

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	19
The developer console	20
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	96
Adding editability to Timer	97
Updating EditableTimer	97
Updating EditableTimerList	99
Defining onEditFormSubmit() in TimersDashboard	99
Deleting timers	102
Adding the event handler to Timer	102
Routing through EditableTimer	103
Routing through EditableTimerList	103
Implementing the delete function in TimersDashboard	104
Adding timing functionality	105
Adding a forceUpdate() interval to Timer	106
Try it out	107
Add start and stop functionality	107
Add timer action events to Timer	107
Create TimerActionButton	108
Run the events through EditableTimer and EditableTimerList	109
Try it out	112
Methodology review	113
Components & Servers	115
Introduction	115
Preparation	115
server.js	115
The Server API	116
text/html endpoint	117

CONTENTS

JSON endpoints	117
Playing with the API	118
Loading state from the server	121
Try it out	124
client	124
Fetch	125
Sending starts and stops to the server	128
Sending creates, updates, and deletes to the server	130
Give it a spin	132
Next up	132
JSX and the Virtual DOM	133
React Uses a Virtual DOM	133
Why Not Modify the Actual DOM?	133
What is a Virtual DOM?	133
Virtual DOM Pieces	134
ReactDOM	135
Experimenting with ReactDOM	135
Rendering Our ReactDOM	137
Adding Text (with children)	139
ReactDOM.render()	140
JSX	141
JSX Creates Elements	141
JSX Attribute Expressions	143
JSX Conditional Child Expressions	143
JSX Boolean Attributes	144
JSX Comments	144
JSX Spread Syntax	144
JSX Gotchas	145
JSX Summary	148
References	149
Advanced Component Configuration with props, state, and children	150
Intro	150
How to use this chapter	151
ReactDOM	152
Creating ReactDOM - createReactClass or ES6 Classes	152
render() Returns a ReactDOM Tree	153
Getting Data into render()	154
props are the parameters	154
PropTypes	155
Default props with getDefaultProps()	157
context	157

CONTENTS

Default value	161
Multiple contexts	161
state	162
Using state: Building a Custom Radio Button	162
Stateful components	168
State updates that depend on the current state	170
Thinking About State	171
Stateless Components	172
Switching to Stateless	173
Stateless Encourages Reuse	175
Talking to Children Components with props.children	175
React.Children.map() & React.Children.forEach()	178
React.Children.toArray()	179
Summary	180
References	180
Forms	181
Forms 101	181
Preparation	181
The Basic Button	182
Events and Event Handlers	184
Back to the Button	185
Text Input	187
Accessing User Input With refs	187
Using User Input	189
Uncontrolled vs. Controlled Components	192
Accessing User Input With state	193
Multiple Fields	195
On Validation	200
Adding Validation to Our App	200
Creating the Field Component	204
Using our new Field Component	208
Remote Data	213
Building the Custom Component	213
Adding CourseSelect	219
Separation of View and State	222
Async Persistence	222
Redux	229
Form Component	234
Connect the Store	238
Form Modules	240
formsy-react	240
react-input-enhancements	240

CONTENTS

tcomb-form	241
winterfell	241
react-redux-form	241
Using Webpack with Create React App	242
JavaScript modules	242
Create React App	244
Exploring Create React App	245
public/index.html	246
package.json	246
src/	249
index.js	251
Booting the app	252
Webpack basics	254
Making modifications to the sample app	260
Hot reloading	260
Auto-reloading	261
Creating a production build	262
Ejecting	265
Buckle up	266
Using Create React App with an API server	268
The completed app	268
How the app is organized	272
The server	273
Client	274
Concurrently	275
Using the Webpack development proxy	278
Webpack at large	280
When to use Webpack/Create React App	281
Unit Testing	282
Writing tests without a framework	282
Preparing Modash	283
Writing the first spec	286
The <code>assertEqual()</code> function	288
What is Jest?	292
Using Jest	292
<code>expect()</code>	293
The first Jest test for Modash	295
The other <code>truncate()</code> spec	297
The rest of the specs	298
Testing strategies for React applications	300
Integration vs Unit Testing	300

CONTENTS

Shallow rendering	301
Enzyme	301
Testing a basic React component with Enzyme	302
Setup	302
The App component	303
The first spec for App	307
More assertions for App	313
Using <code>beforeEach</code>	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 <code>FoodSearch.test.js</code>	340
In initial state	342
A user has typed a value into the search field	344
Mocking with Jest	348
Mocking <code>Client</code>	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 <code>react-router</code>	376
The completed app	376
Building <code>Route</code>	378
Building <code>Link</code>	385
Building <code>Router</code>	390
Building <code>Redirect</code>	395
Using <code>react-router</code>	400
More <code>Route</code>	401
Using <code>Switch</code>	406
Dynamic routing with React Router	408
The completed app	409
The server's API	412
Starting point of the app	414
Using URL params	420
Propagating pathnames as props	427
Dynamic menu items with <code>NavLink</code>	432

CONTENTS

Supporting authenticated routes	435
The Client library	436
Implementing login	437
PrivateRoute, a higher-order component	443
Redirect state	447
Recap	449
Further Reading	449

Part II 450

Intro to Flux and Redux	451
Why Flux?	451
Flux is a Design Pattern	451
Flux overview	452
Flux implementations	453
Redux	453
Redux's key ideas	453
Building a counter	454
Preparation	454
Overview	455
The counter's actions	456
Incrementing the counter	456
Decrementing the counter	458
Supporting additional parameters on actions	459
Building the store	461
Try it out	464
The core of Redux	465
Next up	466
The beginnings of a chat app	467
Previewing	467
State	469
Actions	469
Building the reducer()	470
Initializing state	470
Handling the ADD_MESSAGE action	470
Handling the DELETE_MESSAGE action	473
Subscribing to the store	476
createStore() in full	478
Connecting Redux to React	480
Using store.getState()	480
Using store.subscribe()	481

CONTENTS

Using <code>store.dispatch()</code>	481
The app's components	482
Preparing <code>App.js</code>	483
The <code>App</code> component	483
The <code>MessageInput</code> component	485
The <code>MessageView</code> component	487
Next up	488
Intermediate Redux	490
Preparation	490
Using <code>createStore()</code> from the <code>redux</code> library	491
Try it out	492
Representing messages as objects in state	492
Updating <code>ADD_MESSAGE</code>	493
Updating <code>DELETE_MESSAGE</code>	495
Updating the React components	496
Introducing threads	498
Supporting threads in <code>initialState</code>	499
Supporting threads in the React components	501
Modifying <code>App</code>	502
Turning <code>MessageView</code> into <code>Thread</code>	503
Try it out	504
Adding the <code>ThreadTabs</code> component	504
Updating <code>App</code>	505
Creating <code>ThreadTabs</code>	506
Try it out	506
Supporting threads in the reducer	507
Updating <code>ADD_MESSAGE</code> in the reducer	507
Updating the <code>MessageInput</code> component	513
Try it out	514
Updating <code>DELETE_MESSAGE</code> in the reducer	515
Try it out	517
Adding the action <code>OPEN_THREAD</code>	518
The action object	518
Modifying the reducer	518
Dispatching from <code>ThreadTabs</code>	519
Try it out	520
Breaking up the reducer function	521
A new <code>reducer()</code>	522
Updating <code>threadsReducer()</code>	524
Try it out	527
Adding <code>messagesReducer()</code>	528
Modifying the <code>ADD_MESSAGE</code> action handler	528

CONTENTS

Creating messagesReducer()	529
Modifying the DELETE_MESSAGE action handler	530
Adding DELETE_MESSAGE to messagesReducer()	533
Defining the initial state in the reducers	534
Initial state in reducer()	535
Adding initial state to activeThreadIdReducer()	535
Adding initial state to threadsReducer()	536
Try it out	537
Using combineReducers() from redux	537
Next up	538
Using Presentational and Container Components with Redux	540
Presentational and container components	540
Splitting up ThreadTabs	542
Splitting up Thread	547
Removing store from App	553
Try it out	554
Generating containers with react-redux	554
The Provider component	555
Wrapping App in Provider	555
Using connect() to generate ThreadTabs	556
Using connect() to generate ThreadDisplay	560
Action creators	565
Conclusion	569
Asynchronicity and server communication	569
Using GraphQL	571
Your First GraphQL Query	571
GraphQL Benefits	573
GraphQL vs. REST	574
GraphQL vs. SQL	575
Relay and GraphQL Frameworks	575
Chapter Preview	577
Consuming GraphQL	577
Exploring With GraphiQL	577
GraphQL Syntax 101	581
Complex Types	585
Unions	585
Fragments	586
Interfaces	587
Exploring a Graph	588
Graph Nodes	591
Viewer	592

CONTENTS

Graph Connections and Edges	593
Mutations	597
Subscriptions	598
GraphQL With JavaScript	599
GraphQL With React	600
Wrapping Up	602
GraphQL Server	603
Writing a GraphQL Server	603
Special setup for Windows users	603
Game Plan	604
Express HTTP Server	604
Adding First GraphQL Types	607
Adding GraphiQL	609
Introspection	611
Mutation	612
Rich Schemas and SQL	615
Setting Up The Database	616
Schema Design	620
Object and Scalar Types	622
Lists	627
Performance: Look-Ahead Optimizations	629
Lists Continued	632
Connections	635
Authentication	642
Authorization	644
Rich Mutations	648
Relay and GraphQL	651
Performance: N+1 Queries	652
Summary	656
React Native	658
Init	659
Routing	660
<Navigator />	663
renderScene()	665
configureScene()	667
Web components vs. Native components	670
<View />	671
<Text />	671
<Image />	671
<TextInput />	671

CONTENTS

<TouchableHighlight />, <TouchableOpacity />, and <TouchableWithoutFeedback />	671
<ActivityIndicator />	672
<WebView />	672
<ScrollView />	672
<ListView />	673
Styles	681
StyleSheet	682
Flexbox	683
HTTP requests	701
What is a promise	702
Enter Promises	704
Single-use guarantee	706
Creating a promise	706
Debugging with React Native	708
Where to go from here	709
Appendix A: PropTypes	711
Validators	712
string	713
number	713
boolean	714
function	715
object	716
object shape	716
multiple types	717
instanceOf	718
array	719
array of type	720
node	721
element	722
any type	723
Optional & required props	723
custom validator	724
Appendix B: ES6	726
Prefer const and let over var	726
Arrow functions	726
Modules	729
Object.assign()	732
Template literals	733
The spread operator (...)	733
Enhanced object literals	734

CONTENTS

Default arguments	734
Destructuring assignments	735
Changelog	738
Revision 36 - 2018-04-02	738
Revision 34 - 2017-10-17	738
Revision 33 - 2017-08-31	738
Revision 32 - 2017-06-14	738
Revision 31 - 2017-05-18	738
Revision 30 - 2017-04-20	738
Revision 29 - 2017-04-13	739
Revision 28 - 2017-04-10	739
Revision 27 - 2017-03-16	739
Revision 26 - 2017-02-22	739
Revision 25 - 2017-02-17	739
Revision 24 - 2017-02-08	739
Revision 23 - 2017-02-06	739
Revision 22 - 2017-02-01	740
Revision 21 - 2017-01-27	740
Revision 20 - 2017-01-10	740
Revision 19 - 2016-12-20	740
Revision 18 - 2016-11-22	740
Revision 17 - 2016-11-04	740
Revision 16 - 2016-10-12	740
Revision 15 - 2016-10-05	741
Revision 14 - 2016-08-26	741
Revision 13 - 2016-08-02	741
Revision 12 - 2016-07-26	741
Revision 11 - 2016-07-08	741
Revision 10 - 2016-06-24	741
Revision 9 - 2016-06-21	741
Revision 8 - 2016-06-02	741
Revision 7 - 2016-05-13	741
Revision 6 - 2016-05-13	742
Revision 5 - 2016-04-25	742
Revision 4 - 2016-04-22	742
Revision 3 - 2016-04-08	742
Revision 2 - 2016-03-16	742
Revision 1 - 2016-02-14	742

Book Revision

Revision 36 - Supports React 16.3.0 (2018-04-02)

Bug Reports

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

For further help dealing with issues, refer to "How to Get the Most Out of This Book."

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%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.

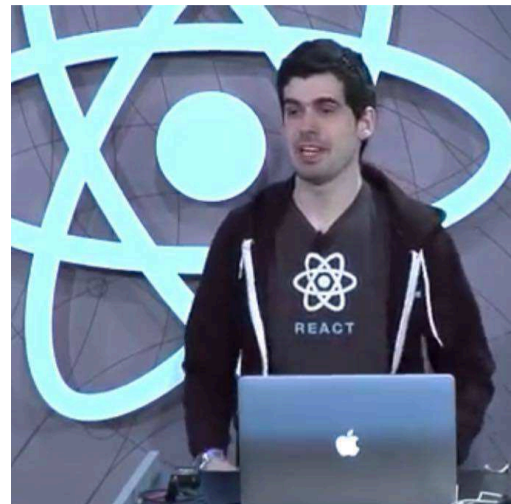
With React, by conceptually **re-rendering everything whenever anything changes**, not only do you have code that is **safe by default**, it is also **less work** as you only need to write the creation path: updates are taken care of for you.

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.



Christopher Chedeau - Front-end Engineer at Facebook

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** 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](https://twitter.com/Vjeux)⁵ Front-end Engineer at Facebook

⁵<https://twitter.com/Vjeux>