CHAPTER 2

Understanding React

React is a flexible and powerful open-source framework for developing client-side applications; it takes cues from the world of server-side development and applies them to HTML elements, and it creates a foundation that makes building rich web applications easier. In this book, I explain how React works and demonstrate the different features it provides.

THIS BOOK AND THE REACT RELEASE SCHEDULE

The React team makes frequent releases, which means there is an ongoing stream of fixes and features. Minor releases tend not to break existing features and largely contain bug fixes. The major releases can contain substantial changes and may not offer backward compatibility.

It doesn't seem fair or reasonable to ask readers to buy a new edition of this book every few months, especially since the majority of React features are unlikely to change even in a major release. Instead, I am going to post updates following the major releases to the GitHub repository for this book, `https://github.com/Apress/pro-react-16`.

This is an ongoing experiment for me (and for Apress), and I don't yet know what form those updates may take—not least because I don't know what the major releases of React will contain—but the goal is to extend the life of this book by supplementing the examples it contains.

I am not making any promises about what the updates will be like, what form they will take, or how long I will produce them before folding them into a new edition of this book. Please keep an open mind and check the repository for this book when new React versions are released. If you have ideas about how the updates could be improved, then e-mail me at adam@adam-freeman.com and let me know.

Should I Use React?

React isn't the solution to every problem, and it is important to know when you should use React and when you should seek an alternative. React delivers the kind of functionality that used to be available only to server-side developers but is delivered entirely in the browser. The browser has to do a lot of work each time an HTML document to which React has been applied is loaded: data has to be loaded, components have to be created and composed, expressions have to be evaluated, and so on, creating the foundation for the features that I described in Chapter 1 and those that I explain throughout the rest of this book.

This kind of work takes time to perform, and the amount of time depends on the complexity of the React application and—critically—on the quality of the browser and the processing capability of the device.

A. Freeman, *Pro React 16*, https://doi.org/10.1007/978-1-4842-4451-7_2

You won't notice any performance issues when using the latest browsers on a capable desktop machine, but old browsers on underpowered smartphones can really slow down the initial setup of a React application.

The goal, therefore, is to perform this setup as infrequently as possible and deliver as much of the app as possible to the user when it is performed. This means giving careful thought to the kind of web application you build. In broad terms, there are two basic kinds of web application: *round-trip* and *single-page*.

Understanding Round-Trip Applications

For a long time, web apps were developed to follow a *round-trip* model. The browser requests an initial HTML document from the server. User interactions—such as clicking a link or submitting a form—leads the browser to request and receive a completely new HTML document. In this kind of application, the browser is essentially a rending engine for HTML content, and all of the application logic and data resides on the server. The browser makes a series of stateless HTTP requests that the server handles by generating HTML documents dynamically.

A lot of current web development is still for round-trip applications, especially for line-of-business projects, not least because they put few demands on the browser and have the widest possible client support. But there are some serious drawbacks to round-trip applications: they make the user wait while the next HTML document is requested and loaded, they require a large server-side infrastructure to process all the requests and manage all the application state, and they can require more bandwidth because each HTML document has to be self-contained, which can lead to the same content being included in each response from the server. React is not well-suited to round-trip applications because the browser has to perform the initial setup process for each new HTML document that is received from the server.

Understanding Single-Page Applications

Single-page applications (SPAs) take a different approach. An initial HTML document is sent to the browser, but user interactions lead to HTTP requests for small fragments of HTML or data inserted into the existing set of elements being displayed to the user. The initial HTML document is never reloaded or replaced, and the user can continue to interact with the existing HTML while the HTTP requests are being performed asynchronously, even if that just means seeing a "data loading" message.

React is well-suited to single-page applications because the work that the browser has to perform to initialize the application has to be performed only once, after which the application runs in the browser, responding to user interaction and requesting the data or content that is required in the background.

COMPARING REACT TO VUE.JS AND ANGULAR

There are two main competitors to React: Angular and Vue.js. There are differences between them, but, for the most part, all of these frameworks are excellent, all of them work in similar ways, and all of them can be used to create rich and fluid client-side applications.

The real difference between these frameworks is the developer experience. Angular requires you to use TypeScript to be effective, for example, whereas it is just an option with React and Vue.js projects. React and Vue.js mix HTML and JavaScript together in a single file, which not everyone likes, although the way this is done differs for each framework.

My advice is simple: pick the framework that you like the look of the most and switch to one of the others if you don't get on with it. That may seem like an unscientific approach, but there isn't a bad choice to make, and you will find that many of the core concepts carry over between frameworks even if you change the one you use.

Understanding Application Complexity

The type of application isn't the only consideration when deciding whether React would be well-suited to a project. The complexity of a project is also important, and I often from readers who have embarked on a project using a client-side framework such as React, Angular, or Vue.js, when something much simpler would have been sufficient. A framework such as React requires a substantial time commitment to master (as the size of this book illustrates), and this effort isn't justified if you just need to validate a form or populate a `select` element programmatically.

In the excitement that surrounds client-side frameworks, it is easy to forget that browsers provide a rich set of APIs that can be used directly and that these are the same APIs that React relies on for all of its features. If you have a problem that is simple and self-contained, then you should consider using the browser APIs directly, starting with the Document Object Model (DOM) API. You will see that some of the examples in this book use the browser APIs directly, but a good place to start if you are new to browser development is `https://developer.mozilla.org`, which contains good documentation for all of the APIs that browsers support.

The drawback of the browser APIs, especially the DOM API, is that they can be awkward to work with and older browsers tend to implement features differently. A good alternative to working directly with the browser APIs, especially if you have to support older browsers, is jQuery (`https://jquery.org`). jQuery simplifies working with HTML elements and has excellent support for handling events, animations, and asynchronous HTTP requests.

React comes into its own in large applications, where there are complex workflows to implement, different types of users to deal with, and substantial volumes of data to be processed. In these situations, you can work directly with the browser APIs, but it becomes difficult to manage the code and hard to scale up the application. The features provided by React make it easier to build large and complex applications and to do so without getting bogged down in reams of unreadable code, which is often the fate of complex projects that don't adopt a framework.

What Do I Need to Know?

If you decide that React is the right choice for your project, then you should be familiar with the basics of web development, have an understanding of how HTML and CSS work, and have a working knowledge of JavaScript. If you are a little hazy on some of these details, I provide primers for the features I use in this book in Chapters 3 and 4. `https://developer.mozilla.org` is a good place to brush up on the fundamentals of HTML, CSS, and JavaScript.

How Do I Set Up My Development Environment?

The only development tools needed for React development are the ones you installed in Chapter 1 when you created your first application. Some later chapters require additional packages, but full instructions are provided. If you successfully built the application in Chapter 1, then you are set for React development and for the rest of the chapters in this book.

What Is the Structure of This Book?

This book is split into three parts, each of which covers a set of related topics.

Part 1: Getting Started with React

Part 1 of this book provides the information you need to get started with React development. It includes this chapter and primer/refresher chapters for the key technologies used in React development, including HTML, CSS, and JavaScript. Chapter 1 showed you how to create a simple React application, and Chapters 5–8 take you through the process of building a more realistic application, called SportsStore.

Part 2: Working with React

Part 2 of this book covers the core React features that are required in most projects. React provides a lot of built-in functionality, which I describe in depth, along with the way that custom code and content is added to a project to create bespoke features.

Part 3: Creating Complete React Applications

React relies on additional packages to provide the advanced features that are required by most complex applications. In Part 3 of this book, I introduce the most important of these packages, show you how they work, and explain how they add to the core React features.

Are There Lots of Examples?

There are *loads* of examples. The best way to learn React is by example, and I have packed as many of them into this book as I can, along with screenshots so you can see the effects of each feature. To maximize the number of examples in this book, I have adopted a simple convention to avoid listing the same code or content over and over. When I create a file, I will show its full contents, just as I have in Listing 2-1. I include the name of the file and its folder in the listing's header, and I show the changes that I have made in bold.

Listing 2-1. Using a Callback in the SimpleButton.js File in the src Folder

```
import React, { Component } from "react";

export class SimpleButton extends Component {

    constructor(props) {
        super(props);
        this.state = {
            counter: 0,
            hasButtonBeenClicked: false
        }
    }

    render = () =>
        <button onClick={ this.handleClick }
            className={ this.props.className }
            disabled={ this.props.disabled === "true"
                    || this.props.disabled === true  }>
                { this.props.text} { this.state.counter }
                { this.state.hasButtonBeenClicked &&
                    <div>Button Clicked!</div>
                }
                </button>
    }

    handleClick = () => {
        this.setState({ counter: this.state.counter + 1 },
            () => this.setState({ hasButtonBeenClicked: this.state.counter > 0 }));
        this.props.callback();
    }
}
```

This is a listing from Chapter 11, which shows the contents of a file called `SimpleButton.js` that can be found in the `src` folder. Don't worry about the content of the listing or the purpose of the file; just be aware that this type of listing contains the complete contents of a file and that the changes you need to make to follow the example are shown in bold.

Some files in a React application can be long, but the feature that I am describing requires only a small change. Rather than list the complete file, I use an ellipsis (three periods in series) to indicate a partial listing, which shows just part of the file, as shown in Listing 2-2.

Listing 2-2. Making Multiple Updates in the SimpleButton.js File in the src Folder

```
...
handleClick = () => {
    for (let i = 0; i < 5; i++) {
        this.setState({ counter: this.state.counter + 1});
    }
    this.setState({ hasButtonBeenClicked: true });
    this.props.callback();
}
...
```

This is a later listing from Chapter 11, and it shows a set of changes that are applied to only one part of a much larger file. When you see a partial listing, you will know that the rest of the file does not have to change and that only the sections marked in bold are different.

In some cases, changes are required in different parts of a file, which makes it difficult to show as a partial listing. In this situation, I omit part of the file's contents, as shown in Listing 2-3.

Listing 2-3. Implementing a Lifecycle Method in the Message.js File in the src Folder

```
import React, { Component } from "react";
import { ActionButton } from "./ActionButton";

export class Message extends Component {

    // ...other methods omitted for brevity...

    componentDidMount() {
        console.log("componentDidMount Message Component");
    }

    componentDidUpdate() {
        console.log("componentDidUpdate Message Component");
    }
}
```

The changes are still marked in bold, and the parts of the file that are omitted from the listing are not affected by this example.

Where Can You Get the Example Code?

You can download the example projects for all the chapters in this book from `https://github.com/Apress/pro-react-16`. The download is available without charge and contains everything that you need to follow the examples without having to type in all of the code.

Where Can You Get Corrections for This Book?

You can find errata for this book at `https://github.com/Apress/pro-react-16`.

How Can You Contact Me?

If you have problems making the examples in this chapter work or if you find a problem in the book, then you can e-mail me at adam@adam-freeman.com, and I will try my best to help. Please check the errata for this book to see whether it contains a solution to your problem before contacting me.

Summary

In this chapter, I explained when React is a good choice for projects and outlined the alternatives and competitors. I also outlined the content and structure of this book, explained where to get updates, and explained how to contact me if you have problems with the examples in this book. In the next chapter, I provide a primer for the HTML and CSS features that I use in this book to explain React development.