Understanding React

The Simplest Practical Guide to Start Coding in React


Includes:
React **Hooks** and React **Router**

Enrique Pablo Molinari

Contents

About the Author

My name is Enrique Pablo Molinari. I have been working in the software industry for the last 22 years, working in different software projects from different companies as developer, technical lead and architect. I'm a passionate developer and also a passionate educator. In addition to my work on the software industry, I'm teaching Object Oriented Design and Advance Database Systems at Universidad Nacional de Río Negro.

Understanding React is my second book. I have also written Coding an Architecture Style, a book about hands-on software architecture. You can find more about my thoughts on software development at my blog: Copy/Paste is for Word. I would be very happy if you want to ping me by email at enrique.molinari@gmail.com to send thoughts, comments or questions about this book, the other or my blog.

What is this book about?

Every successful framework or library provides something unique which gives developers a new tool for writing better software. In the case of React, that tool is called **component**. You might be thinking that you have been reading about components as the solution to your spaghetti software nightmare for the last 15 years without any success. You are not wrong. However, React is an exception. It provides the constructions and tools to build highly cohesive components to assemble your next application. In this book, we will study React core concepts, to end up being very practical describing how to split an application into components and to fully implement it. But before that, it is necessary to study some Javascript concepts. Understanding these concepts will make you a better React developer. If you are already a Javascript developer, then you can just ignore the initial chapter. But if your experience is mainly on server side programming languages like Java, C#, PHP, etc, the initial chapter will give you the necessary basis.

Development Environment

There are many development environments out there, and you can choose the one you are more comfortable with. In any case if you don't have a preference, I recommend Visual Studio Code (VS Code). And to be more productive, especially if you are new to React, I suggest installing the extension VS Code ES7 React/Redux/React-Native/JS snippets which provides JavaScript and React snippets. I would also suggest installing Prettier, which is a JavaScript/React code formatter.

To install an extension, in Visual Studio Code, go to the File menu, then Preferences and then Extensions. You will see a search box that will allow you to find the extensions that you want to install.

Finally, I really recommend configuring VS Code to format your source files on save. You can do that by going to the File menu, then Preferences and then Settings. On the search box type Editor: Format On Save. This will format your code right after you save it.

Part I

Introduction

Chapter 1

Essential JavaScript Concepts

You can use React just by learning from React docs and you will also be able to build applications, without digging into JavaScript. However, if you want to master React and that means, understand why and how certain things work, you must learn some specific concepts from JavaScript.

In this chapter we will explain those JavaScript concepts and syntactical constructions needed to make a solid learning path to React. If you want to dig in more details on some of the topics explained here or others about JavaScript I recommend to visit the Mozilla[1] web site. Indeed, this section is based on learning paths and ideas taken from there. Having said that, let's begin.

From the Developer Mozilla JavaScript Documentation [1] JavaScript is defined as:

"JavaScript (JS) is a lightweight, interpreted, or just-in-time compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat. JavaScript is a prototype-based, multi-paradigm, single-threaded, dynamic language, supporting object-oriented, imperative, and declarative styles."

If you are a Java, C# or C++ developer that definition might sound a bit intimidating. The thing is that you do have to learn some concepts. Especially those that are not available in compiled languages (if your experience comes from there). To start with these concepts we will first explain basic language constructions and with that in place we will explain what it means for a language to have **first-class functions** and to be **prototype-based**, **multi-paradigm**, **single-threaded** and **dynamic**.

THIS IS A SAMPLE

Part II

Understanding React

Chapter 2

Essential React Concepts

In this chapter we will learn the core concepts behind react. With different examples and with a different approach, we mainly follow the learning path suggested by the official docs[2], which is fantastic.

2.1 React Principles

I have to start saying that I *love* React! I love it because I love designing software in a professional way. With that I mean with practices and mainly with syntactical constructions that help keeping application's source code *modifiable* after several years of iterations. We say a software is modifiable, if I know on every change, where that change will impact. In React you design applications by assembling **components**, which are built by using plain JavaScript classes or functions. The guys behind React's design decisions have challenged very established patterns like MVC, where you have the display logic and the markup separated in different abstractions. They say that these elements are naturally coupled. In React you have the display logic (fetching data, event handling, etc) and the markup in the same abstraction: the component. And each component represents a fragment of functionality of your View. This is what makes React so great and the reason I love it. And among other features provided by React, understanding how to build applications by assembling components is the main goal of this book.

I really recommend you to see the explanation about react design principles by Pete Hunt: React - Rethinking Best Practices.

THIS IS A SAMPLE

2.2 React Components

React applications are built by creating components. Component is probably one of the most confusing terms in software engineering. So, we will provide our definition of what a component is in React. A React component is a self contained piece of functionality that is created by using a plain JavaScript `class` or `function`. It manages its own state and ideally performs a single task. It might collaborate with other components and knows how to paint itself on the browser.

In the previous paragraph I said that React components know how to paint itself in the browser and they can be defined using the `class` or the `function` syntactical constructions. If you use a `class` you have to add a method called `render`, which is the method invoked by React when the component requires it to be painted in the browser. And if you use a `function`, it is the function that gets invoked by React when the component requires it to be painted in the browser.

Let's create our first React component using a JavaScript `class`:

```
1  import { Component } from "react";
2
3  export default class Person extends Component {
4    render() {
5      return <p>This is a <strong>Person</strong>
       ↪  Component</p>;
6    }
7  }
```

You first need to `import` the `Component` class from React Core, because your JavaScript classes must extend from it. And finally you have to define the `render` method that is invoked to paint the component on the browser. Note that this method just returns HTML (or at least looks like HTML as we will see later). Below we create the same component as the one above but using a `function` instead of a class:

```
1  export default function Person() {
2    return <p>This is a <strong>Person</strong> Component</p>;
3  }
```

2.3 Rendering Components

I have my first component, how do I get it rendered into the DOM? To answer this question we will learn some React concepts.

Take a look again at the component we have created in section 2.2, the "HTML" snippet that the function component returns (or the render method in the class-based component) is not really HTML. It is called JSX, which stands for **JavaScript XML**. It is a *syntax extension* to JavaScript and what the React team recommends to use to paint your components on the browser.

In JSX, you can embed any valid JavaScript Expression between curly braces. As an example, below you can see the variable `name` declared and initialised (on line 2) and used inside the JSX syntax (line 6).

```
1  export default function Animal() {
2    let name = "Eze the Dog";
3
4    return (
5      <p>
6        This is <strong>{name}</strong>
7      </p>
8    );
9  }
```

Browsers do not understand JSX syntax. To make it work we have to translate JSX into JavaScript, using a compiler like Babel. If you create your React application using the `create-react-app` tool like we did before, you get this covered without needing to deal with it.

JSX gets translated into a JavaScript expression which at execution time, is evaluated along with the expressions defined by you in curly braces and painted on the browser (injecting in the DOM). As JSX are expressions, it is possible to see a JSX piece of code as a first class object. Which allows you to do, for instance, what you see below:

```
1  function passingAsArgument(jsx) {
2    return jsx;
3  }
4
5  export default function Animal() {
6    let name = "Dog";
7    //assign a JSX block to a variable
8    let jsx = passingAsArgument(
9      <p>
10       This is a <strong>{name}</strong>
```

```
11        </p>
12      );
13
14      return jsx;
15    }
```

In the example above, we are calling a function passing a JSX expression as an argument and the return of that function (which is just this same argument) is assigned to the variable `jsx` on line 8.

As you might have noted, JSX is the tool you use in React to paint the components you create on the browser. Let's start writing some code. In the VS Code project we have created in section **??**, create a JavaScript file called `src/Person.js` with the `Person` component we have created in section 2.2. Then, open the file `src/index.js`, delete their content and paste the following:

```
1   import React from "react";
2   import ReactDOM from "react-dom";
3   import Person from "./Person";
4
5   ReactDOM.render(
6     <React.StrictMode>
7       <Person />
8     </React.StrictMode>,
9     document.getElementById("root")
10  );
```

The `src/index.js` file is our JavaScript main module (the entry point). In this main file we call to the `ReactDOM.render` function (line 5 above), which expects as their first argument a JSX expression, and as the second argument the DOM element in which the JSX expression will be rendered (injected). If you check the `public/index.html`, you will see the markup `<div id="root"></div>` we are referencing on line 9 above.

On line 7 we are telling React to render the `Person` component. That will instantiate the `Person class` and execute the `render` method, in the case of a class-based component, or execute the `Person function` in the case of a function-based component. In either case, the output is inserted into the DOM, generating what is shown below:

```
1  <div id="root">
2   <p>This is a <strong>Person</strong> Component</p>
3  </div>
```

The `div` element on line 1 above comes from the `public/index.html` and the `p`, the paragraph with the text inside, comes from rendering the `Person` component. If you start the application as we explained in the section **??**, you will see the results in the browser. You can also inspect the DOM with the browser's development tool.

Additionally, note that in the `src/index.js`, the `<Person/>` component is wrapped by the `<React.StrictMode>` component which helps us during development to inform us about potential problems with our React code. Have a look at strict mode in React official docs.

And finally, note that component names, in this case `Person`, start with a capital letter. React sees components starting with a capital letter as custom components (your components) and that requires the function or class definition to be in scope (that is why in the `src/index.js` we have to import the `Person` component). And React sees components starting with a lowercase letter as DOM tags, like `div`, `p`, `strong`, etc.

THIS IS A SAMPLE

Part III

Practical React

Chapter 3

A Simple CRUD Application

In this chapter you will learn how to build a simple CRUD application. The application will support the classic CRUD operations of users data. Along the implementation you will learn several React concepts and deal with common problems you will face when coding in React. We will see how to create and submit forms, display data in tabular format (data grids), open modals and also very specific React concepts like the children prop and conditional rendering.

To code the application we will use Material UI[6]. It provides a rich set of React components, like forms, grids, modals and many more, styled with the material design theme.

The full source code of this application is available at github/crud. We will go in detail explaining every component we have created. However, I recommend you to follow the steps here to install and run the application, in order you can play with it while reading the chapter.

THIS IS A SAMPLE

Chapter 4

Creating a Blog

In this chapter we will design and build a blog application that we call **react-blog**. The full source code is available on github following the link: react-blog. To see the react-blog application running, I have written a small back-end application that can be downloaded and installed from github too, following the link: blog-api. The blog-api backend application exposes the set of APIs required by the react-blog application. They provide the content for the blog.

To build the react-blog application I have used the html theme called Editorial from html5up.net[7]. From the original sources I have just removed JQuery, as we are only interested in the HTML markup and the CSS files.

Let's start by describing what functionality the blog application will have. Like in the previous chapter, we will present a set of figures that illustrate what we are going to build.

4.1 Identifying Components

THIS IS A SAMPLE

Bibliography

[1] https://developer.mozilla.org/en-US/docs/Web/JavaScript

[2] https://reactjs.org/docs/getting-started.html

[3] https://www.ecma-international.org/

[4] https://nodejs.org/

[5] http://latentflip.com/loupe/

[6] https://material-ui.com/

[7] https://html5up.net/