UNIVERSITY OF WATERLOO

Faculty of Engineering

Department of Electrical and Computer Engineering

React and Redux Rendering Performance Report

Bespoke Metrics, Inc.

55 York Street, Suite 1203

Toronto, Ontario, Canada

Prepared by

Chang Bok Lee

ID 2031001

userid cb3lee

3A, Computer Engineering

23 December 2017

Chang Bok Lee

23 Sheppard Ave East, Unit 706

North York, Ontario

M2N 0C8

December 23, 2017

Vincent Gaudet, chair

Electrical and Computer Engineering

University of Waterloo

Waterloo, Ontario

N2L 3G1

Dear Sir:

This report “React Rendering Performance Report”, was prepared for my 3B work report. This report is intended for the WKRPT 301 course and it was written during my work term at Bespoke Metric, where I worked as a full stack web developer for the company’s main product, COMPASS.

Bespoke Metrics provides data control and analysis services. The current main product, COMPASS, is focused on managing risks between subcontractors and general contractors in major construction projects.

The software engineering department of Bespoke Metrics is mainly separated into two teams; Front-end and Back-end. The Front-end team, in which I was employed, is managed by Jason Bellamy and primarily focused on developing the COMPASS application user interface. The front-end of COMPASS application is designed as a single page web application.

I would like to thank Mr. Bellamy for his guidance around the web application ecosystem. He helped me to understand React and Redux. Without his help, I would not be writing about React Performance in this report. I also would like to thank my mentor Ms. M Heo for reviewing my report. I hereby confirm that I have received no further help other than what is mentioned above in writing this report. I also confirm this report has not been previously submitted for academic credit at this or any other academic institution.

Sincerely,

Chang Bok Lee

ID 20361001

# Contributions

The Engineering Team of Bespoke Metrics Inc. was relatively small in size. It consisted of four full time employees and six coops including myself. There were nine other full time employees and one coop student in the finance team. The company was less than a year old when I started my term. The company was in the process of finalizing the terms and agreements with its first major client. At the stage, the company’s first product, COMPASS, was successfully launched and being used on a daily basis.

COMPASS is a web application targeting to help construction general contractors to manage their risks over hiring subcontractors. The product was roughly divided into three parts; general contractor mode, subcontractor mode, and admin mode. Subcontractors are required to fill in the questionnaire electronically and the answers are aggregated to generate quantifiable credit ratings. The credit rating of each subcontractor then would be presented to general contractors. The engineering team was divided into two teams; backend and frontend. The backend team was responsible for building a scalable server architecture providing RESTful API for the frontend team to use. The frontend team was focused on creating a single-page application. All coops were asked to join either team. I chose to join the frontend team.

The frontend is a web application built with React, Redux, and many other libraries. My main tasks were focused on adding more features to the COMPASS frontend. Along with the other coops, I successfully launched the very first admin page and added features to it. Due to the size and the stage of the company, it was natural to see bugs in production and I contributed to fix the bugs promptly. In the beginning of the work term, I prepared a presentation for some of the other coops to get familiar with the development tools such as *git* and issue trackers. The pages and the features for the data administrators were the major milestone that I accomplished. I implemented a dynamic form generator class that produce a form from a schema retrieved from the backend API. The form generator is capable of handling nested questions by recursively travelling down the tree and preparing a closure. If a triggering form action, such as choosing a radio button that has nested questions attached to it, happens, then the closure will populate the nested questions. The implementation required deeper understanding in *React*, *JSX* and *Redux*. I was also able to contribute to refactoring the codebase to be more scalable and expandable using the knowledge that I acquire while implementing the features.

The relationship between this report and my job is that the report involves different techniques to optimize the rendering speed of the application. The application contains large lists, either in a table format or a carousel card format, that needs to be updated based on the server responses and the user interactions in real time. Although, *React* has generally high performance, it can bottleneck in the situations. This report analyses different technique to improve performance of a *React* and *Redux* application. The manager of the frontend team, Mr. Bellamy, asked me if I could prepare a report for *React* and *Redux*. This report is an expanded version of what I reported to him. After the submission of this report, a final copy will be delivered to Mr. Bellamy. It would be grateful if the gains from this report make a better performing *React* application. Writing this report helped me understand what and how to measure the performance of a web based application. Also, I gained deeper understanding of what is happening under the hood of *React* which is extremely valuable for my future developments.

I am glad that I was able to understand the whole COMPASS product’s stack. I am satisfied what I learned from Bespoke Metrics In the broader scheme of things, the work that I performed improved the only product keeping clients satisfied.

# Summary

Table of Contents

[Contributions iii](#_Toc503196157)

[Summary v](#_Toc503196158)

[List of Figures vii](#_Toc503196159)

[List of Tables viii](#_Toc503196160)

[1. Introduction 1](#_Toc503196161)

[1.1 Bespoke Metrics 1](#_Toc503196162)

[1.2 COMPASS 1](#_Toc503196163)

[1.3 Frontend stack 2](#_Toc503196164)

[1.4 Scope 3](#_Toc503196165)

[2. Test Outline 4](#_Toc503196166)

[2.1 Test Environment 4](#_Toc503196167)

[2.2 Test Plan 5](#_Toc503196168)

[3. React 6](#_Toc503196169)

[3.1 Lifecycle 6](#_Toc503196170)

[3.2 Optimizing state and props 6](#_Toc503196171)

[Glossary 8](#_Toc503196172)

[References 9](#_Toc503196173)

[Appendix A webpack.config.js 10](#_Toc503196174)

# List of Figures

**No table of figures entries found.**  
This is an automatic table of contents. To use it, apply heading styles (on the Home tab) to the text that goes in your table of contents, and then update this table.
If you want to type your own entries, use a manual table of contents (in the same menu as the automatic one).

# List of Tables

Table 2-1. Supported Protocols of each Service provider…………………………………………4

Table 2-2. Azure IoT Hub Pricing…………………………………………………………………6

# Introduction

## Bespoke Metrics

Bespoke Metrics Inc. is a start-up based in the downtown Toronto. The company founded to deliver values to its clients by providing tailored data analytics. The current focus of the company is in the construction industries. A general contractor is company that manages the whole construction project. They vary in size; from one man with one truck to multi-billion dollars giants such as EllisDon and PCL Construction. General contractors (usually) hires subcontractors to do the actual construction while they manage the different subcontractors and coordinate the different stages of the construction. Every construction project, from renovating homes to building airports, comes with risks. Especially, in a mega project like building a shopping mall or renovating the Union station, a small failure from a subcontractor could cause a serious damage to the whole project, a catastrophic effect to the schedule, or a significant financial loss. As an example, if a subcontractor for roofing job could not finish its job by the deadline, then the concrete cannot be poured to form the base of the building as it might rain in the next few days. Therefore, controlling and understanding the reliability of subcontractors are one of the most important factors to conduct a successful project. Bespoke Metrics provides a tailored data analytic service, COMPASS, to the general contractors that can quantify the reliability of each subcontractor. It is similar to credit ratings in financial and insurance industries. COMPASS is publicly accessible but requires a credential.

## COMPASS

COMPASS is a contract risk management service. The main target clients are general contractors who hires multiple subcontractors to accomplish a construction project. Generally, such construction project takes many subcontractors and COMPASS helps managing the risks of each subcontractor. The current COMPASS product has three different modes; general contractor, subcontractor, and administrator. The subcontractors are required to access COMPASS and fill out the provided questionnaire. The questionnaire asks about the company’s information such as health and safety information, and financial information. All the input from the questionnaire then would be piped to a model that aggregates all the information and generate a credit rating. The credit rating then would be visible to the general contractor. The engineering team is separated into a frontend team and a backend team. The frontend team is responsible for creating a user friendly single page web application. And the backend team is responsible for a scalable REST API. This report focuses on the frontend.

## Frontend stack

***Single-page Application***

The frontend is the GUI of the COMPASS service. It is a single page web application that gets delivered to its users by CDN such as CloudFront by AWS. All of the required assets and the files that are needed to be served are bundled by *Webpack.* A single page web application is an app that works inside of a browser and does not require page reloading during use. [**1**] On the initial access to the page, the browser will load the entire application and the entire application does not require page reload to navigate to other pages. Its initial load could be slower due to the bundle size that contains entire application. However, this is becoming a smaller problem as the general internet speed is getting faster. Also, there are optimizations such as server side rendering are present to improve the initial loading speed. A single page app removes the page reloading upon navigating to other features making it feels natural to use.

***React and Redux***

The frontend of COMPASS uses *React*. *React* is a JavaScript framework library that helps creating user interfaces. It has gained popularity over the past couple years. In *React*, an HTML DOM is managed as a *component*. A *component* contains states and properties that can be used to render the DOM. A component gets re-rendered whenever there is an update to its state or properties. A DOM rendering is relatively expensive compared to computing a JavaScript object. *React* detects the state changes at the object level then update only the DOMs that are affected. It is naturally faster than manipulating the DOM itself and the DOM carries the required information as a custom attribute like *data-foobar.* A benefit of using a *component* is that it is reusable. In fact, there are thousands of open-sourced *React* *components* available online. It also allows you to write the HTML markups for each component directly inside of a JavaScript file.

The other major library that COMPASS uses is *Redux. Redux* is a predictable state container for JavaScript apps. It is inevitable that a *React component* needs to access some other component’s states and properties. This could be done through having a parent component then sharing the data with the siblings. However, it is difficult to manage chains of dependencies and passing the properties around gets tedious very quickly. There are different design patterns to make the app state manageable including MVC, Flux and Redux. The famous MVC did not scale well for Facebook’s huge codebase. The main problem for them was the bidirectional communication. [**3**] Facebook introduced Flux which is an unidirectional data flow architecture. The idea of Flux is to have *stores* for all the states and the properties required for the app. Updating the stores is only available through dispatching an action. Flux enforces unidirectional data flow and eliminates complex state managements. In *Redux*, the library reduces the number of the stores to one. To use *Redux*, reducers have to be provided to the store. Each reducer is a pure function. Since a pure function cannot depend on reading any hidden value outside of the function scope, such as another field in the store or any other global variable, *Redux* can support helpful features like time-travel debugging. [**4**]

## Scope

This report will take a step into the practical usage of *React* and analyse its performance. Although React is a high performant library, its performance could only be maximized when the implementation is in lined with the library. The result of performance tests in different methods of updating the component state will help us to analyse how to use the library properly. Also, this report will cover how *Redux* can be optimized to have a better performance with *React.* COMPASS is using version 15 of *React.* Recently, version 16 has been released with a major update in its rendering engine which is expected to have much higher performance. The analysis of the performance improvement would be included.

# Test Outline

## Test Environment

The testing environment is setup on the author’s personal laptop. The result is subject to change based on the machine spec and the settings. Therefore, the results will focus on the relative differences rather than actual number.

***NPM***

NPM (Node Package Manager) is used to manage all the dependencies of the testing environment. It supports installing a specific version of a library by executing npm install <library\_name>@<semver>

***Webpack***

Webpack is used to bundle the testing components. The minifying feature to reduce the bundle size is unused since the bundle size is out of the scope of this report. Also, none of the performance optimization plugins are included to keep the variables minimum. The webpack configuration is available in Appendix A of this report.

***ECMAScript 6 (ES6)***

The test environment uses ES6 syntax to reflect COMPASS’s ES6 syntax. ES6 introduced a set of syntactic sugars for the developers to use. The test environment uses features like arrow function, class, and destructing assignments. This report will focus on the impact of the rest parameter and destructing assignments. A transpiler, *babel,* is used to support ES6.

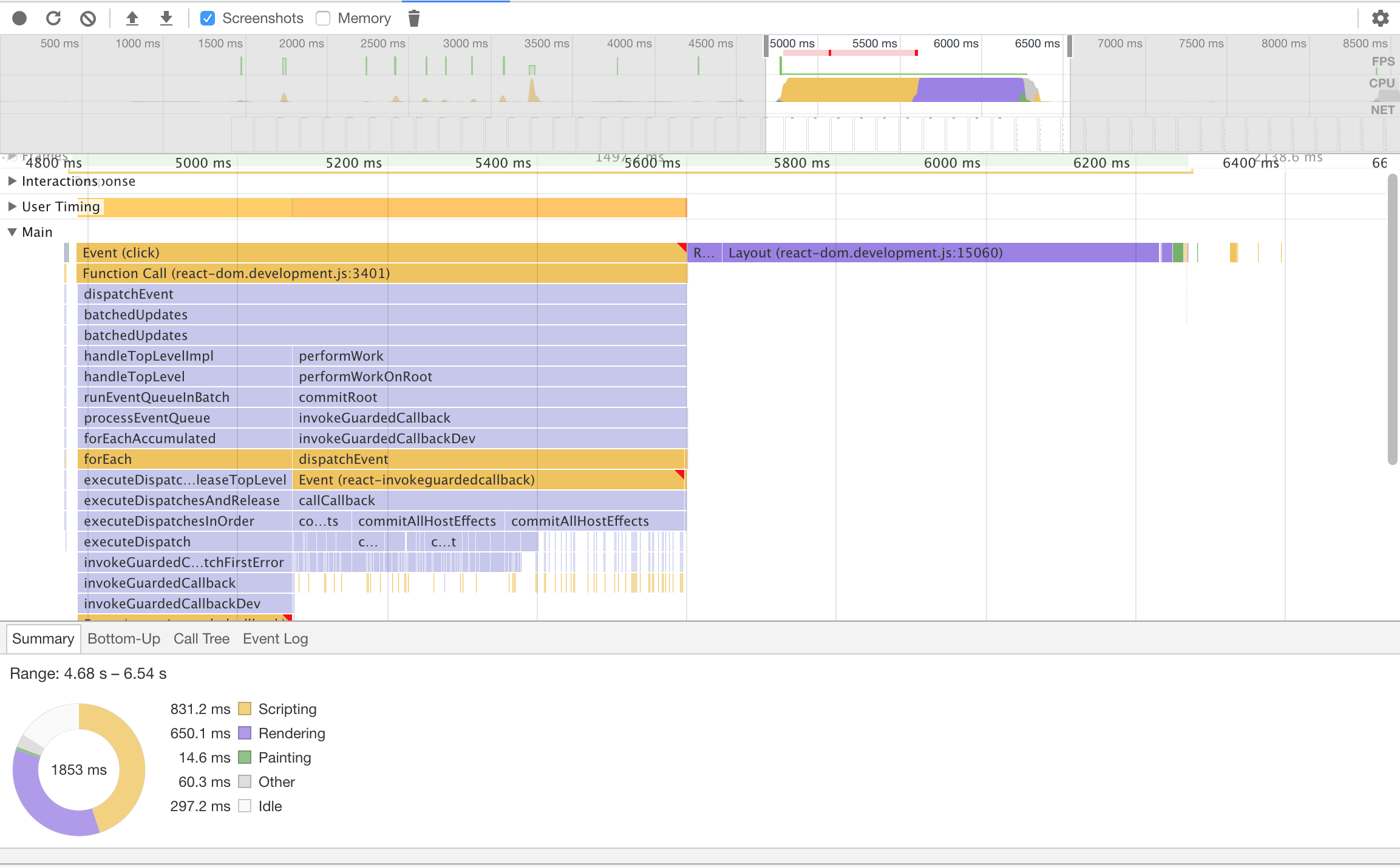


Figure 1 The screenshot of the performance tool of Chrome

***Chrome Developer Tools***

Chrome has a collection of powerful tools helpful for the developers. This report especially focuses on the *performance* tab of the developer tools suite. The performance tab supports profiling of the application allowing the detailed information to be collected. It is also useful to detect bottlenecks throughout the applications. Figure 1 represents a typical view and where the measured data collected from. The data of interest is the scripting time and the rendering time.

## Test Plan

The report starts with the pure *React* optimization. It will render a list of elements (rows) that is clickable. The element will change its color once it is clicked. The change of color is simply a toggling between on/off. The test subjects are as follows:

1. Rendering time with the state of the whole list is contained in the element
2. Rendering time with the state of the list is stored in each row
3. Impact of using …(rest) operator
4. Impact of using shouldComponentUpdate() function to pick only what’s needed
5. Impact of extending React.PureComponent rather than React.Component

In the next part, an analysis of the performance when using *React* with *Redux.* COMPASS barely uses *React* without *Redux.* The tests are similar to the pure *React* tests.

1. Rendering time with the reducer updating the whole list
2. Rendering time with having a sub-reducer for each item in the list
3. Optimizing with mapStateToProps()

Lastly, this report includes a performance comparison between version 15 and version 16 of COMPASS. Version 16 of *React* is also called as *React Fiber* which includes a major update in the engine promising improvements in the rendering performance. This test case is denoted as case (*i)*.

# React

## Lifecycle

React allows componentization of a DOM element. The created components could easily be used in other component, making it possible to compose an application at a small piece by piece. Each *component* has a lifecycle from its creation to the removal. The chart on Figure 2 shows the change of lifecycle state when an operation is applied to a component. The black node in the centre of the figure represent “normal” state. Every time there is a change in the component’s props or state, component would go through the life cycles on the right side of the black node. A component is simply a JavaScript object; hence the computation is relatively cheaper compared to directly working with the DOM. According to the official *React* document re-rendering of a DOM element could be asynchronous. It is part of an optimization that *React* does out of the box. *React* tries to minimize the number of times that ReactDOM.render() gets called. Some high speed real time operations would be batched together (e.g. burst of clicks or repeated keystrokes) and re-render the DOM only once. Naturally, it is important to pay attention to the states and the properties as they dictate how components are being rendered.

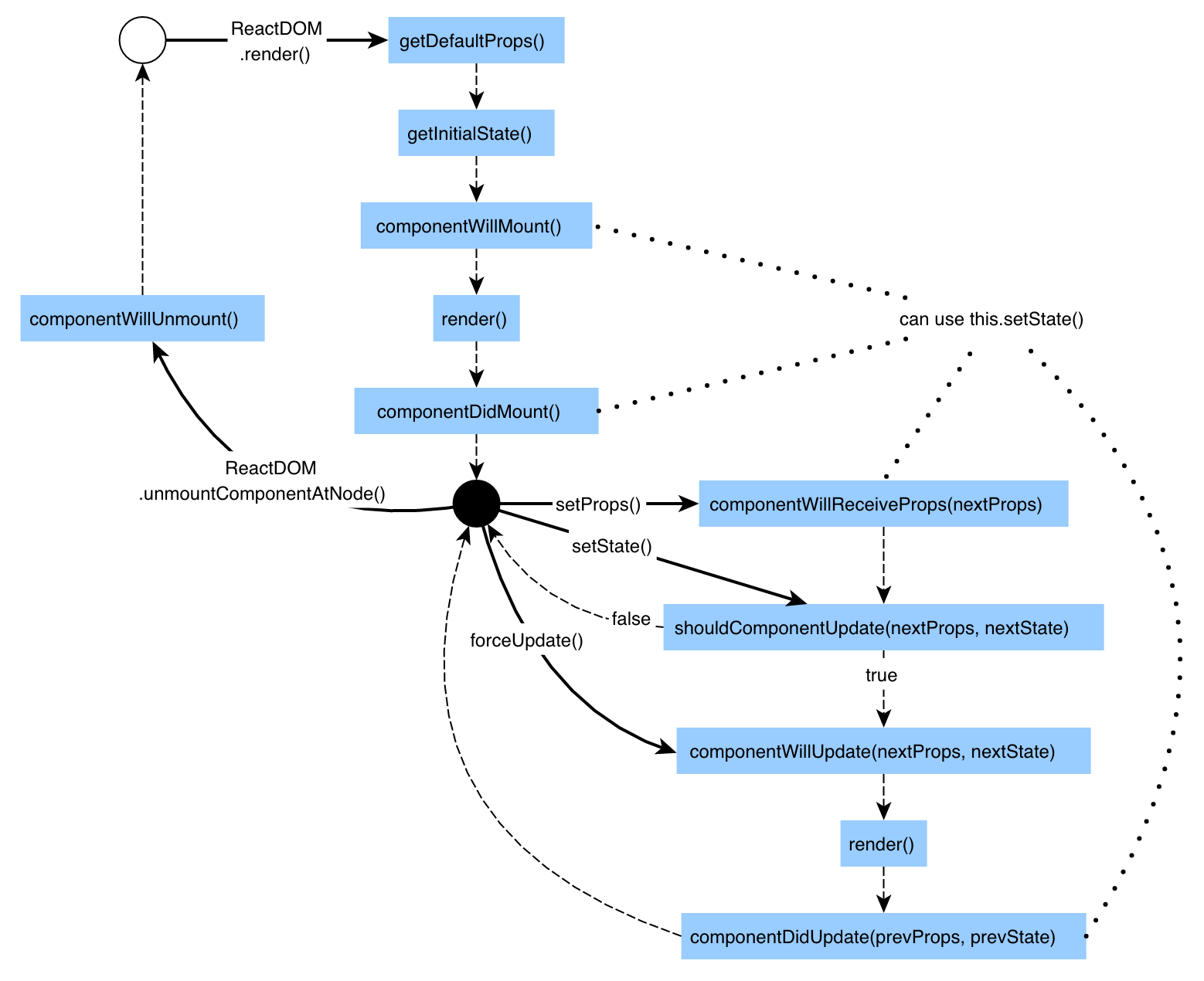


Figure 2. React Component Lifecycle [**2**]

## Optimizing state and props

As it is shown in the life cycle chart, only setProps() and setState() trigger render(). Therefore, it is important to call the functions when it is necessary to update the DOM. In this section, the first two test cases are discussed; case a and b. To share a state between components in *React*, it is necessary to have a shared parent. The shared parent contains the state that needs to be shared and distributes the state as a property to the children components. The first test case (a) is when the operation of a child component is shared as the parent’s state.

The test is conducted as follows:

1. Generate a list of elements. To test the impact of having multiple elements, the length of the list ranges from 1 to 10,000.
2. Record the initial rendering and scripting time.
3. Update (click) one of the elements to change the state.
4. Record the rendering time and the scripting time.
5. Update the same element to toggle it back to original state.
6. Record the rendering time and the scripting time.

The values are average of 3 different attempts. The detail measurement data is available in Appendix B on Table B-1.

Table 1. Result of test case (a)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Length of the list | initial load (ms) | Toggle on (first click) | | Toggle off (second click) | |
| scripting (ms) | rendering (ms) | scripting (ms) | rendering (ms) |
| 1 | 15.7 | 3.2 | 0.23 | 2.4 | 0.3 |
| 100 | 21.2 | 10.7 | 0.6 | 10.2 | 0.5 |
| 10000 | 1291.9 | 360.5 | 20.7 | 342 | 16.7 |

# Glossary

GUI: Graphical User Interface

CDN: Content Delivery Network

DOM: Document Object Model

AWS: Amazon Web Services

MVC: Model-View-Controller

GC: General Contractor

# References

[1] Neoteric. Single-Page Application vs. Multiple-page application. [Online].  
 <https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application->  
 2591588efe58

[2] T. McGinnis, An Introduction to Life Cycle Events in React. [Online]  
 <https://tylermcginnis.com/an-introduction-to-life-cycle-events-in-react-js>

[3] A. Salihefendic, Flux vs. MVC (Design Patterns). [Online]  
 <https://medium.com/hacking-and-gonzo/flux-vs-mvc-design-patterns-57b28c0f71b7>

[4] A. Alexander, “The Definition of Pure Function,” in *Functional Programming Simplified,*  CreateSpace Independent Publishing, 2017, pp. 31-32

# Appendix A webpack.config.js

1. const path = require('path');
2. const webpack = require('webpack');
3. const CleanWebpackPlugin = require('clean-webpack-plugin');
4. module.exports = {
5. entry: "./src/index.js",
6. devtool: "source-map",
7. plugins: [new CleanWebpackPlugin(["build"]), new webpack.DefinePlugin({
8. "process.env.NODE\_ENV": JSON.stringify("development")
9. })],
10. module: {
11. loaders: [{
12. test: /\.js$/,
13. exclude: path.join(process.cwd(), "node\_modules"),
14. include: path.join(process.cwd(), "src/"),
15. loaders: [{
16. loader: "babel-loader"
17. }]
18. }]
19. },
20. output: {
21. path: path.resolve(process.cwd(), "build"),
22. filename: "application.js"
23. }
24. };

# Appendix B Test Results in Detail

Table B-1. Test case (a)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Length of the list | Trial | initial load (ms) | Toggle on (first click) | | Toggle off (second click) | |
| scripting (ms) | rendering (ms) | scripting (ms) | rendering (ms) |
| 1 | #1 | 18.4 | 2.8 | 0.2 | 1.8 | 0.2 |
| #2 | 11.9 | 3.6 | 0.3 | 3.2 | 0.4 |
| #3 | 16.8 | 3.3 | 0.2 | 2.3 | 0.3 |
| Average | 15.7 | 3.2 | 0.23 | 2.4 | 0.3 |
| 100 | #1 | 22.1 | 14 | 0.5 | 8.5 | 0.6 |
| #2 | 20.1 | 8 | 0.7 | 10.3 | 0.5 |
| #3 | 21.4 | 10.1 | 0.6 | 11.9 | 0.4 |
| Average | 21.2 | 10.7 | 0.6 | 10.2 | 0.5 |
| 10000 | #1 | 1374.6 | 451.1 | 22 | 326.9 | 16 |
| #2 | 1211.7 | 313.5 | 19 | 313.1 | 17.1 |
| #3 | 1289.4 | 317 | 21 | 386 | 17 |
| Average | 1291.9 | 360.5 | 20.7 | 342 | 16.7 |