

Contents

Book Revision	1
Bug Reports	1
Chat With The Community!	1
Be notified of updates via Twitter	1
We'd love to hear from you!	1
Foreword	2
How to Get the Most Out of This Book	1
Overview	1
Running Code Examples	2
Project setups	2
Code Blocks and Context	3
Code Block Numbering	3
Getting Help	3
Emailing Us	4
Technical Support Response Time	5
Get excited	5
Part I	6
Your first React Web Application	7
Building Product Hunt	7
Setting up your development environment	7
Code editor	7
Node.js and npm	7
Install Git	8
Browser	8
Special instruction for Windows users	8
Ensure IIS is installed	8
JavaScript ES6/ES7	8
Getting started	9
Sample Code	9

CONTENTS

Previewing the application	9
Prepare the app	11
What's a component?	15
Our first component	16
JSX	18
The developer console	19
Babel	21
ReactDOM.render()	23
Building Product	25
Making Product data-driven	27
The data model	28
Using props	28
Rendering multiple products	32
React the vote (your app's first interaction)	37
Propagating the event	37
Binding custom component methods	39
Using state	42
Setting state with this.setState()	44
Updating state and immutability	46
Refactoring with the Babel plugin transform-class-properties	51
Babel plugins and presets	51
Property initializers	52
Refactoring Product	53
Refactoring ProductList	54
Congratulations!	56
Components	57
A time-logging app	57
Getting started	58
Previewing the app	58
Prepare the app	58
Breaking the app into components	62
The steps for building React apps from scratch	69
Step 2: Build a static version of the app	71
TimersDashboard	71
EditableTimer	73
TimerForm	74
ToggleableTimerForm	75
Timer	76
Render the app	77
Try it out	78
Step 3: Determine what should be stateful	79
State criteria	79

CONTENTS

Applying the criteria	80
Step 4: Determine in which component each piece of state should live	81
The list of timers and properties of each timer	82
Whether or not the edit form of a timer is open	82
Visibility of the create form	82
Step 5: Hard-code initial states	83
Adding state to TimersDashboard	83
Receiving props in EditableTimerList	84
Props vs. state	85
Adding state to EditableTimer	85
Timer remains stateless	86
Adding state to ToggleableTimerForm	86
Adding state to TimerForm	88
Step 6: Add inverse data flow	91
TimerForm	92
ToggleableTimerForm	93
TimersDashboard	95
Updating timers	97
Adding editability to Timer	97
Updating EditableTimer	98
Updating EditableTimerList	100
Defining onEditFormSubmit() in TimersDashboard	100
Deleting timers	103
Adding the event handler to Timer	103
Routing through EditableTimer	104
Routing through EditableTimerList	104
Implementing the delete function in TimersDashboard	105
Adding timing functionality	106
Adding a forceUpdate() interval to Timer	107
Try it out	108
Add start and stop functionality	108
Add timer action events to Timer	108
Create TimerActionButton	109
Run the events through EditableTimer and EditableTimerList	110
Try it out	113
Methodology review	114
Components & Servers	116
Introduction	116
Preparation	116
server.js	116
The Server API	117
text/html endpoint	118

CONTENTS

JSON endpoints	118
Playing with the API	119
Loading state from the server	122
Try it out	125
client	125
Fetch	126
Sending starts and stops to the server	129
Sending creates, updates, and deletes to the server	131
Give it a spin	133
Next up	133
JSX and the Virtual DOM	134
React Uses a Virtual DOM	134
Why Not Modify the Actual DOM?	134
What is a Virtual DOM?	134
Virtual DOM Pieces	135
ReactDOM	136
Experimenting with ReactDOM	136
Rendering Our ReactDOM	138
Adding Text (with children)	140
ReactDOM.render()	141
JSX	142
JSX Creates Elements	142
JSX Attribute Expressions	144
JSX Conditional Child Expressions	144
JSX Boolean Attributes	145
JSX Comments	145
JSX Spread Syntax	145
JSX Gotchas	146
JSX Summary	149
References	150
Advanced Component Configuration with props, state, and children	151
Intro	151
How to use this chapter	152
ReactDOM	152
Creating ReactDOM - createClass or ES6 Classes	152
render() Returns a ReactDOM Tree	153
Getting Data into render()	154
props are the parameters	155
PropTypes	156
Default props with getDefaultProps()	158
context	158

CONTENTS

state	163
Using state: Building a Custom Radio Button	164
Stateful components	169
State updates that depend on the current state	171
Thinking About State	173
Stateless Components	174
Switching to Stateless	175
Stateless Encourages Reuse	177
Talking to Children Components with <code>props.children</code>	177
<code>React.Children.map()</code> & <code>React.Children.forEach()</code>	180
<code>React.Children.toArray()</code>	181
Summary	182
References	182
Forms	183
Forms 101	183
Preparation	183
The Basic Button	184
Events and Event Handlers	186
Back to the Button	187
Text Input	189
Accessing User Input With <code>refs</code>	189
Using User Input	191
Uncontrolled vs. Controlled Components	194
Accessing User Input With <code>state</code>	195
Multiple Fields	197
On Validation	202
Adding Validation to Our App	202
Creating the Field Component	206
Using our new Field Component	210
Remote Data	215
Building the Custom Component	215
Adding <code>CourseSelect</code>	221
Separation of View and State	224
Async Persistence	224
Redux	231
Form Component	236
Connect the Store	240
Form Modules	242
<code>formsy-react</code>	242
<code>react-input-enhancements</code>	242
<code>tcomb-form</code>	243
<code>winterfell</code>	243

CONTENTS

react-redux-form	243
Using Webpack with Create React App	244
JavaScript modules	244
Create React App	246
Exploring Create React App	247
public/index.html	248
package.json	249
src/	251
index.js	253
Booting the app	255
Webpack basics	256
Making modifications to the sample app	262
Hot reloading	262
Auto-reloading	263
Creating a production build	264
Ejecting	267
Buckle up	268
Using Create React App with an API server	270
The completed app	270
How the app is organized	274
The server	275
Client	276
Concurrently	277
Using the Webpack development proxy	280
Webpack at large	282
When to use Webpack/Create React App	283
Unit Testing	284
Writing tests without a framework	284
Preparing Modash	285
Writing the first spec	288
The <code>assertEqual()</code> function	290
What is Jest?	294
Using Jest	294
<code>expect()</code>	295
The first Jest test for Modash	297
The other <code>truncate()</code> spec	299
The rest of the specs	300
Testing strategies for React applications	302
Integration vs Unit Testing	302
Shallow rendering	303
Enzyme	303

CONTENTS

Testing a basic React component with Enzyme	304
Setup	304
The App component	305
The first spec for App	309
More assertions for App	313
Using beforeEach	316
Simulating a change	319
Clearing the input field	323
Simulating a form submission	325
Writing tests for the food lookup app	332
FoodSearch	334
Exploring FoodSearch	336
Writing FoodSearch.test.js	340
In initial state	342
A user has typed a value into the search field	344
Mocking with Jest	348
Mocking Client	351
The API returns results	357
The user clicks on a food item	362
The API returns empty result set	366
Further reading	370
Routing	373
What's in a URL?	373
React Router's core components	375
Building the components of react-router	376
The completed app	376
Building Route	378
Building Link	385
Building Router	390
Building Redirect	395
Using react-router	399
More Route	400
Using Switch	405
Dynamic routing with React Router	407
The completed app	408
The server's API	411
Starting point of the app	413
Using URL params	419
Propagating pathnames as props	426
Dynamic menu items with NavLink	431
Supporting authenticated routes	434
The Client library	435

CONTENTS

Implementing login	436
PrivateRoute, a higher-order component	442
Redirect state	446
Recap	448
Further Reading	448

Part II 449

Intro to Flux and Redux	450
Why Flux?	450
Flux is a Design Pattern	450
Flux overview	451
Flux implementations	452
Redux	452
Redux's key ideas	452
Building a counter	453
Preparation	453
Overview	454
The counter's actions	455
Incrementing the counter	456
Decrementing the counter	457
Supporting additional parameters on actions	459
Building the store	460
Try it out	464
The core of Redux	465
Next up	466
The beginnings of a chat app	466
Previewing	466
State	468
Actions	469
Building the reducer()	469
Initializing state	469
Handling the ADD_MESSAGE action	470
Handling the DELETE_MESSAGE action	473
Subscribing to the store	475
createStore() in full	477
Connecting Redux to React	480
Using store.getState()	480
Using store.subscribe()	480
Using store.dispatch()	481
The app's components	481

CONTENTS

Preparing App.js	482
The App component	483
The MessageInput component	484
The MessageView component	486
Next up	488
Intermediate Redux	489
Preparation	489
Using createStore() from the redux library	490
Try it out	491
Representing messages as objects in state	491
Updating ADD_MESSAGE	492
Updating DELETE_MESSAGE	494
Updating the React components	495
Introducing threads	497
Supporting threads in initialState	498
Supporting threads in the React components	500
Modifying App	501
Turning MessageView into Thread	502
Try it out	503
Adding the ThreadTabs component	503
Updating App	504
Creating ThreadTabs	505
Try it out	505
Supporting threads in the reducer	506
Updating ADD_MESSAGE in the reducer	506
Updating the MessageInput component	512
Try it out	513
Updating DELETE_MESSAGE in the reducer	514
Try it out	516
Adding the action OPEN_THREAD	517
The action object	517
Modifying the reducer	517
Dispatching from ThreadTabs	518
Try it out	519
Breaking up the reducer function	520
A new reducer()	521
Updating threadsReducer()	523
Try it out	526
Adding messagesReducer()	527
Modifying the ADD_MESSAGE action handler	527
Creating messagesReducer()	528
Modifying the DELETE_MESSAGE action handler	529

CONTENTS

Adding DELETE_MESSAGE to messagesReducer()	532
Defining the initial state in the reducers	533
Initial state in reducer()	534
Adding initial state to activeThreadIdReducer()	534
Adding initial state to threadsReducer()	535
Try it out	536
Using combineReducers() from redux	536
Next up	537
Using Presentational and Container Components with Redux	539
Presentational and container components	539
Splitting up ThreadTabs	541
Splitting up Thread	546
Removing store from App	552
Try it out	553
Generating containers with react-redux	553
The Provider component	554
Wrapping App in Provider	554
Using connect() to generate ThreadTabs	555
Using connect() to generate ThreadDisplay	559
Action creators	564
Conclusion	568
Asynchronicity and server communication	568
Using GraphQL	570
Your First GraphQL Query	570
GraphQL Benefits	572
GraphQL vs. REST	573
GraphQL vs. SQL	574
Relay and GraphQL Frameworks	574
Chapter Preview	576
Consuming GraphQL	576
Exploring With GraphQL	576
GraphQL Syntax 101	580
Complex Types	584
Unions	584
Fragments	585
Interfaces	586
Exploring a Graph	587
Graph Nodes	590
Viewer	591
Graph Connections and Edges	592
Mutations	596

CONTENTS

Subscriptions	597
GraphQL With JavaScript	598
GraphQL With React	599
Wrapping Up	601
GraphQL Server	602
Writing a GraphQL Server	602
Special setup for Windows users	602
Game Plan	603
Express HTTP Server	603
Adding First GraphQL Types	606
Adding GraphiQL	608
Introspection	610
Mutation	611
Rich Schemas and SQL	614
Setting Up The Database	615
Schema Design	619
Object and Scalar Types	621
Lists	626
Performance: Look-Ahead Optimizations	628
Lists Continued	631
Connections	634
Authentication	641
Authorization	643
Rich Mutations	647
Relay and GraphQL	650
Performance: N+1 Queries	651
Summary	655
Relay	657
Introduction	657
What We're Going to Cover	658
What We're Building	658
Guide to the Code Structure	662
Relay is a Data Architecture	663
Relay GraphQL Conventions	664
Exploring Relay Conventions in GraphQL	665
Fetching Objects By ID	665
Walking Connections	669
Changing Data with Mutations	674
Relay GraphQL Queries Summary	675
Adding Relay to Our App	675
Quick Look at the Goal	675

CONTENTS

A Preview of the Author Page	678
Containers, Queries, and Fragments	679
Validating Our Relay Queries at Compile Time	679
Setting Up Routing	685
Adding Relay to Our Routes	687
App Component	688
AuthorQueries Component	689
AuthorPage Component	689
Try It Out	691
AuthorPage with Styles	693
BooksPage	695
BooksPage Route	695
BooksPage Component	696
BooksPage render()	698
BookItem	699
BookItem Fragment	701
Fragment Value Masking	701
Improving the AuthorPage	703
Changing Data With Mutations	706
Building a Book's Page	707
Book Page Editing	710
Mutations	713
Defining a Mutation Object	714
Inline Editing	718
Conclusion	720
Where to Go From Here	721
React Native	722
Init	723
Routing	724
<Navigator />	727
renderScene()	729
configureScene()	731
Web components vs. Native components	734
<View />	735
<Text />	735
<Image />	735
<TextInput />	735
<TouchableHighlight />, <TouchableOpacity />, and <TouchableWithoutFeedback />	735
<ActivityIndicator />	736
<WebView />	736
<ScrollView />	736

CONTENTS

<ListView />	737
Styles	745
StyleSheet	746
Flexbox	747
HTTP requests	765
What is a promise	766
Enter Promises	768
Single-use guarantee	770
Creating a promise	770
Debugging with React Native	772
Where to go from here	773
Appendix A: PropTypes	775
Validators	776
string	777
number	777
boolean	778
function	779
object	780
object shape	780
multiple types	781
instanceOf	782
array	783
array of type	784
node	785
element	786
any type	787
Optional & required props	787
custom validator	788
Appendix B: ES6	790
Prefer const and let over var	790
Arrow functions	790
Modules	793
Object.assign()	796
Template literals	797
The spread operator (...)	797
Enhanced object literals	798
Default arguments	798
Destructuring assignments	799
Changelog	802
Revision 32 - 2017-06-14	802

CONTENTS

Revision 31 - 2017-05-18	802
Revision 30 - 2017-04-20	802
Revision 29 - 2017-04-13	802
Revision 28 - 2017-04-10	802
Revision 27 - 2017-03-16	802
Revision 26 - 2017-02-22	803
Revision 25 - 2017-02-17	803
Revision 24 - 2017-02-08	803
Revision 23 - 2017-02-06	803
Revision 22 - 2017-02-01	803
Revision 21 - 2017-01-27	803
Revision 20 - 2017-01-10	804
Revision 19 - 2016-12-20	804
Revision 18 - 2016-11-22	804
Revision 17 - 2016-11-04	804
Revision 16 - 2016-10-12	804
Revision 15 - 2016-10-05	804
Revision 14 - 2016-08-26	804
Revision 13 - 2016-08-02	804
Revision 12 - 2016-07-26	804
Revision 11 - 2016-07-08	805
Revision 10 - 2016-06-24	805
Revision 9 - 2016-06-21	805
Revision 8 - 2016-06-02	805
Revision 7 - 2016-05-13	805
Revision 6 - 2016-05-13	805
Revision 5 - 2016-04-25	805
Revision 4 - 2016-04-22	805
Revision 3 - 2016-04-08	805
Revision 2 - 2016-03-16	806
Revision 1 - 2016-02-14	806

Book Revision

Revision 32 - Supports React 15.5.4 (2017-06-14)

Bug Reports

If you'd like to report any bugs, typos, or suggestions just email us at: react@fullstack.io¹.

Chat With The Community!

There's an unofficial community chat room for this book using Gitter. If you'd like to hang out with other people learning React, come [join us on Gitter](https://gitter.im/fullstackreact/fullstackreact)²!

Be notified of updates via Twitter

If you'd like to be notified of updates to the book on Twitter, [follow @fullstackio](https://twitter.com/fullstackio)³

We'd love to hear from you!

Did you like the book? Did you find it helpful? We'd love to add your face to our list of testimonials on the website! Email us at: react@fullstack.io⁴.

¹<mailto:react@fullstack.io?Subject=Fullstack%20React%20book%20feedback>

²<https://gitter.im/fullstackreact/fullstackreact>

³<https://twitter.com/fullstackio>

⁴<mailto:react@fullstack.io?Subject=react%20%20testimonial>

Foreword

Web development is often seen as a crazy world where the way you develop software is by throwing hacks on top of hacks. I believe that React breaks from this pattern and instead has been designed from first principle which gives you a solid foundation to build on.

A major source of bugs for front-end applications was around synchronizing the data model with the DOM. It is very hard to make sure that whenever data changes, everything in the UI is updated with it.

React's first innovation was to introduce a pure-JavaScript representation of the DOM and implement diffing in userland and then use events which send simple commands: create, update, delete.

With React, by conceptually **re-rendering everything whenever anything changes**, not only do you have code that is **safe by default**, it is also **less work** as you only need to write the creation path: updates are taken care of for you.

Browsers have, for a long time, been incompatible in various ways due to the large API surface area of what they have to support to make the DOM work. Not only does React provide a great way to solve browser differences, but it enables use cases that were never before possible for a front-end library, such as server-side rendering and the ability to implement rendering targets like native iOS, Android, and even hardware components.

But the most important thing about React and the main reason why you should read this book: not only will you use it to **make great applications for your users**, it will also **make you a better developer**. Libraries come and go all the time and React is likely going to be no exception. What makes it special is that it teaches you **concepts that can be reused throughout your entire career**.

You will become better at JavaScript because React doesn't come with a templating system. Instead, React pushes you to use the **full power of JavaScript** to build your user interface.

You are going to practice using parts of *functional programming* with `map` and `filter` and also encouraged to use the **latest features of JavaScript** (including ES6). By not abstracting away data management, React will force you to think about how to architect your app and encourage you to consider concepts like immutability.

I'm very proud that the community built around React is not afraid of "rethinking best practices." The community challenges the status quo in many areas. My advice to you is to read this **excellent**



Christopher Chedeau - Front-end Engineer at Facebook

book to learn and understand the fundamentals of React. Learning new concepts may feel strange but “give it 5 minutes” and practice them until you feel comfortable.

Then, **try to break the rules**. There is no one best way to build software and React is no exception. React actually embraces this fact by providing you with escape hatches when you want to do things outside of the React-way.

Come up with crazy ideas and who knows, maybe you are going to invent the successor to React!

– Christopher Chedeau [@vjeux](#)⁵ Front-end Engineer at Facebook

⁵<https://twitter.com/Vjeux>