

BRANDON RICHEY | RYAN YU
ENDRE VEGH | THEOFANIS DESPOUDIS
ANTON PUNITH | FLORIAN SLOOT

THE REACT WORKSHOP

GET STARTED WITH BUILDING WEB
APPLICATIONS USING PRACTICAL TIPS
AND EXAMPLES FROM REACT USE CASES

Packt>

WEB DEVELOPMENT

THE REACT WORKSHOP

Get started with building web applications using practical tips and examples from React use cases

Brandon Richey, Ryan Yu, Endre Vegh, Theofanis Despoudis,
Anton Punith, and Florian Sloot

Packt>

THE REACT WORKSHOP

Copyright © 2020 Packt Publishing

All rights reserved. No part of this course may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this course to ensure the accuracy of the information presented. However, the information contained in this course is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this course.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this course by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Authors: Brandon Richey, Ryan Yu, Endre Vegh, Theofanis Despoudis, Anton Punith, and Florian Slood

Reviewers: Souvik Basu, Daniel Bugl, Brandan Jones, SudarshanReddy Kurri, David Parker, and Cihan Yakar

Managing Editor: Ranu Kundu

Acquisitions Editors: Anindya Sil, Sneha Shinde, Alicia Wooding, Archie Vankar, and Karan Wadekar

Production Editor: Shantanu Zagade

Editorial Board: Megan Carlisle, Samuel Christa, Mahesh Dhyani, Heather Gopsill, Manasa Kumar, Alex Mazonowicz, Monesh Mirpuri, Bridget Neale, Dominic Pereira, Shiny Poojary, Abhishek Rane, Brendan Rodrigues, Erol Staveley, Ankita Thakur, Nitesh Thakur, and Jonathan Wray

First published: August 2020

Production reference: 1210820

ISBN: 978-1-83864-556-4

Published by Packt Publishing Ltd.

Livery Place, 35 Livery Street

Birmingham B3 2PB, UK

Table of Contents

Preface	i
<hr/>	
Chapter 1: Getting Started with React	1
<hr/>	
Introduction	2
Problems before React	2
Introducing React	5
Getting Started with Create React App	6
Setting Up a Development Environment	7
Tweaking Create React App's Options	9
Using Create React App to Create Our Project	10
Exploring the Created Project	13
Exploring the Rest of the Scaffold	17
Introduction to JSX	18
Diving Deeper into JSX	19
Exercise 1.01: Implementing JSX	21
Creating a React Component	23
Preparing Our Code to Start Writing Our Component	23
Understanding Props	24
Exercise 1.02: Creating Our First React Component	26
Understanding React Rendering	28
Building Components with Classes	30
Exercise 1.03: Implementing a Click Counter in React Using Classes ..	30
Activity 1.01: Design a React Component	34
Summary	36

Chapter 2: Dealing with React Events **39**

Introduction – Talking to JavaScript with React	40
Designing the State of Our Application	41
Quick UI Examples	42
Getting Started – Building Our Baseline Component	45
Exercise 2.01: Writing the Starter State for Our Code	47
Event Handlers in React	49
onClick	50
Exercise 2.02: Writing Our First Event Handler	50
onBlur	53
Context of Event Handlers	55
In-line Bind Statements	57
Constructor Bind Statements	58
Exercise 2.03: Building our Username Validator	59
Exercise 2.04: Using Alternative Class Declarations to Avoid Binds	61
Form Validation in React	64
Exercise 2.05: Creating a Validation for Input Fields	65
Activity 2.01: Create a Blog Post Using React Event Handlers	70
Summary	71

Chapter 3: Conditional Rendering and for Loops **73**

Introduction	74
Conditional Rendering	74
Exercise 3.01: Building Our First App Using Conditional Rendering	76
Nested Conditional Rendering	82
Exercise 3.02: Building a Conditional Quiz App	83
Rendering Loops with React	92

Rendering Loops	94
Exercise 3.03: Refining Our Quiz Display with Loops	96
Activity 3.01: Building a Board Game Using Conditional Rendering ...	99
Summary	102
Chapter 4: React Lifecycle Methods	105
Introduction	106
Overview of the Component Lifecycle	106
Exercise 4.01: Implementing Lifecycle Methods	109
The Mount Lifecycle	112
constructor()	113
Exercise 4.02: Conditional Rendering and the Mount Lifecycle	113
render()	116
componentDidMount()	117
The Update Lifecycle	118
render()	118
componentDidUpdate()	118
Exercise 4.03: Simulating a Slow AJAX Request and Prerendering Your Component	119
The Unmount Lifecycle	123
componentWillUnmount()	124
Exercise 4.04: Messaging on Unmount	125
Activity 4.01: Creating a Messaging App	129
Summary	131

Chapter 5: Class and Function Components	133
Introduction	134
Introduction to Thinking with React Components	134
Atomic Design	134
Pragmatic Approach	136
Building React Components	139
Data Flow	140
State and Props	140
Class Components	140
Exercise 5.01: Creating Profile Component as a Class Component ...	143
Function Component	151
Exercise 5.02: Changing the Loader Component as a Function Component	152
Differences between Class and Function Components	159
Syntax	159
Handling State	161
Activity 5.01: Creating a Comments Section	163
Summary	166
Chapter 6: State and Props	169
Introduction	170
State in React	170
Initializing and Using State	171
Setting State	171
setState	172
Custom Methods and Closure	172
Exercise 6.01: Building an App to Change the Theme	173

Props in React	178
Children Prop	179
Props are Immutable	179
Exercise 6.02: Creating a Modal Screen	180
Activity 6.01: Creating a Products App Page	191
Summary	193
Chapter 7: Communication between Components	195
Introduction	196
Getting Started	198
Passing Data from Parent to Child Components	200
Passing Data to Direct Child Components	201
Example 1: Sending Data from a Parent Component to a Direct Child Component	201
Example 2: Receiving Data in a Child Class Component	202
Example 3: Receiving Data in a Child Function Component	204
Example 4: Sending number and Boolean as Props from the Parent Component	205
Example 5: Receiving number and boolean Values in Class-Based and Functional Components	206
Destructuring Props	208
Example 6: Destructuring Prop Values in a Child Class Component	209
Example 7: Destructuring Prop Values in a Function Component	211
Exercise 7.01: Sending and Receiving Objects as Props from the Parent	212
The {children} Prop	215
Exercise 7.02: Sending Child Elements Using the children Prop	216
Sending and Receiving an Array through Props	217

Exercise 7.03: Sending, Receiving, and Displaying an Array from a Parent to a Child	221
Passing Data to a Child Component Multiple Levels Down	226
Splitting a Component into Smaller Components.....	228
Exercise 7.04: Splitting into Smaller Components	229
Passing a Component through a Prop	230
Exercise 7.05: Creating a Photo Function Component	232
Higher-Order Components	235
Exercise 7.06: Creating a HOC Function That Can Be Called with DonationColor	237
Render Props	241
Exercise 7.07: Adding donationColor	241
Passing Data from Children to a Parent	247
Exercise 7.08: Passing Data from a Child to a Parent Component	249
Passing Data Between Components at Any Level	253
Exercise 7.09: Adding the addList Callback Function	256
The Context API	262
Exercise 7.10: Creating the <AnimalCount> Component Using the React Context API	265
Activity 7.01: Creating a Temperature Converter	269
Summary	272
Chapter 8: Introduction to Formik	275
Introduction	276
Uncontrolled Components	276
Exercise 8.01: Creating Our First Uncontrolled Form Component	278
Controlled Components	281

Exercise 8.02: Converting Our Form Component from Uncontrolled to Controlled	282
Introduction to Formik	285
Advantages of Formik	286
Anatomy of a Formik Component	286
Initial Values and Handlers	290
Formik Higher-Order Components	292
Connect	294
Validating a Form	294
Exercise 8.03: Adding Field Validators	295
Controlling When Formik Runs Validation Rules	297
Schema Validation	298
Exercise 8.04: Controlling Schema Validation Phases	300
Submitting a Form	303
Activity 8.01: Writing Your Own Form Using Formik	305
Summary	307
Chapter 9: Introduction to React Router	309
Introduction	310
Understanding Browser Routing	310
Exercise 9.01: Building a Basic Router using the Location API	311
Basics of React Router	316
Exercise 9.02: Implementing a Switch Router	317
Adding Params to Your Routes	320
Exercise 9.03: Adding Params to Routes	321
Adding Page Not Found for Routes	325
Exercise 9.04: Adding a NotFound Route	326

Nesting Routes	328
Exercise 9.05: Nested Routes and the Link Component	329
Activity 9.01: Creating an E-Commerce Application	335
Summary	337

Chapter 10: Advanced Routing Techniques: Special Cases 339

Introduction	340
React Router Special Cases	340
Passing URL Parameters between Routes	340
Exercise 10.01: Creating URL Parameters	341
Matching Unknown Routes with 404 Pages	345
Exercise 10.02: Creating Unknown Routes	345
Rendering Nested Routes	348
Exercise 10.03: Creating Nested Routes	350
Protecting Routes	354
Preventing OutBound Transitions	354
Exercise 10.04: Protected Routes	355
Preventing Inbound Transitions	359
Activity 10.01: Creating Authentication Using Routing Techniques ..	360
Summary	363

Chapter 11: Hooks – Reusability, Readability, and a Different Mental Model 365

Introduction	366
Hooks	366
useState	367

Exercise 11.01: Displaying an Image with the Toggle Button	368
useEffect – from Life Cycle Methods to Effect Hooks	372
Exercise 11.02: Creating a Login State Using useEffect	376
Comparing useEffect Hooks with Life Cycle Methods	382
Comparing Hooks to Render Props	384
Activity 11.01: Creating a Reusable Counter	386
Summary	387
Chapter 12: State Management with Hooks	389
Introduction	390
useState Hook: A Closer Look	390
Setter Functions on Arrays	391
Exercise 12.01: Array Manipulation Using Hooks	392
Equality and Immutability for Objects in React	396
Limitations of useState	396
Using the useReducer Hook	396
Reducer Function in React	398
Exercise 12.02: useReducer with a Simple Form	400
Effects with Cleanup	407
Activity 12.01: Creating a Chat App Using Hooks	409
Summary	411
Chapter 13: Composing Hooks to Solve Complex Problems	413
Introduction	414
Context API and Hooks	414
useContext Hook	415

Exercise 13.01: Adding Context to the ShoppingCart App	416
Props and Context	424
Props for Customization	425
Another Example: Theming as a Service	426
Exercise 13.02: Creating Context Using useContext and useEffect ...	428
Activity 13.01: Creating a Recipe App	434
Summary	436
Chapter 14: Fetching Data by Making API Requests	439
Introduction	440
RESTful API	440
Five Common HTTP Methods	442
PUT versus PATCH	442
HTTP Status Codes	444
Accept Header and Content-Type Header	444
Different Ways of Requesting Data	447
XMLHttpRequest	448
Exercise 14.01: Requesting Data Using XMLHttpRequest	450
Fetch API	452
Exercise 14.02: Requesting Data Using the Fetch API	454
Axios	455
Exercise 14.03: Requesting Data Using Axios	457
Comparison of XMLHttpRequest, the Fetch API, and Axios	459
Axios versus the Fetch API	463
Better Response Handling.....	463
Better Error Handling.....	463
Testing APIs with Postman	465

Exercise 14.04: GET and POST Requests with Postman	467
Exercise 14.05: PUT, PATCH, and DELETE Requests with Postman	471
Making API Requests in React	473
React Boilerplate and Axios	473
Exercise 14.06: Installing React Boilerplate and Axios	474
Testing the NASA API with Postman	475
Exercise 14.07: Testing the Endpoint with Postman	476
Fetching Data with React	477
Exercise 14.08: Creating a Controlled Component to Fetch Data	477
Activity 14.01: Building an App to Request Data from Unsplash	483
Summary	485
Chapter 15: Promise API and async/await	487
Introduction	488
What Is the Promise API?	488
Exercise 15.01: Fetching Data through Promises	491
What Is async/await?	499
async	500
await	500
then()	501
error()	501
finally()	502
Better Error Handling with async/await	502
Exercise 15.02: Converting submitForm() to async/await	504
async/await within Loops	508
for...of	509
forEach()	510

map()	510
Activity 15.01: Creating a Movie App	511
Summary	514

Chapter 16: Fetching Data on Initial Render and Refactoring with Hooks 517

Introduction	518
Fetching Data upon Initial Rendering	518
Exercise 16.01: Fetching Popular Google Fonts on Initial Rendering	522
Fetching Data on Update	529
Infinite Loop	532
Exercise 16.02: Fetching Trending Google Fonts	533
React Hooks to Fetch Data	539
Exercise 16.03: Refactoring the FontList Component	541
More Refactoring with Custom Hook	545
Exercise 16.04: Refactoring a FontList Component with a Custom Hook	546
Activity 16.01: Creating an App Using Potter API	551
Summary	554

Chapter 17: Refs in React 557

Introduction	558
Why React Refs?	558
References	558
Exercise 17.01: Creating Custom Upload Buttons with Refs	559
Ways of Creating React Refs	563

Exercise 17.02: Measuring the Dimensions of a div Element in a Class-Based Component	566
Exercise 17.03: Measuring the Element Size in a Functional Component	569
Forwarding Refs	571
Exercise 17.04: Measuring Button Dimensions Using a Forwarded Ref	575
Activity 17.01: Creating a Form with an Autofocus Input Element	579
Summary	581
Chapter 18: Practical Use Cases of Refs	583
Introduction	584
Recap of React Refs Basics	584
Encapsulation via Props	586
DOM Manipulation Helpers	587
The cloneElement function in React	587
Exercise 18.01: Cloning an Element and Passing It an onClick Prop	589
The createPortal function in ReactDOM	592
Exercise 18.02: Creating a Global Overlay Using Portals	595
Activity 18.01: Portable Modals Using Refs	599
Summary	602
Appendix	605
Index	775

PREFACE

ABOUT THE BOOK

You already know you want to learn React, and the smartest way to learn React is to learn by doing. *The React Workshop* focuses on building up your practical skills so that you can create dynamic, component-based web applications and interactive React UIs that provide exceptional user experiences. You'll learn from real examples that lead to real results.

Throughout *The React Workshop* book, you'll take an engaging step-by-step approach to understand React. You won't have to sit through any unnecessary theory. If you're short on time, you can jump into a single exercise each day or spend an entire weekend learning about two-way data binding. It's your choice. Learning on your terms, you'll build up and reinforce key skills in a way that feels rewarding.

Every physical print copy of *The React Workshop* unlocks access to the interactive edition. With videos detailing all exercises and activities, you'll always have a guided solution. You can also benchmark yourself against assessments, track progress, and receive content updates. You'll even earn a secure credential that you can share and verify online upon completion. It's a premium learning experience that's included with your printed copy. To redeem, follow the instructions located at the start of your React book.

Fast-paced and direct, *The React Workshop* is ideal for React beginners. You'll build and iterate on your code like a software developer, learning along the way. This process means that you'll find that your new skills stick, embedded as best practice, a solid foundation for the years ahead.

AUDIENCE

The React Workshop is for web developers and programmers who are looking to learn React and use it for creating and enhancing web applications. Although the workshop is for beginners, prior knowledge of JavaScript programming and HTML and CSS is necessary to easily understand the concepts that are covered in this book.

ABOUT THE CHAPTERS

Chapter 1, Getting Started with React, gets you typing code immediately. You will learn the basics of React, as well as how to install and configure React projects with Create React App.

Chapter 2, Dealing with React Events, provides the starting point for creating interactive web apps with React by introducing events. We'll not only create events but integrate them into our React components.

Chapter 3, Conditional Rendering and for Loops, is where we expand upon programmatically creating React components either when conditions are set or when we need to add multiple React components as a list of items.

Chapter 4, React Lifecycle Methods, is where we go from passively relying on React to build our components to hooking up our components to the different lifecycle methods, allowing us to determine what logic to execute at each stage of our React component's life.

Chapter 5, Class and Function Components, discusses industry best practices to identify the component hierarchy and break the UI down into logical components. This chapter forms the basis for your understanding of creating UIs in React, be they simple or complex, and provides you with the basic tools required to build React applications.

Chapter 6, State and Props, discusses the components that use state and props. You will learn how to handle state in a React application and how to change state variables according to requirements.

Chapter 7, Communication between Components, helps you understand how to pass data between React classes and functional components. Also, this chapter dives into more advanced patterns such as higher-order components, render props, and the Context API.

Chapter 8, Introduction to Formik, gives you a complete picture of using Formik to build declarative forms. There are many ways to handle forms in React but Formik combines the best approaches.

Chapter 9, Introduction to React Router, gets you comfortable with one of the most commonly used React libraries in modern React web apps: React Router. We will cover the basics of using React Router, including implementing our own version to understand what is happening under the hood.

Chapter 10, Advanced Routing Techniques: Special Cases, looks at advanced techniques with React Router and gets you onto the next level. You will learn how to catch missing routes, how to nest routes, and how to protect routes from unauthorized accesses.

Chapter 11, Hooks – Reusability, Readability, and a Different Mental Model, prepares you for using the latest addition to React: hooks. We'll explore how to make your code more readable and more reusable via hooks.

Chapter 12, State Management with Hooks, builds on your knowledge of hooks to completely replace class-based component state management by exploring more advanced topics such as building your own state hooks.

Chapter 13, Composing Hooks to Solve Complex Problems, brings you from novice React hooks knowledge to expert React hooks knowledge, putting context hooks into the mix to completely eliminate the need for class-based components.

Chapter 14, Fetching Data by Making API Requests, shows you the various ways to fetch data by making API requests in React. This chapter also covers fetching data from servers using RESTful APIs, the Fetch API, and Axios and compares these methods.

Chapter 15, Promise API and `async/await`, takes a deep dive into the Promise API and `async/await`, which are essential techniques and the modern way to fetch data from the server.

Chapter 16, Fetching Data on Initial Render and Refactoring with Hooks, shows you the techniques of fetching data on initial render and fixing issues when a component falls into an infinite loop.

Chapter 17, Refs in React, introduces you to how to use references in React. You will be able to apply the knowledge gained from the chapter to implement references in different ways.

Chapter 18, Practical Use Cases of Refs, introduces you to some use cases of references and how to leverage their functionalities in your code. You will be able to identify scenarios in which to use references so that you can manipulate DOM elements directly.

CONVENTIONS

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows:

"The Promise API also comes with several methods, such as `then()`, `catch()`, and `finally()`."

Words that you see on the screen, for example, in menus or dialog boxes, also appear in the text like this: "Let's search for **London**, and the **temperature** should be displayed."

A block of code is set as follows:

```
"eslintConfig": {  
  "extends": "react-app"  
},
```

New terms and important words are shown like this: "Installing Node.js will also give us npm (the Node Package Manager)."

"**Callback hell** is an anti-pattern that consists of multiple nested callbacks".

Long code snippets are truncated and the corresponding names of the code files on GitHub are placed at the top of the truncated code. The permalinks to the entire code are placed below the code snippet. It should look as follows:

App.js

```
17 <div>  
18   Username: <input type="text" /><br />  
19   Password: <input type="text" /><br />  
20   Password Confirmation: <input type="text" /><br />  
21   Email: <input type="text" /><br />  
22 </div>  
23 <button>Submit</button>  
24 </div>
```

The complete example code is available at <https://packt.live/35qT2L3>.

BEFORE YOU BEGIN

You are here and you are ready to get started with building your web applications in React. We need to install Node.js and Node Package Manager (NPM) to get started, and then additionally include installations for Yarn and Create React App. The installation instructions work for Windows, Linux, and Mac operating systems.

INSTALLATION INSTRUCTIONS

NODE.JS/NPM

To install Node.js/NPM, you will need to do the following:

1. Visit <https://nodejs.org> and visit the **Download** page.
2. From there, choose the appropriate package for your platform (or you can opt to download the source code and compile it yourself).

3. Verify the results. Open a Terminal or Command Prompt and verify that Node.js and NPM are installed with the **node -v** and **npm -v** commands, which should return the version installed for each.

NOTE

For the purposes of this book, we will be working with the "current" version, not the "LTS" version.

YARN

To install Yarn:

1. With Node.js and NPM installed, you can now visit **yarnpkg.com** and navigate to the **Installation** section.
2. From there, choose the operating system you're using and the version (the latest stable version will work just fine for us).
3. Follow the instructions on the page and verify the results by typing **yarn -v** in your Command Prompt/Terminal window.

CREATE REACT APP

To install **Create React App**:

1. We will be working with the **npm create-react-app** command, which does not require us to install **create-react-app** locally. Should you want to, however, you can install it locally either via Yarn or NPM.

For NPM:

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

For Yarn:

```
$ yarn add global create-react-app
```

After that, verify the installation is working by running **create-react-app -v**.

EDITORS/IDES

You can download and use Visual Studio Code as an editor (<https://code.visualstudio.com/>). Visual Studio Code is a lightweight yet highly efficient source code editor that runs on your desktop and is available for Windows, macOS, and Linux operating systems. It provides you with built-in support for JavaScript, TypeScript, and Node.js. It is an easy tool to help you get started coding in React.

INSTALLING THE CODE BUNDLE

Download the code files from GitHub at <https://packt.live/3cx4XdN>. Refer to these code files for the complete code bundle.

If you have any issues during installation (or have any other feedback) you can reach us at **`workshops@packt.com`**.

1

GETTING STARTED WITH REACT

OVERVIEW

This chapter will explain the context and the reasons for using React framework and Create React App by utilizing the different options and scripts available for Create React App. The Create React App CLI tool will enable you to bootstrap a React project quickly. You will learn how to build a React component entirely with functional and class syntax using basic JSX. You will also be able to use components together to better structure your code. In this chapter, you will learn about the context and history of the React JS framework and be able to bootstrap your first React project.

INTRODUCTION

React is a frontend JavaScript framework. Its design principles take into consideration a lot of the common problems that developers used to face while building the user-facing components in their web projects. React doesn't exist to fix server problems or database problems, but rather to fix the common design and development problems of the frontend that exist in more complex and stateful web applications.

React is a framework that was created during the frenzy of frontend JavaScript frameworks. React by itself isn't just JavaScript that you can use to build elements on a page. After all, the problem of adding Document Object Model(DOM) elements to a web page has had solutions since the early days of Prototype and jQuery (or even further back than that). Instead, it is helpful to think of React as existing in a world that bridges the gap between JavaScript and HTML or, more specifically, the code and the browser.

Let's look at the problems that existed before the React JavaScript framework came into the picture.

PROBLEMS BEFORE REACT

Web development is a trade that can sometimes be incredibly confusing and complex. The act of getting the buttons, pictures, and text that we see on a website is not always a simple endeavor.

Think about a web page with a little form to log into your account. You have at least two text boxes, usually for your username and your password, where you need to enter your details. You have a button or link that allows you to log in after the details have been entered, with maybe at least one extra little link in case you forget your password. You probably also have a logo for this site you are logging into, and maybe some sort of branding or otherwise compelling visual elements. Here is an example of a simple login page:

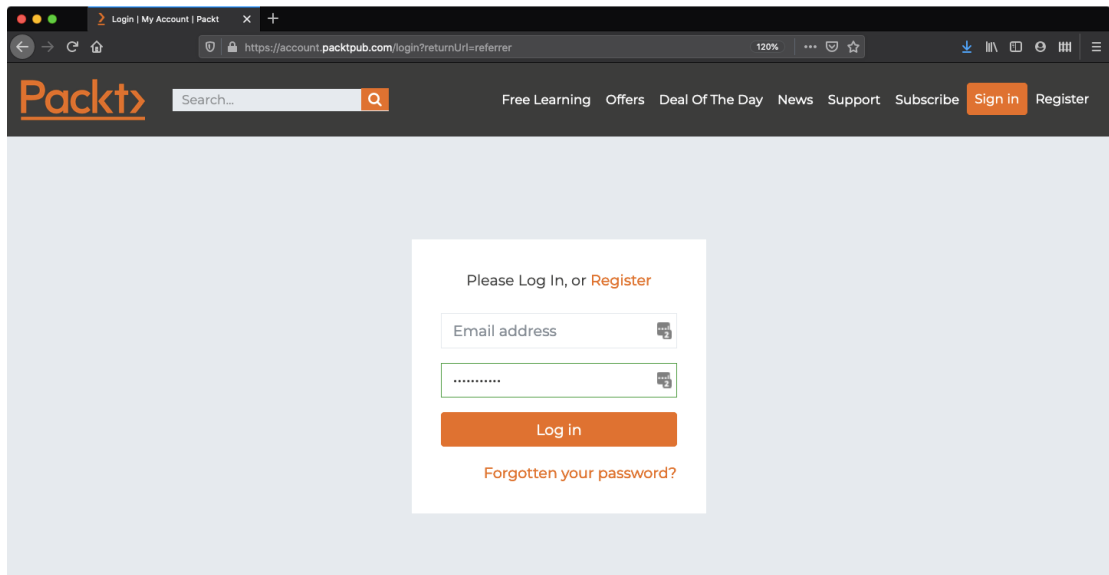
A screenshot of a web browser displaying the Packt login form. The browser's address bar shows the URL 'https://account.packtpub.com/login?returnUrl=referrer'. The Packt logo is in the top left, and a search bar is next to it. The top navigation bar includes links for 'Free Learning', 'Offers', 'Deal Of The Day', 'News', 'Support', 'Subscribe', 'Sign in', and 'Register'. The main content area features a white login form with the text 'Please Log In, or Register'. It contains two input fields: 'Email address' and a password field with masked characters. Below the password field is an orange 'Log in' button. At the bottom of the form is a link that says 'Forgotten your password?'. The browser's status bar at the bottom indicates a 120% zoom level.

Figure 1.1: Login form

All of that doesn't just exist, though. A web developer needs to sit down and build each of these pieces and put them together in such a way so that the audience visiting that site can interact with each component individually. The real problem here is that all of that can be incredibly complicated to put together. For example, you might have multiple form inputs that all interact with each other depending on the values you entered and selected. Each piece needs to be able to interact with the next, and there are multiple different types of complex interactions that need to occur that go beyond just something happening as a result of a button being clicked.

For example, let's go back to our Packt login form example and examine some of the intricacies of interaction that exist even in such a simple and commonplace element. We will start off with the simplest example:

- Data needs to be entered into the username field.
- Data needs to be entered into the password field.
- The user needs to click the login button.
- The user is now logged in.

That's enough to make this work, but it's not a great user experience, is it? There's no validation, and nothing to help the user to get through the process smoothly or gracefully to let them know beforehand when the form fails to show the desired output.

So, we need to expand with additional use cases:

- If the user's username and password do not work, the user should see a failed login message.
- If the user fails to enter a username, the user should see a message reminding them to fill in the username field.
- If the user fails to enter a password, the user should see a message reminding them to fill in the password field.

These cases increase the complexity of the code pretty significantly.

Additionally, a few more elements for this web page are necessary:

- A message box displaying information if a username/password combination is incorrect, which evidently requires interaction with the server
- A message box displaying information if either of the username/password fields is left blank

The web page will get progressively more complicated as you move further along in improving the user experience.

For example, if we want to break it down further:

- What if we want to progressively check the username field to make sure it matches an email format if we are using emails for usernames?
- What if we want to validate whether each field has been filled (either for format or if the values have been filled) as we move through each of the fields, displaying a red box around input fields that are blank or skipped as we move along?

At each step, we are introducing new and increasingly complex levels of interaction between different UI elements and the collective state of each component.

Now, let's break the state of each component into:

- Their visual state (how the field is displayed to the user)
- Their data state (what is entered into the field)

- Their state in relation to other UI elements (how the login form knows about the data state of the username and password fields)
- Their interaction state (can the button be clicked if either the username or password fields are blank?)

Suddenly, you will realize that a few lines of HTML just will not cut it anymore. Now you need more complex blocks of JavaScript code, with state management both for visual and data states for each component, and a way to tie them all together.

You need to represent all of it in a way that won't require it to be rewritten every time someone needs to implement a similar form, and in a way that does not make a developer groan every time they have to touch the code. Let's see where React fits into all of this and how it addresses this problem.

INTRODUCING REACT

While the example in the previous section is a complex problem, the good news is that with React, the solution isn't terribly complex.

Instead of thinking about each element your browser sees and displays for the user as separate from your HTML code and JavaScript code (and thus separate from the states that each exhibits at any given point in time), React allows you to think of all of the code as part of the same data structure, all intertwined and working together:

- **The component**
- **The state**
- **The display (or render)**

This reduces a lot of the complexity and overhead while trying to amalgamate component state and component display with a mixture of CSS and JavaScript along the way. React represents a more elegant way to allow the developer to think holistically about what is on the browser and what the user is interacting with, and how that state changes along the way.

By itself, all of that is a great help to us as developers but React also introduces a full scaffold for setting things up quickly and easily in a way that lets us get to development faster and fiddle less with configuration along the way.

React introduces a way for us to set up how to represent the states and display of components, but additional tools make the setup and configuration process even easier. We can further optimize the process for getting started by introducing a special command-line tool for React projects called Create React App.

This tool minimizes the amount of time a developer needs to spend getting everything set up and allows developers to instead focus on the most important part of building your web application: **the development**.

GETTING STARTED WITH CREATE REACT APP

Create React App introduces a scaffold to React applications. Through this, configuration and setup are minimized around opinionated configuration (a set of configurations where a lot of the decisions have been made for you) and pre-built structures. Your directories, input, and output are all handled for you.

What is a scaffold? A scaffold in the development world is something that sets up boilerplate (that is, often-repeated bits of code or configuration) details for you with the idea that you will be using the same configuration, setup, and directory structure that the project is using. A scaffold allows you to get to development significantly faster, with the drawback that it may be harder to break away from decisions made in the scaffold later on in your development process, or that you may end up with more dependencies or structure than you need.

The Create React App scaffold is designed around the idea of the developer experience: that setting up a new project should be seamless and painless. If you want to start developing right now and not have to worry about how you are going to test your application, or how you are going to structure your application, or what libraries you are going to include in your application, then a scaffold such as Create React App is the way to go.

Try to think of a time that you had to do something that required a lot of setup (baking a cake, painting a picture, exercising, or whatever). Think of a scaffold you could use to bootstrap that process and speed it up for the next time that you want to repeat that activity. What would you do? What stuff would you include with it? Are there any parts of the process that might differ from person to person (these would indicate "opinionated" scaffolds or frameworks)?

Since we are trying to minimize friction at the starting point of our project, we are going to use this scaffold to get up and running quickly and painlessly.

SETTING UP A DEVELOPMENT ENVIRONMENT

Having digested the context of React, Create React App, and the problem at hand, it's time to start diving into getting everything ready for us to start writing our code. To do that, though, we will need to make sure that our development environment is ready for us to start using Create React App.

The first thing we will need to do is install Node.js (instructions can be found in the *Preface* of this book) to make sure we can use the command-line tool to create new React projects, and to make sure we can do so in a best-practices manner.

If you visit <https://packt.live/34FodRT>, you will see download links for your platform front and center, with a choice to either download the *LTS version* or the *current version*.

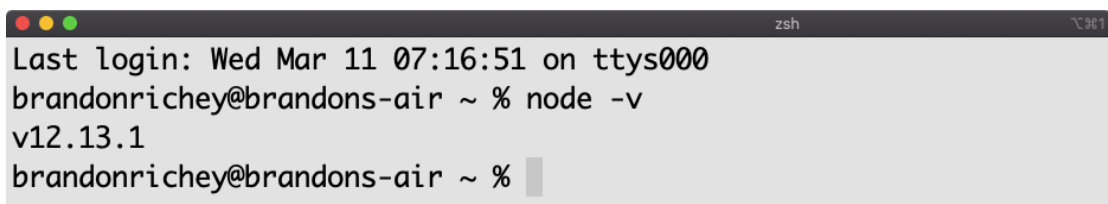
NOTE

LTS is the long-term support version of the Node.js runtime. This is designed more for things that are running on the server side in production environments and specifically need to not change over time, whether for security or stability reasons.

We will opt for the *current version* just to make sure we are using the latest and greatest for everything React along the way.

You will then need to follow the instructions to download and install Node.js via the installer. Each platform has a different way to get everything properly installed and set up, but for the most part, it's either a simple installer or a simple package-download to get started.

After getting everything installed, your next step should be to open a command-line/terminal window to verify that you can run Node.js. If you run the command **node -v**, you should see the version you just downloaded and installed:

A screenshot of a terminal window with a dark background. The title bar shows three colored window control buttons (red, yellow, green) on the left, the text 'zsh' in the center, and a window icon on the right. The terminal text shows a successful login, a command prompt, the execution of 'node -v', and the output 'v12.13.1'.

```
Last login: Wed Mar 11 07:16:51 on ttys000
brandonrichey@brandons-air ~ % node -v
v12.13.1
brandonrichey@brandons-air ~ %
```

Figure 1.2: Verifying whether Node.js is installed

Installing Node.js will also give us npm (the Node Package Manager) as well, which we will also want to verify with the command **npm -v**:

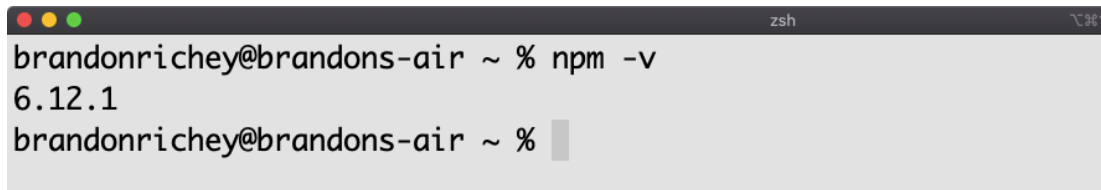
A terminal window with a dark background and light text. The prompt is 'brandonrichey@brandons-air ~ %'. The command 'npm -v' has been entered, and the output '6.12.1' is displayed on the next line. The prompt is now 'brandonrichey@brandons-air ~ %' with a cursor.

Figure 1.3: Verifying the version of Node.js

We can now start using another utility, **npx**, to execute Create React App and get started quickly.

NOTE

npx is a utility that executes **npm** package binaries (such as Create React App) with a minimal amount of fuss.

Instead of just running **create-react-app**, let's take a look at the help documentation instead with the **--help** flag in our call to **npx**. The output will be as shown below (the output will slightly vary for different operating systems):

```
$ npx create-react-app --help
Usage: create-react-app <project-directory> [options]
Options:
  -V, --version            output the version number
  --verbose                print additional logs
  --info                   print environment debug info
  --scripts-version <alternative-package> use a non-standard version of
react-scripts
  --use-npm
  --use-pnp
  --typescript
  -h, --help              output usage information
Only <project-directory> is required.
A custom --scripts-version can be one of the following:
  - a specific npm version: 0.8.2
  - a specific npm tag: @next
  - a custom fork published on npm: my-react-scripts
  - a local path relative to the current working directory: file:../
my-react-scripts
```

```
- a .tgz archive: https://mysite.com/my-react-scripts-0.8.2.tgz
- a .tar.gz archive: https://mysite.com/my-react-scripts-0.8.2.tar.gz
It is not needed unless you specifically want to use a fork. If you have
any problems, do not hesitate to file an issue: https://github.com/
facebook/create-react-app/issues/new
```

NOTE

Take a look at the output from the help page to figure out what you need to pass to **create-react-app**.

Let's check the React app version using the command **--version**:

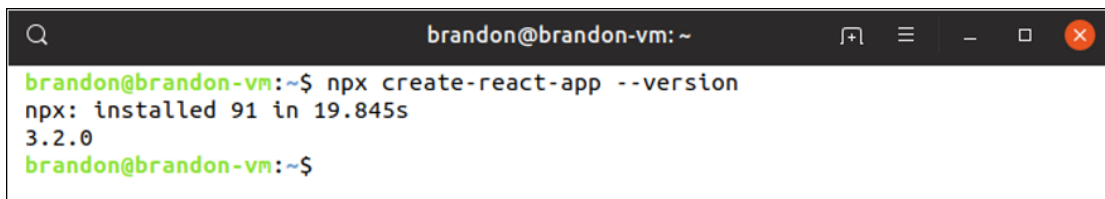
A terminal window titled 'brandon@brandon-vm:~' showing the command 'npx create-react-app --version' being executed. The output is 'npx: installed 91 in 19.845s' followed by '3.2.0' on a new line. The prompt returns to 'brandon@brandon-vm:~\$'.

Figure 1.4: Checking the Create React App version

In the next section, we will discuss how we can tweak the options in Create React App.

TWEAKING CREATE REACT APP'S OPTIONS

We need to dive a little bit deeper to better understand the different options that are available when we are using Create React App to generate our new React project structure. We can use it with no options and only declare the project name, but if we wanted to tweak our default configuration at all (for example, if we wanted to use a specific version of **react-scripts** or use something like TypeScript with our project), we would need to use one of the other options available to us.

We already talked about getting the version, but we can also pass some additional flags to change what we see or change how our project structure is generated along the way:

If we pass **--verbose** (by running **npx create-react-app --verbose**), for example, we will see a lot more information (and more detailed information) about each step along the way as the project is created and structured. You will want to use something like this if your project has the chance of running into issues during the creation step and you need to figure out, in greater detail, what's happening along the way. You can use this and **--info** to really understand all of the things that might be influencing the creation process if you are running into any issues.

We have the `--scripts-version` flag, which can tell Create React App to use a custom version of react-scripts, which is a helpful collection of scripts, dependencies, and utilities that facilitate the React app creation process. For the purposes of our first projects, however, we will leave this alone for now.

The `--use-npm` and `--use-pnp` flags exist because by default, Create React App will use **yarn** as its default package manager. If you pass in the `--use-npm` flag, Create React App will avoid the dependency on **yarn**, similar to the `--use-pnp` flag. Again, we are fine with the default settings here, so we will not be using either of those flags in our early projects.

Finally, the last non-help option we can pass is `--typescript`, which generates a Create React App scaffold with TypeScript hooked in appropriately (more on this in detail in the following chapters). You might use Typescript if you are looking to implement more development structure or stricter adherence to types in your React projects.

USING CREATE REACT APP TO CREATE OUR PROJECT

It's finally time to roll up our sleeves and get to work. You should have a very thorough understanding of Create React App's offerings, but now it's time to actually put that understanding to use.

Again, remember that Create React App creates a project for us, and that includes a lot of extras that act as the main scaffolding for our project. Much like a painter's scaffold, from far away it can seem like a lot of things have been added just to begin working, but the beauty is that it is all work that we would have to do anyway. This allows us to bypass that step and go right into getting some work done!

We will create our first project, which we will call **hello-react**, by running the following command:

```
$ npx create-react-app hello-react
```

The output will be as follows(truncated output):

```
npx: installed 91 in 23.129s
Creating a new React app in /home/brandon/code/hello-react.
Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts...
...
└─ react@16.10.2
└─ scheduler@0.16.2
Done in 151.99s.
```

```
Done in 49.76s.
Initialized a git repository.
Success! Created hello-react at /Users/brandon.richey/Development/react-
book-2/chapter1/code/hello-react
Inside that directory, you can run several commands:
yarn start
Starts the development server.
yarn build
Bundles the app into static files for production.
yarn test
Starts the test runner.
yarn eject
Removes this tool and copies build dependencies, configuration files and
scripts into the app directory. If you do this, you can't go back.
We suggest that you begin by typing:
cd hello-react
yarn start
Happy hacking!
```

This will give us our initial project, scaffolding and all, and allow us to jump right into coding.

Before we do that, we can see the output from our **npx create-react-app** command that tells us some of the other scripts we can run via **yarn**: **start**, **build**, **test**, and **eject**. We will be using **start**, **build**, and **test** the most, so we will focus on those and skip **eject** for now.

Each of the commands is broken down into a very specific and common scenario of application development and maintenance:

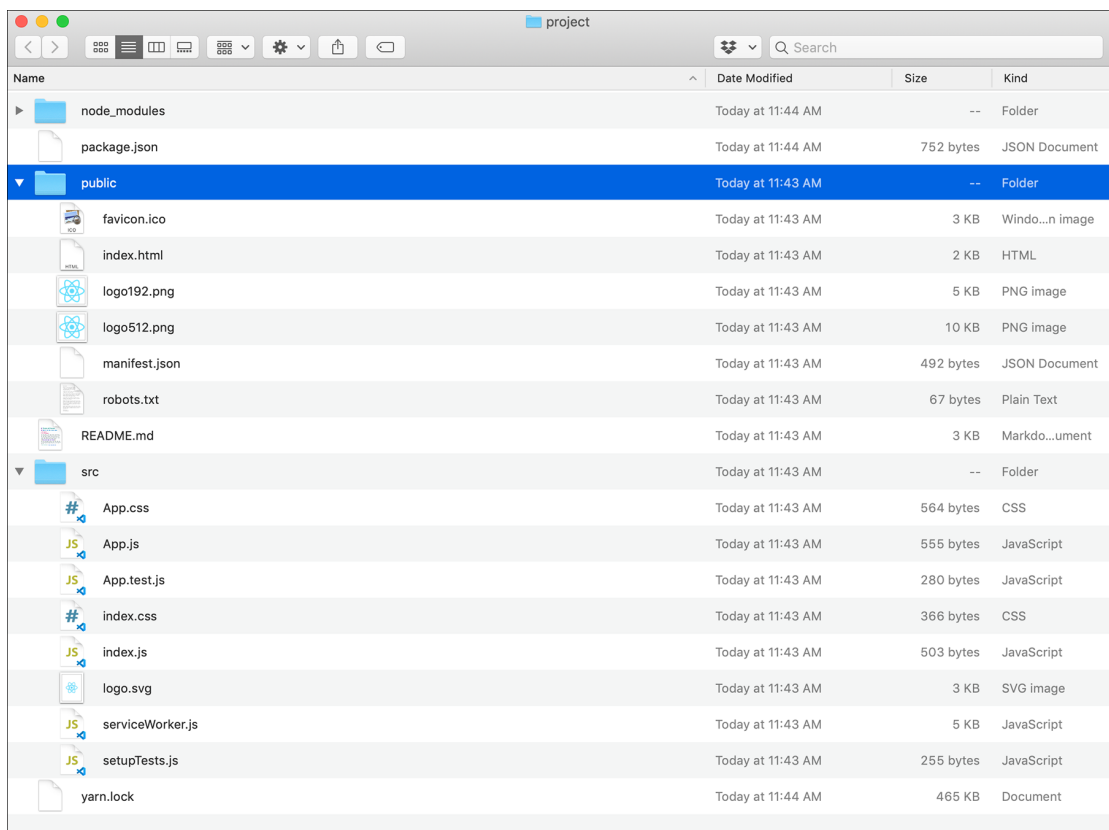
- **start** spins up a server listening locally on your computer on a specific port, where it watches for any changes made in the project, and then, when those changes are made and saved to the filesystem, it automatically reloads the appropriate project files. This gives you the ability to see changes that you make in your code as you save each file and make tweaks and modifications as necessary along the way. This is something that by itself could take a lot of time and effort to set up, so we are using Create React App just to avoid having to do this every time. Almost immediately, this makes the app scaffold worth it.

- Next, we have **build**, which is used to produce an optimized production build of our code. This takes all of the project's code (not just the JavaScript, but other file types, such as CSS) and optimizes as much as possible for use in a production environment. This includes minification (the process of stripping code files down to be as small as they possibly can be) and any other major optimizations that allow the code to download faster and execute quicker when it's out and serving real production traffic.
- Finally, of the most common commands, we have **test**. Now, **test** will run all of the tests for your code and display the results in an easy-to-read manner. It will also automatically run unit tests for each code change, including just the tests that failed when related code was modified, resulting in a broken test suite.
- **eject** is not often used but is invaluable when you start hitting certain limitations of the Create React App scaffold. **eject** is a command that pulls the covers off all of the configuration and scripts behind the scenes in your project, allowing you to edit and tweak any configuration detail that you will like. This also removes the safety net of the scaffolding, so it becomes easier to accidentally change something in your project and potentially break things or negatively impact the reliability of your production builds. Also, when you run this command, there is no turning back, so this is definitely something that you only want to do when you are good and ready and understand the potential impact on your project.

Now, let's start exploring the created project.

EXPLORING THE CREATED PROJECT

We have run the command to generate our project, but now it's time to take a deeper look and understand a little more of the scaffold and what we have been provided with. Here is an example of the files that are automatically generated in a new project:



Name	Date Modified	Size	Kind
node_modules	Today at 11:44 AM	--	Folder
package.json	Today at 11:44 AM	752 bytes	JSON Document
public	Today at 11:43 AM	--	Folder
favicon.ico	Today at 11:43 AM	3 KB	Windo...n image
index.html	Today at 11:43 AM	2 KB	HTML
logo192.png	Today at 11:43 AM	5 KB	PNG image
logo512.png	Today at 11:43 AM	10 KB	PNG image
manifest.json	Today at 11:43 AM	492 bytes	JSON Document
robots.txt	Today at 11:43 AM	67 bytes	Plain Text
README.md	Today at 11:43 AM	3 KB	Markdo...ument
src	Today at 11:43 AM	--	Folder
App.css	Today at 11:43 AM	564 bytes	CSS
App.js	Today at 11:43 AM	555 bytes	JavaScript
App.test.js	Today at 11:43 AM	280 bytes	JavaScript
index.css	Today at 11:43 AM	366 bytes	CSS
index.js	Today at 11:43 AM	503 bytes	JavaScript
logo.svg	Today at 11:43 AM	3 KB	SVG image
serviceWorker.js	Today at 11:43 AM	5 KB	JavaScript
setupTests.js	Today at 11:43 AM	255 bytes	JavaScript
yarn.lock	Today at 11:44 AM	465 KB	Document

Figure 1.5: Files generated in a new react project

We will start off by just exploring the top levels and then look more closely at any noteworthy individual files along the way. You'll find that even the starter project includes quite a few files, and while we will not go into extreme detail on each individual file, we will talk a little bit about how they all fit together overall.

README.md

The first notable file, right in the project root, is our **README.md** file. This is a *markdown* file that tells people how to use and interact with your project. Usually, this should include instructions, a brief synopsis of the project, how to start the project, how to run the tests, and any additional dependencies or common catches that someone will need to know to use the code base you have created. Think of this as the first official piece of documentation for the project that you are building.

Deep dive: package.json

The next notable file in the root directory is the **package.json** file. **package.json** exists to tell Node.js more details about this project. This could be anything from metadata about the project (name, description, author, and more) to dependency lists.

Let's take a look at a sample generated package file:

package.json

```
1 {  
2   "name": "hello-react",  
3   "version": "0.1.0",  
4   "private": true,  
5   "dependencies": {  
6     "react": "^16.8.6",  
7     "react-dom": "^16.8.6",  
8     "react-scripts": "3.0.1"  
9   },
```

The complete code can be found here: <https://packt.live/2y121qR>

Here, we have some notable keys and some more metadata-style keys combined together:

- We have a **name** key, which is basically just the name of your project itself.
- Next, we have the **version**, which should be used to indicate the semantic versioning for your project.