

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	9
JavaScript ES6/ES7	9
Getting started	10
Sample Code	10
Previewing the application	10

CONTENTS

Prepare the app	12
What's a component?	16
Our first component	17
JSX	19
The developer console	21
Babel	22
ReactDOM.render()	24
Building Product	26
Making Product data-driven	28
The data model	29
Using props	29
Rendering multiple products	34
React the vote (your app's first interaction)	39
Propagating the event	40
Binding custom component methods	42
Using state	47
Setting state with this.setState()	48
Updating state and immutability	50
Refactoring with the Babel plugin transform-class-properties	56
Babel plugins and presets	56
Property initializers	57
Refactoring Product	58
Refactoring ProductList	59
Congratulations!	61
Components	62
A time-logging app	62
Getting started	63
Previewing the app	63
Prepare the app	63
Breaking the app into components	67
The steps for building React apps from scratch	74
Step 2: Build a static version of the app	76
TimersDashboard	76
EditableTimer	78
TimerForm	79
ToggleableTimerForm	80
Timer	81
Render the app	82
Try it out	83
Step 3: Determine what should be stateful	84
State criteria	85
Applying the criteria	85

CONTENTS

Step 4: Determine in which component each piece of state should live	86
The list of timers and properties of each timer	87
Whether or not the edit form of a timer is open	87
Visibility of the create form	87
Step 5: Hard-code initial states	88
Adding state to TimersDashboard	88
Receiving props in EditableTimerList	89
Props vs. state	90
Adding state to EditableTimer	90
Timer remains stateless	91
Adding state to ToggleableTimerForm	91
Adding state to TimerForm	93
Step 6: Add inverse data flow	96
TimerForm	97
ToggleableTimerForm	98
TimersDashboard	100
Updating timers	102
Adding editability to Timer	102
Updating EditableTimer	103
Updating EditableTimerList	105
Defining onEditFormSubmit() in TimersDashboard	105
Deleting timers	108
Adding the event handler to Timer	108
Routing through EditableTimer	108
Routing through EditableTimerList	109
Implementing the delete function in TimersDashboard	109
Adding timing functionality	111
Adding a forceUpdate() interval to Timer	112
Try it out	113
Add start and stop functionality	113
Add timer action events to Timer	113
Create TimerActionButton	114
Run the events through EditableTimer and EditableTimerList	115
Try it out	118
Methodology review	119
Components & Servers	121
Introduction	121
Preparation	121
server.js	121
The Server API	122
text/html endpoint	123
JSON endpoints	123

CONTENTS

Playing with the API	124
Loading state from the server	127
Try it out	129
client	130
Fetch	130
Sending starts and stops to the server	133
Sending creates, updates, and deletes to the server	136
Give it a spin	137
Next up	137
JSX and the Virtual DOM	138
React Uses a Virtual DOM	138
Why Not Modify the Actual DOM?	138
What is a Virtual DOM?	138
Virtual DOM Pieces	139
ReactDOM	140
Experimenting with ReactDOM	140
Rendering Our ReactDOM	142
Adding Text (with children)	144
ReactDOM.render()	145
JSX	146
JSX Creates Elements	146
JSX Attribute Expressions	147
JSX Conditional Child Expressions	148
JSX Boolean Attributes	148
JSX Comments	149
JSX Spread Syntax	149
JSX Gotchas	150
JSX Summary	153
References	153
Advanced Component Configuration with props, state, and children	154
Intro	154
How to use this chapter	155
ReactDOM	155
Creating ReactDOM - createClass or ES6 Classes	155
render() Returns a ReactDOM Tree	156
Getting Data into render()	157
props are the parameters	158
PropTypes	159
Default props with getDefaultProps()	161
context	161
state	167

CONTENTS

Using state: Building a Custom Radio Button	167
Stateful components	172
Thinking About State	174
Stateless Components	176
Switching to Stateless	177
Stateless Encourages Reuse	178
Talking to Children Components with props.children	179
React.Children.map() & React.Children.forEach()	181
React.Children.toArray()	183
Summary	183
References	184
Forms	185
Forms 101	185
Preparation	185
The Basic Button	186
Events and Event Handlers	188
Back to the Button	189
Text Input	191
Accessing User Input With refs	192
Using User Input	194
Uncontrolled vs. Controlled Components	197
Accessing User Input With state	198
Multiple Fields	200
On Validation	206
Adding Validation to Our App	206
Creating the Field Component	210
Using our new Field Component	214
Remote Data	219
Building the Custom Component	220
Adding CourseSelect	225
Separation of View and State	228
Async Persistence	229
Redux	235
Form Component	241
Connect the Store	244
Form Modules	246
formsy-react	246
react-input-enhancements	247
tcomb-form	247
winterfell	247
react-redux-form	248

CONTENTS

Using Webpack with Create React App	249
JavaScript modules	249
Create React App	251
Exploring Create React App	252
public/index.html	253
package.json	254
src/	256
index.js	258
Bootling the app	262
Webpack basics	264
Making modifications to the sample app	270
Hot reloading	270
Auto-reloading	271
Creating a production build	272
Ejecting	275
Buckle up	276
Using Create React App with an API server	278
The completed app	278
How the app is organized	282
The server	283
Client	284
Concurrently	285
Using the Webpack development proxy	288
Webpack at large	290
When to use Webpack/Create React App	291
Unit Testing	292
Writing tests without a framework	292
Preparing Modash	293
Writing the first spec	297
The <code>assertEqual()</code> function	298
What is Jest?	302
Using Jest	302
<code>expect()</code>	303
The first Jest test for Modash	306
The other <code>truncate()</code> spec	307
The rest of the specs	308
Testing strategies for React applications	310
Integration vs Unit Testing	310
Shallow rendering	311
Enzyme	311
Testing a basic React component with Enzyme	312
Setup	312

CONTENTS

The App component	313
The first spec for App	317
More assertions for App	321
Using beforeEach	324
Simulating a change	327
Clearing the input field	331
Simulating a form submission	333
Writing tests for the food lookup app	340
FoodSearch	342
Exploring FoodSearch	344
Writing FoodSearch.test.js	348
In initial state	350
A user has typed a value into the search field	352
Mocking with Jest	356
Mocking Client	359
The API returns results	365
The user clicks on a food item	370
The API returns empty result set	374
Further reading	378
Routing	381
What's in a URL?	381
React Router's core components	383
Building the components of react-router	384
The completed app	384
Building Route	386
Building Link	393
Building Router	398
Building Redirect	403
Using react-router	407
More Route	408
Using Switch	413
Dynamic routing with React Router	415
The completed app	416
The server's API	419
Starting point of the app	421
Using URL params	427
Propagating pathnames as props	434
Dynamic menu items with NavLink	439
Supporting authenticated routes	442
The Client library	443
Implementing login	444
PrivateRoute, a higher-order component	450

CONTENTS

Redirect state	454
Recap	456
Further Reading	456
Part II	457
Intro to Flux and Redux	458
Why Flux?	458
Flux is a Design Pattern	458
Flux overview	459
Flux implementations	460
Redux	460
Redux's key ideas	460
Building a counter	461
Preparation	461
Overview	462
The counter's actions	463
Incrementing the counter	464
Decrementing the counter	465
Supporting additional parameters on actions	467
Building the store	468
Try it out	472
The core of Redux	473
Next up	474
The beginnings of a chat app	475
Previewing	475
State	477
Actions	477
Building the reducer()	478
Initializing state	478
Handling the ADD_MESSAGE action	478
Handling the DELETE_MESSAGE action	481
Subscribing to the store	484
createStore() in full	486
Connecting Redux to React	489
Using store.getState()	489
Using store.subscribe()	489
Using store.dispatch()	490
The app's components	490
Preparing App.js	491
The App component	492
The MessageInput component	493

CONTENTS

The <code>MessageView</code> component	495
Next up	497
Intermediate Redux	498
Preparation	498
Using <code>createStore()</code> from the <code>redux</code> library	499
Try it out	500
Representing messages as objects in state	500
Updating <code>ADD_MESSAGE</code>	501
Updating <code>DELETE_MESSAGE</code>	503
Updating the React components	504
Introducing threads	506
Supporting threads in <code>initialState</code>	507
Supporting threads in the React components	509
Modifying <code>App</code>	510
Turning <code>MessageView</code> into <code>Thread</code>	511
Try it out	512
Adding the <code>ThreadTabs</code> component	512
Updating <code>App</code>	513
Creating <code>ThreadTabs</code>	514
Try it out	514
Supporting threads in the reducer	515
Updating <code>ADD_MESSAGE</code> in the reducer	515
Updating the <code>MessageInput</code> component	521
Try it out	522
Updating <code>DELETE_MESSAGE</code> in the reducer	523
Try it out	525
Adding the action <code>OPEN_THREAD</code>	526
The action object	526
Modifying the reducer	526
Dispatching from <code>ThreadTabs</code>	527
Try it out	528
Breaking up the reducer function	529
A new <code>reducer()</code>	530
Updating <code>threadsReducer()</code>	532
Try it out	535
Adding <code>messagesReducer()</code>	536
Modifying the <code>ADD_MESSAGE</code> action handler	536
Creating <code>messagesReducer()</code>	537
Modifying the <code>DELETE_MESSAGE</code> action handler	538
Adding <code>DELETE_MESSAGE</code> to <code>messagesReducer()</code>	541
Defining the initial state in the reducers	542
Initial state in <code>reducer()</code>	543

CONTENTS

Adding initial state to <code>activeThreadIdReducer()</code>	544
Adding initial state to <code>threadsReducer()</code>	544
Try it out	546
Using <code>combineReducers()</code> from <code>redux</code>	546
Next up	547
Using Presentational and Container Components with Redux	549
Presentational and container components	549
Splitting up <code>ThreadTabs</code>	551
Splitting up <code>Thread</code>	556
Removing store from <code>App</code>	562
Try it out	563
Generating containers with <code>react-redux</code>	563
The <code>Provider</code> component	564
Wrapping <code>App</code> in <code>Provider</code>	564
Using <code>connect()</code> to generate <code>ThreadTabs</code>	565
Using <code>connect()</code> to generate <code>ThreadDisplay</code>	569
Action creators	574
Conclusion	578
Asynchronicity and server communication	578
Using GraphQL	580
Your First GraphQL Query	580
GraphQL Benefits	582
GraphQL vs. REST	583
GraphQL vs. SQL	584
Relay and GraphQL Frameworks	584
Chapter Preview	586
Consuming GraphQL	586
Exploring With <code>GraphiQL</code>	586
GraphQL Syntax 101	594
Complex Types	598
Unions	599
Fragments	600
Interfaces	601
Exploring a Graph	602
Graph Nodes	605
Viewer	607
Graph Connections and Edges	608
Mutations	611
Subscriptions	612
GraphQL With JavaScript	613
GraphQL With React	615

CONTENTS

Wrapping Up	616
GraphQL Server	617
Writing a GraphQL Server	617
Special setup for Windows users	617
Game Plan	618
Express HTTP Server	618
Adding First GraphQL Types	621
Adding GraphQL	623
Introspection	626
Mutation	627
Rich Schemas and SQL	630
Setting Up The Database	631
Schema Design	635
Object and Scalar Types	636
Lists	642
Performance: Look-Ahead Optimizations	644
Lists Continued	647
Connections	650
Authentication	657
Authorization	659
Rich Mutations	663
Relay and GraphQL	666
Performance: N+1 Queries	667
Summary	671
Relay	673
Introduction	673
What We're Going to Cover	674
What We're Building	674
Guide to the Code Structure	679
Relay is a Data Architecture	681
Relay GraphQL Conventions	681
Exploring Relay Conventions in GraphQL	682
Fetching Objects By ID	683
Walking Connections	688
Changing Data with Mutations	692
Relay GraphQL Queries Summary	693
Adding Relay to Our App	694
Quick Look at the Goal	694
A Preview of the Author Page	696
Containers, Queries, and Fragments	697
Validating Our Relay Queries at Compile Time	698

CONTENTS

Setting Up Routing	703
Adding Relay to Our Routes	705
App Component	706
AuthorQueries Component	707
AuthorPage Component	707
Try It Out	709
AuthorPage with Styles	711
BooksPage	713
BooksPage Route	714
BooksPage Component	715
BooksPage render()	717
BookItem	719
BookItem Fragment	720
Fragment Value Masking	720
Improving the AuthorPage	722
Changing Data With Mutations	726
Building a Book's Page	727
Book Page Editing	730
Mutations	733
Defining a Mutation Object	734
Inline Editing	738
Conclusion	740
Where to Go From Here	741
React Native	742
Init	743
Routing	744
<Navigator />	748
renderScene()	749
configureScene()	751
Web components vs. Native components	755
<View />	755
<Text />	755
<Image />	756
<TextInput />	756
<TouchableHighlight />, <TouchableOpacity />, and <TouchableWithoutFeedback />	756
<ActivityIndicator />	757
<WebView />	757
<ScrollView />	757
<ListView />	757
Styles	765
StyleSheet	766

CONTENTS

Flexbox	767
HTTP requests	793
What is a promise	794
Enter Promises	796
Single-use guarantee	798
Creating a promise	798
Debugging with React Native	800
Where to go from here	802
Appendix A: PropTypes	804
Validators	805
string	806
number	806
boolean	807
function	808
object	808
object shape	809
multiple types	810
instanceOf	811
array	812
array of type	813
node	814
element	815
any type	816
Optional & required props	816
custom validator	817
Appendix B: Tools	819
Curl	819
A GET Request	819
A POST Request	819
Chrome “Copy as cURL”	820
More Resources	821
Changelog	822
Revision 30 - 2017-04-20	822
Revision 29 - 2017-04-13	822
Revision 28 - 2017-04-10	822
Revision 27 - 2017-03-16	822
Revision 26 - 2017-02-22	822
Revision 25 - 2017-02-17	822
Revision 24 - 2017-02-08	823
Revision 23 - 2017-02-06	823

CONTENTS

Revision 22 - 2017-02-01	823
Revision 21 - 2017-01-27	823
Revision 20 - 2017-01-10	823
Revision 19 - 2016-12-20	823
Revision 18 - 2016-11-22	823
Revision 17 - 2016-11-04	824
Revision 16 - 2016-10-12	824
Revision 15 - 2016-10-05	824
Revision 14 - 2016-08-26	824
Revision 13 - 2016-08-02	824
Revision 12 - 2016-07-26	824
Revision 11 - 2016-07-08	824
Revision 10 - 2016-06-24	824
Revision 9 - 2016-06-21	824
Revision 8 - 2016-06-02	825
Revision 7 - 2016-05-13	825
Revision 6 - 2016-05-13	825
Revision 5 - 2016-04-25	825
Revision 4 - 2016-04-22	825
Revision 3 - 2016-04-08	825
Revision 2 - 2016-03-16	825
Revision 1 - 2016-02-14	825

Book Revision

Revision 30 - Supports React 15.4.1 (2017-04-20)

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.

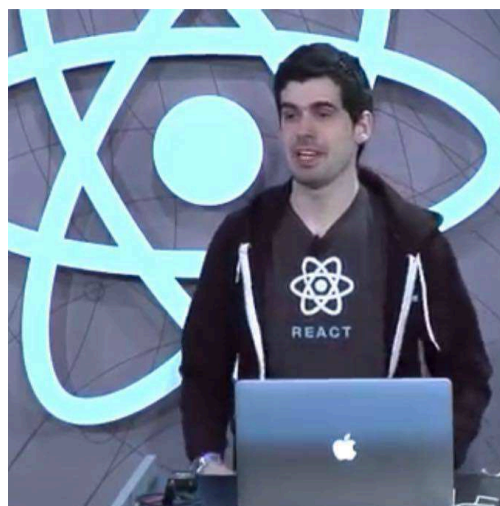
Browsers have, for a long time, been incompatible in various ways due to the large API surface area of what they have to support to make the DOM work. Not only does React provide a great way to solve browser differences, but it enables use cases that were never before possible for a front-end library, 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>