavaScript. ***See also*** **development environment**
Babel, 182
overview, 81–83

test app, 197–198
webpack, 183, 193–196

U

UI (user interface). ***See also*** **events**
Circle component example
array of, 103–105
initial code listing, 99–102
render method, 102
showCircle function, 102–103
theCircle variable, 102
inefficiencies with, 26–29
state management, 7–8
Todo List app, 165–168
unmounting phase (lifecycle methods), 132
updating phase (lifecycle methods)
prop changes, 131–132
state changes, 129–131

V

variables
destination, 18–19
listItems, 175–176
strikes, 93–94
swatchComponent, 88
theCircle, 102
todoEntries, 174
vendor-prefixed properties, 59
views, 143
virtual DOM, 8
visual hierarchy, 52
visuals
defining in JavaScript, 9–11
identifying, 51–53
visual hierarchy, 52

W-X-Y-Z

WebkitFilter property, 59
webpack
installing, 193–195
overview, 183
website, 198