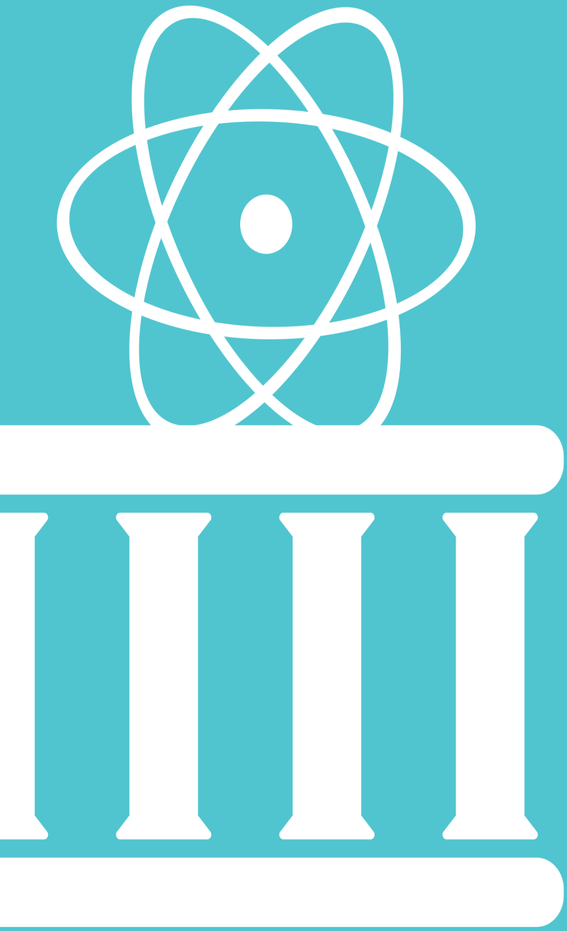
Foundations of *High-Performance* React Applications



by

Thomas Hintz

Foundations of High-Performance React Applications

Thomas Hintz

This book is for sale at http://leanpub.com/foundations-high-performance-react

This version was published on 2021-05-09

Contents

Preface

Welcome to *Foundations of High-Performance React Applications* where we build our own simplified version of React. We'll use our React to gain an understanding of the real React and how to build high-performance applications with it.

This book is based on the first chapter of the book *High-Performance React*. If you enjoy this book and you want to learn more practical ways to utilize the foundations we'll learn here and get a more detailed blueprint for creating high performance React applications, then be sure to check out *High-Performance React*.

This book is not intended to be an introduction to React or JavaScript. While it might be useful to beginners, this book assumes familiarity with both JavaScript and React.

And while this book only specifically addresses React-DOM, the foundations apply equally to React-Native and other React implementations because they are all based on the same core React library and algorithms.

The code in this book is clear and simple so as to best communicate the algorithms we'll be exploring. It is not intended to be used in production, but it is functional. I think you'll likely find it useful to follow along by writing the code yourself. It will help you better understand how it works, and even more critically, it will allow you to play with it and test how the algorithms work with your own examples.

Even if you don't write out the code yourself and, instead, read through this book more like a novel, I believe the fundamentals will still stick with you and provide value in your React programs-to-come.

I'm very excited to take you on this journey with me and, so, now it's time to learn what lies at the very foundation of React.

Acknowledgments

Introduction

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/foundations-high-performance-react.

Components of React

Markup in JavaScript: JSX

Getting Ready to Render with createElement

Render: Putting Elements on the Screen

There are now only two major puzzles remaining in our quest for our own React. The next piece is `render`. How do we go from our JSM tree of nodes to actually displaying something on screen? We do this by exploring the `render` method.

The signature for our `render` method should be familiar to you:

```
function render(element, container)
```

This is the same signature as that of React itself. We begin by just focusing on the initial render. In pseudocode it looks like this:

```
function render(element, container) {
   const domElement = createDOMElement(element);
   setProps(element, domElement);
   renderChildren(element, domElement);
   container.appendChild(domElement);
```

Our DOM element is created first. Then we set the properties, render the children elements, and finally append the whole tree to the container.

Now that we have an idea of what to build we'll work on expanding the pseudocode until we have our own fully functional `render` method by using the same general algorithm that React uses. In our first pass we'll focus on the initial render and ignore reconciliation.

Reconciliation is basically React's "diffing" algorithm. We'll be exploring it after we work out the initial render.

```
function render(element, container) {
   const { type, props } = element;

   // create the DOM element
   const domElement = type === 'TEXT' ?
     document.createTextNode(props.nodeValue) :
     document.createElement(type);

   // set its properties
   Object.keys(props)
     .filter((key) => key !== 'children')
     .forEach((key) => domElement[key] = props[key]);

   // render its children
   props.children.forEach((child) =>
      render(child, domElement));

   // add our tree to the DOM!
   container.appendChild(domElement);
}
```

The `render` method starts by creating the DOM element. Then we need to set its properties. To do this we first need to filter out the `children` property and then we simply loop over the keys, setting each property directly. Following that, we render each of the children by looping over them and recursively calling `render` on each child with the `container` set to the current DOM element (which is each child's parent).

Now we can go all the way from our JSX-like notation to a rendered tree in the browser's DOM! But so far we can only add things to our tree. To be able to remove and modify the tree we need one more part: reconciliation.

Reconciliation, or How React Diffs

Fibers: Splitting up Render

Putting it all together

Now that we've explored how React renders your components, it's time to finally create some components and use them!

```
const SayNow = ({ dateTime }) => {
   return ['h1', {}, [`It is: ${dateTime}`]];
};

const App = () => {
   return ['div', { 'className': 'header' },
           [SayNow({ dateTime: new Date() }),
            ['input',
             { 'type': 'submit', 'disabled': 'disabled' },
             []]
           ]
          ];
}

render(createElement(App()),
       document.getElementById('root'));
```

We are creating two components that output JSM, as we defined it earlier. We create one component prop for the `SayNow` component: `dateTime`. It gets passed from the `App` component. The `SayNow` component prints out the `DateTime` passed in to it. You might notice that we are passing props the same way one does in the real React, and it just works!

The next step is to call render multiple times.

```
setInterval(() =>
   render(createElement(App()),
          document.getElementById('root')),
   1000);
```

If you run the code above you'll see the DateTime display being updated every second. And if you watch in your dev tools, or if you profile the run, you'll see that the only part of the DOM that gets updated or replaced is the part that changes (aside from the DOM props). We now have a working version of our own React.

This implementation is designed for teaching purposes and has some known bugs, like always updating the DOM props, along with other issues. Fundamentally, it functions the same as React, but if you want to use it in a more production-like setting, it would take a lot more development.

Conclusion