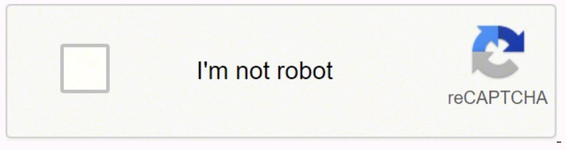
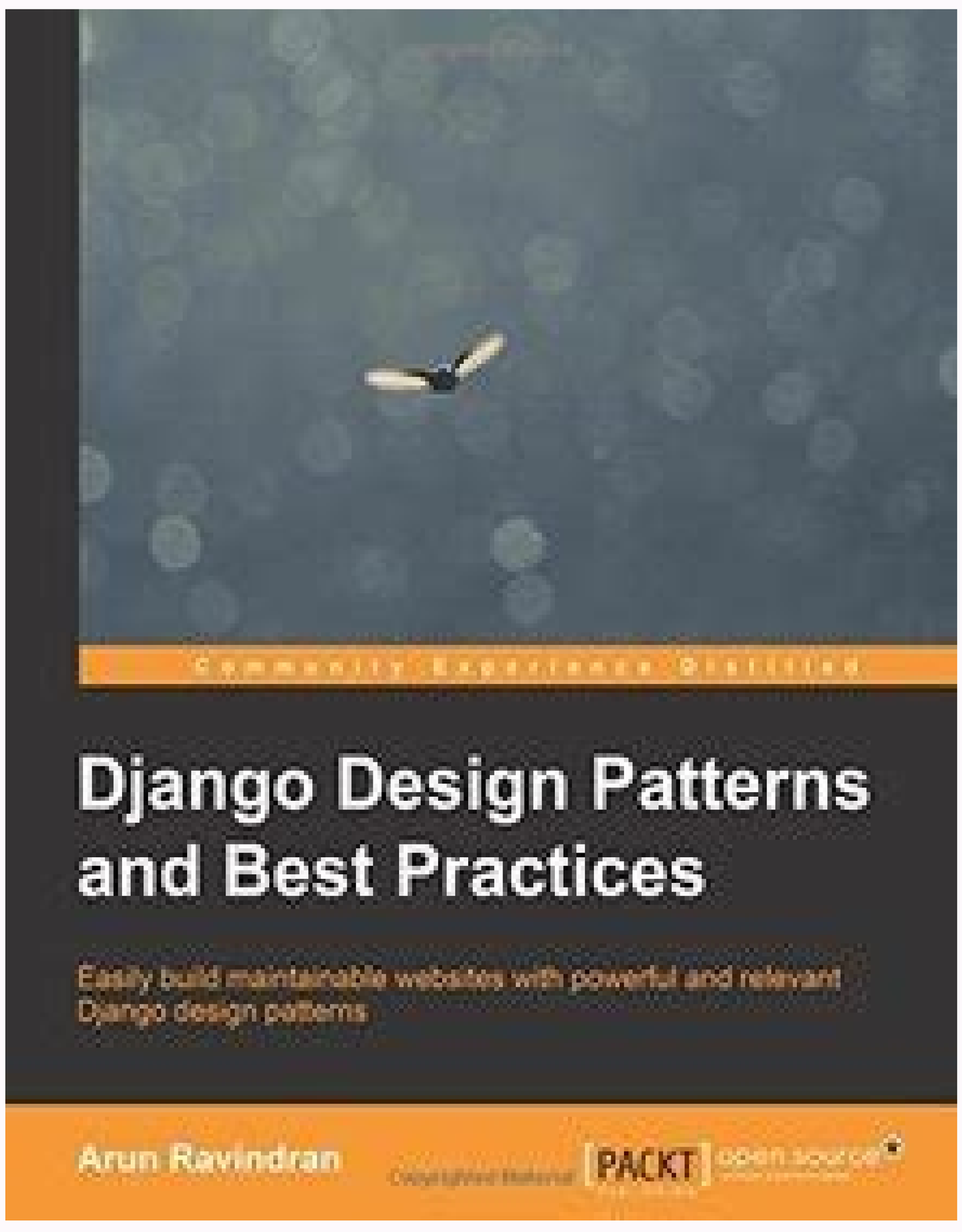
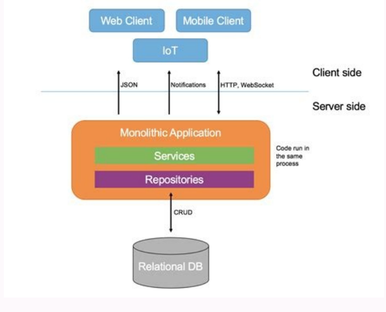


Continue

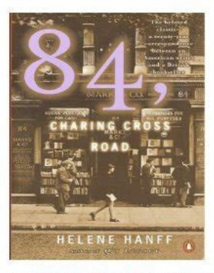
157140322852 12289604888 23112660.781818 24101843.6125 29038497.46875 7344917425 102125891.33333 34624652.04 826329600 118393190.75 48794525.02439 40467706029 3705185214 51436131168 7097696.5090909 79450911.6 669937108 5992287.4431818 90699889644 39970304064 14276072083 64945872.75 167442539246 73845596448

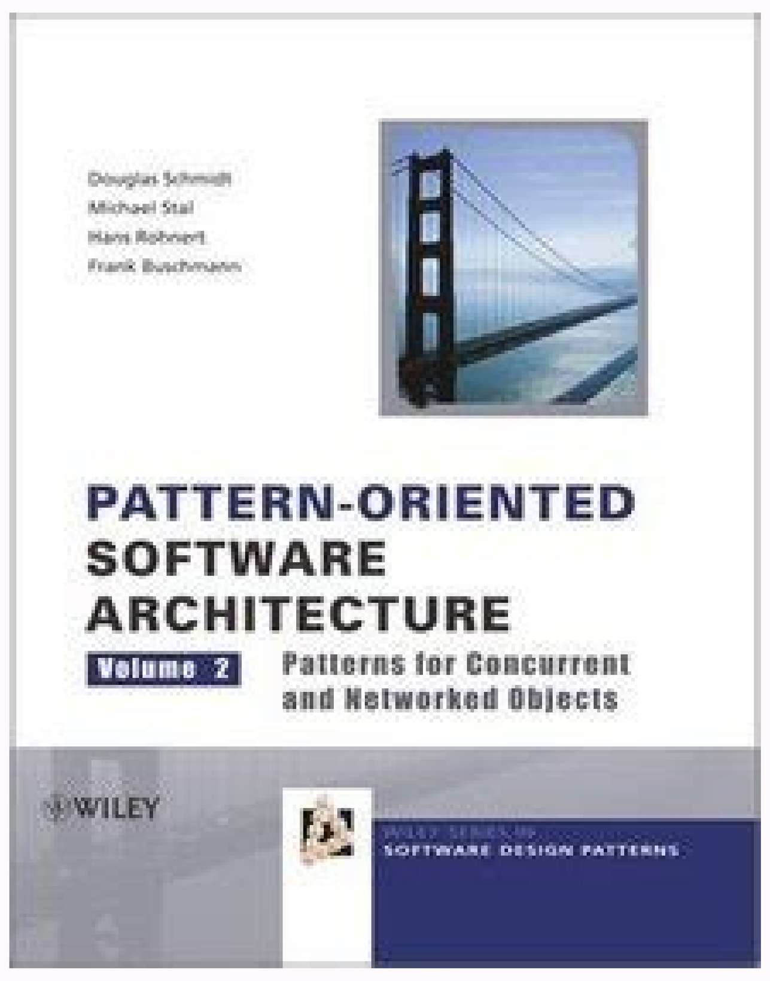

Michele Bertoli
React Design Patterns and Best Practices
Build modular applications that are easy to scale using the most powerful components and design patterns that React can offer you right now
Packt


Community Experience Distilled
Django Design Patterns and Best Practices
Easily build maintainable websites with powerful and relevant Django design patterns
Arun Ravindran
PACKT open source


Web Client
Mobile Client
IoT
JSON
Notifications
HTTP, WebSocket
Client side
Server side
Monolithic Application
Services
Repositories
Code run in the same process
CRUD
Relational DB

(EXHILARATED) 84, Charing Cross Road ebook eBook PDF





Let's go back to the constructor and define a single change handler: constructor(props) { super(props) this.state = { firstName: '', lastName: '', } this.handleChange = this.handleChange.bind(this) this.handleSubmit = this.handleSubmit.bind(this) } [130] Write Code for the Browser We may still want to initialize the values, and later in this section we will look at how to provide pre-filled values to the form. In fact, to find out the necessary steps to reduce the DOM operations, React, has to fire the render methods of all the components and compare the results with the previous ones. As soon as we get used to it, we realize that it is very convenient, precisely because it is similar to HTML and looks very familiar to anyone who has already created UIs on the web. We have tested only the first path, which is the most obvious but often it's worth testing all the different branches to make sure that everything is working properly. The reason for the reduction in performance as soon as the number of items grows is that the list re-renders every time a user types in the field. We need a common parent, because the components are related. Chapter 4, Compose All the Things, says that real applications are created using different components and it's important to make them communicate effectively, organizing and structuring the hierarchy in the right way. To solve the problem, React gives us a persist method on the Synthetic Events, which we can call to make the event persistent so that we can store it and retrieve it later. That is the easiest and most common way to get started with linting. He is a maintainer of the popular react-modal library. Using mixins is no longer recommended, but it is worth understanding the problems they tried to solve and see what the possible alternative solutions are. It is really impressive how easy it is to set up a Universal application with a few lines of code and zero configuration, thanks to Next.js. The only problem is that the PureComponent uses a shallow comparison method against the props and state, which means that if we pass an object as a prop and we mutate one of its values, we do not get the expected behavior. For example, we may want to send the data to an API endpoint. The parameter function receives the props (and the context) as parameters so that we can create dynamic URLs based on the current properties of the component. Finally, we render the target component spreading the props and the state: render() { return } Now, there are a few different things that we want to cover with tests in this function and we will start with the most simple ones. To show this behavior, we are going to create a new component, similar to the list we used in Chapter 9, Improve the Performance of Your Applications, when we learned how to use keys and PureComponent. We can also help the library to optimize its job by using the keys. I personally decided to try all of them, and I created a repository where I publish a small button component built with each one of the available solutions: In the beginning, there were two or three, while today there are more than forty. When the data is received, it can be transformed into new information and passed to the other children down the tree. You can read the full terms of the CLA at the following URL: Suppose, for example, you were building an application with React and you found a bug, what should you do? We will look at both ways of building the best test environment. In fact, templating systems and their Domain-Specific Language (DSL) offer a subset of features, and they try to provide the functionalities of a real programming language without reaching the same level of completeness. Last but not least, there is a + button which we use to submit the value and add it to the array of items. We started by going through the problems of CSS at scale, specifically, the problems that they had at Facebook while dealing with CSS. The component is called Todos, that is a class which extends React.Component: class Todos extends React.Component Inside the constructor, we initialize the state and bind the event handlers: constructor(props) { super(props) this.state = { items: ['foo', 'bar'], value: '', } this.handleChange = this.handleChange.bind(this) this.handleClick = this.handleClick.bind(this) } The state has two attributes: The items: Which is the list of items with a couple of default values; and it's also the array where the new items will be added The value: Which is the current value of the input that the users can fill to add new items [217] Improve the Performance of Your Applications You should be now able to infer the functionalities of the event handlers from their names. To enforce formatting where the attributes are indented by two spaces in relation to the element name, we can just enable the following rule: "react/jsx-indent-props": [2, 2] From now on, if we do not indent the attributes with two spaces, ESLint will fail. It is going to be a minimal and simple set-up on purpose, because the goal here is to show how SSR works rather than providing a comprehensive solution or a boilerplate, even though you could use the example application as a starting point for a real-world application. If, in some other parts of the application, we need to display the same data structure, we do not have to create a new component; we just wrap this one into a new container that, for example, could load the latitude and longitude from a different endpoint. For example, we can change the Spread component in the following way: const Spread = props => And use it in this way: As we have seen many times, with React it's always a good practice to be explicit. For example, it supports pseudo classes using an SASS-like syntax: const Button = styled.button` background-color: #ff0000; width: 320px; padding: 20px; border-radius: 5px; border: none; outline: none; &:hover { color: #fff; } &:active { position: relative; top: 2px; } [ 176 ] Make Your Components Look Beautiful It also supports Media queries: const Button = styled.button` background-color: #ff0000; width: 320px; padding: 20px; border-radius: 5px; border: none; outline: none; &:hover { color: #fff; } &:active { position: relative; top: 2px; } @media (max-width: 480px) { width: 160px; } ` There are many other features that this library brings to your project. We'll build components that are reusable across the application, structure applications, and create forms that actually work. We have to operate on a copy of the array because the unshift method does not return a new array but it mutates the original one, which is something we should avoid when working with the state. In CSS, in fact, it is almost impossible to achieve a proper isolation between files or components. If something goes wrong or we want to debug a particular state of the application we can now open the console, and write: Lumberjack.back() For going back in time and un-applying the state that has been changed, use the following: Lumberjack.forward() For going forward in time and re-applying the state. There are different ways in which we make ESLint help us write better code. The server configuration is slightly different, but it should be very easy for you to follow and understand: const server = { entry: './src/server.js', output: { path: './dist', filename: 'server.js', }, module: { loaders }, target: 'node', externals: [nodeExternals()], } [186] Server-Side Rendering for Fun and Profit As you can see, entry, output, and module are basically the same, except for the file names. Universal applications are better for SEO, and they enable knowledge sharing between the frontend and the backend. You may have written a small/medium application with React and you want to improve your skills and answer all your open questions. This chapter will go through some advanced concepts, such as events, animations, and how to interact with the DOM. If we run npm test again we see that four tests are now passing. For example, if we want to render an image inside the button and a text wrapped into a span, we can do this: Click me! The preceding snippet gets rendered in the browser as follows: Click me! This is a pretty convenient way to allow components to accept any children elements and wrap those elements inside a predefined parent. Then there are three event handlers, which are defined using a class property and an arrow function so that they do not need to be manually bound in the constructor. When an event we are interested in is fired by the browser, React calls the handler on the specific components on its behalf. That is because we are telling React that the component does not need to be updated. Suppose you have a Price component that you use to display the currency and the value. Finally, you saw how easy it is to publish packages on the npm registry and how to choose the right version number to avoid breaking other people's code. Let's now move on to something more exciting and let's try to transform the WindowResize mixin we saw in the previous section into a HoC function that we can reuse across our application. If we use the component in this way: There are no issues. The green tests are now seven in total and there is only one more to go. This is a great feature of React, because we can forget about the browsers we are targeting and write event handlers and functions that are vendor-agnostic. Attributes We must always keep in mind that JSX is not a standard language and that it gets transpiled into JavaScript. [95] Compose All the Things Doing that is bad because it pollutes the state with additional attributes and those attributes may also clash with the attributes used in the components itself. Checking if all the props and all the state attributes are changed is a boring job and it is sometimes hard to maintain complex shouldComponentUpdate implementations, especially when the requirements change frequently. [164] Make Your Components Look Beautiful The first one is entry, which tells Webpack where the main file of our application is: entry: './index.js', The second one is module, which is where we tell Webpack how to load the external dependencies. In the test file we first load all the dependencies: import import import import React from 'react' { shallow, mount } from 'enzyme' withData from './with-data' getJSON from './get-json' We are importing both the shallow and mount functions from Enzyme because the simple tests can be run without the DOM, but since we need to test what happens during the lifecycle hooks we need mount as well. This is far from being a useful feature in a real application but you can easily figure out how, with a small amount of effort, you can create some more complex operations like, for example, showing a new window with the item's data. Autobinding createClass has a cool feature that is pretty convenient but it can also hide the way JavaScript works which is misleading, especially for beginners. This error message is particularly useful because it also gives us information on which component we have to modify to fix the issue. If you did not understand the previous sentence, don't worry; we will talk about linting in the following section. It is also crucial that all the old tests pass to make sure the new code does not introduce regressions in the codebase. The good part is that we are not limited to having elements as children of elements, but we can use JavaScript expressions such as functions or variables. As we gain experience working on the Web, we learn that we cannot master everything and we should find the right way to keep ourselves updated to avoid the fatigue. In this case, we have: { "plugins": ["transform-react-constant-elements"] } The second Babel plugin we can choose to use to improve performance is the React inline elements transform, which replaces all the JSX declarations (or the createElement calls) with a more optimized version of them to make execution faster. React tries to move a step forward by putting the templates where they belong: next to the logic. This makes us more confident with our code so that we can touch any line without worrying about regressions. The second set of dependencies is: npm install --save-dev babel-jest babel-preset-es2015 babel-presetreact As you may know, we now have to create a .babelrc file, which is used by Babel to know the presets and the plugins that we would like to use inside the project. Let's now dive into a real world example and take a look at how we can transform a nonreusable component into a reusable one with a generic and cleaner interface. Control statements Conditionals and loops are very common operations in UI templates and you may feel wrong using the JavaScript ternary or the map function to perform them. It could seem an easy task, but it is not, especially when classes are applied on the fly or concatenated in the client. Using hooks that are executed earlier, such as componentWillMount, will not work either, because the data fetching operation is async while the renderToString is not. In the second section, we started with a simple component, which was able to load data from GitHub, and we made it reusable, thanks to HoC. We learned the basics of prop types and applied those concepts to the reusable components we created. When it comes to setting the default props instead, we used to use a function to return the default properties object, but with classes we have to define a defaultProps attribute on the class and assign the default props to it. If you look at this code in six months' time, you will still find it clear just by reading the name of the function. Once the tools and their functionalities are well understood, we will cover a component from the Redux repository with tests and we'll look at some common testing solutions that can be applied in complex scenarios. The content of the snapshot is the output of the render method of a package called reacttest-renderer, which has to be installed with the following command: npm install --save-dev react-test-renderer Once the test rendered is ready, we can create a new file called TodoTextInput–snapshot.spec.js with the following import statements: import React from 'react' import renderer from 'react-test-renderer' import TodoTextInput from './TodoTextInput' We import React to be able to use JSX, the renderer that creates the tree for the snapshot, and finally the target component that we want to test. We are now ready to write the real tests, with the first one being where we check if the props are passed correctly down to the target: test('passes the props to the component', () => { const ListWithGists = withData()(List) const username = 'gaearon' const wrapper = shallow() expect(wrapper.prop('username')).toBe(username) }) It should be pretty clear, but let's see together what it does. Then we have the button with its event handler. One of the biggest problems with that mixin was in fact that it was using the state of the component to provide the innerWidth value. The solution here is to split the component into smaller ones, each one with a clear responsibility and state. We also have other code bundles from our rich catalog of books and videos available at ing/. Let's now see how we can write a test to check that an onClick handler passed to the component gets called when the button is clicked. This method has been introduced in React 0.14, and it is very powerful because it makes the code easier to maintain and reuse. When we update the state with the new value of the controlled component, in fact, React calls the render again to see if the elements are different. In this section, we will see one of those instances. This means that the function receives the actual template with all the JavaScript expressions, and this makes the library able to use the full power of JavaScript to apply the styles to the elements. In this section, we will see how to apply a similar pattern to our components to make them more clean and maintainable. For example, if you try to add the following code to the previously created list: shouldComponentUpdate() { return false } You can see that clicking on the + button has no effect whatsoever on the application inside the browser, even though the state changes. The returned function is a stateless functional component that receives some props and renders the original component. In fact, using the wrong key can give us unexpected behaviors in some instances. Reader feedback is important for us as it helps us develop titles that you will really get the most out of. Most importantly, learning the reasons why some techniques are considered bad practice helps us understand how React works and how we can use it effectively. For example, if we try to add hundreds of items inside the list we see that the performance are not affected and the input field is not laggy. [124] 6 Write Code for the Browser There are some specific operations we can do when we work with React and the browser. When the JSON is parsed and returned, we can save the raw gists inside the internal state of the component to make them available in the render method: render() { return ({this.state.gists.map(gist => ({gist.description}))}) } The render method is simple–we just loop through the gists and map each one of them into a element that shows their description. There is also the option to write custom validation functions. We will look at how to use it in more detail in Chapter 7, Make Your Components Look Beautiful, but now we will focus on the way it works. Following this approach, called Semantic Versioning (SemVer), is good practice and it makes your users more confident when they have to update your package. The isClicked flag is an example of that, and another one could be the value of an input field before it gets submitted. With this simple rule in mind, we can cover many instances where we might harm the performance if we did not use the keys. Other than that, user interfaces are frequently updated to improve experience, maximize conversions, or simply add new features. Great, we now have a living style guide to document all the states of our components that can be used to share information with designer and product managers. Something like: const statuses = ['open', 'close'] ... The required information will appear under the Errata section. In the real world, running ESLint manually or getting the feedback live in the editor, even if it is very useful, is not enough, because we can miss some warnings or errors, or we can simply ignore them. For example, our WindowResize mixin could expect the component to implement a handleResize function and give developers the freedom of doing some operations when the size changes instead of using the state to trigger the update. We first create a geolocation.js file in our components folder and define the Geolocation component using a class: class Geolocation extends React.Component We then define a constructor, where we initialize the internal state and bind the event handlers: constructor(props) { super(props) this.state = { latitude: null, longitude: null, } this.handleSuccess = this.handleSuccess.bind(this) } Now, we can use the componentDidMount callback to fire the request to the APIs: componentDidMount() { if (navigator.geolocation) { navigator.geolocation.getCurrentPosition(this.handleSuccess) } } When the browser returns the data, we store the result into the state using the following function: handleSuccess({ coords }) { this.setState({ latitude: coords.latitude, longitude: coords.longitude, }) } [86] Compose All the Things Finally, we show the latitude and longitude using the render method: render() { return (Latitude: {this.state.latitude} Longitude: {this.state.longitude}) } It is important to note that, during the first render, latitude and longitude are null because we ask the browser for the coordinates when the component is mounted. Whenever a component's state changes, React calls the render methods on the nodes again and it compares the result with the previous tree of React elements. Smaller components are better for several reasons: for example, they are more maintainable and testable which make it easier to find and fix bugs. With React in particular, the concept of isomorphic web applications became very popular within the JavaScript community. There are a lot of similarities in the way we build UIs with React and the principles of FP, and the more we are aware of it, the better our code will be. From a React point of view, it does not make any difference if we output a div or an SVG element from the render method, and this is what makes it so powerful. The style attribute represents the final style instead, but we do not set the value directly; we use the spring function, so the value is interpolated from the initial state to the final one. Inside the button.spec.js we first import the dependencies: import React from 'react' import TestUtils from 'react-addons-test-utils' import Button from './button' The first one is React, needed to write JSX, the second one is the TestUtils and we will see later how to use it. In May 2015, Mark Dalgleish, one of the creators of CSS Modules, figured out that you could import CSS inside a Webpack bundle as well, and he pushed the concept forward. The way the styles are applied is first by creating a unique class name, adding it to the element, and then injecting the corresponding style in the head of the document. First of all, let's see what a HoC looks like: const HoC = Component => EnhancedComponent HoCs are functions that take a component as input and return an enhanced one as the output. Finally, we'll write tests effectively, and you'll learn how to contribute to React and its ecosystem. The first is the component we will use to wrap the elements we want to animate, while the function is a utility that can interpolate a value from its initial state to the final one: import { Motion, spring } from 'react-motion' Let's look at the code: const Transition = () => ({interpolatingStyle => (Hello React)}) There are a lot of interesting things here. In React, we compose components to create applications where components are a fundamental unit of our structure. The same applies to non-string attributes: Click me! Differences with HTML So far, we have looked at the similarities between JSX and HTML. We expect the first to return false and the second to return true. The fact that we can use JavaScript in our conditions is a big plus, but there are many different ways to express conditions in JSX and it is important to understand the benefits and problems of each one of these in order to write code that is both readable and maintainable. It should be clear to you how to write a simple test to cover this case with Enzyme, so let's check it together: test('submits the form', () => { First of all we define an onSubmit mock function using Jest and we mount the component storing a reference to the wrapper: const onSubmit = jest.fn() const wrapper = shallow() Secondly, we find the first field and we fire the change event on it, passing the value that we want to update: const firstName = wrapper.find('[name="firstName"]') firstName.simulate('change', { target: { name: 'firstName', value: 'Christopher' } }) We then do the same thing with the second field: const lastName = wrapper.find('[name="lastName"]') lastName.simulate('change', { target: { name: 'lastName', value: 'Chedeau' } }) Once the fields are updated, we submit the form simulating an event: const form = wrapper.find('form') form.simulate('submit', { preventDefault: () => {} }) Now, let's write the expectations: expect(onSubmit).toHaveBeenCalledWith('Christopher Chedeau') [264] About Testing and Debugging And close the test block: }) Running npm test in the console shows a green message, which is good; but if you look at the implementation of the test you can easily spot some problems and potential optimization issues. Let's start with the add function we have seen before and transform it into a curried function. The returned function is usually enhanced with some special behaviors. For these and other useful features, we may want to try a package which makes working with CSS Modules even smoother. This rule applies to all the vendor prefixes, except for ms, which is lowercase. After the refs, we will look at how we can easily implement animations with the React addons and third-party libraries such as react-motion. How it works Apart from the differences in declaring the initial state using the factory or extending the Component, the important concept we've learned is that each stateful React component can have an initial state. They are great for the many reasons that we saw in Chapter 3, Create Truly Reusable Components, and for the way they make our applications simpler and easier to reason about but they do not have (yet) any internal implementation that makes them faster. Let's apply storybook to the List example that we created in the previous chapter. If the newTodo prop is true, then the state is set again to an empty string. To use React, it is also important to know the basics of functional programming. } The function that we define as errorHandler receives an error report containing useful information to reproduce and fix the error. That makes the Function as Child solution more flexible. As a JavaScript developer, it should be easy for you to follow this section, even if you have never seen a Node.js application before. The first benefit is that we can wrap components, passing them variables at runtime rather than fixed properties, as we do with HoCs. A good example is a Fetch component that loads some data from an API endpoint and returns it down to the children function: {data => } Secondly, composing components with this approach does not force the children to use some predefined prop names. An example usage is as follows: We can obviously use the full power of React and set some default parameters so that, if the Circle icon is rendered without props, we still show something. The withData function returns a class defined as follows: class extends React.Component With a constructor where the data is initialized: constructor(props) { super(props) this.state = { data: [] } } The data is loaded in the componentDidMount lifecycle hook: componentDidMount() { const endpoint = typeof url === 'function' ? We will see the server-side rendering in detail in Chapter 8, Server-Side Rendering for Fun and Profit, but, for now, you just need to know that firing an async API call when the component gets rendered on the server can give you unexpected results. In this way, you can produce new parts of the user interface very quickly, writing only a few lines of code. When the Buttons get clicked on, they notify the parent, which sends the updated value to the Display component. Snapshots are pictures of the component with some props at a given point in time. Immutability As we have seen, the most powerful tool we can use to improve the performance of our React application is the shouldComponentUpdate using the PureComponent. Should component update For many developers, the way the reconciler algorithm works is confusing. There are different ways of getting an access token, and they are well explained in the GitHub documentation. Click on Code Downloads & Errata. In the next chapter, we will see how to work with React effectively in the browser. We can now make the change at one single point, and all the components that are using it will benefit from the modification. That does not mean that we cannot use event handlers with classes, we just have to bind our functions manually. However, most of the time, whenever we test components, the most important thing for us is that the output is correct and that it does not change unexpectedly. You now know the best practices when making code open source and the benefits and the responsibilities that come with it. He is a big fan of open source and is always trying to learn something new. The last one is the previously created Button component. In general, we aim to remove all the logic from our components, and especially from our render methods, but sometimes we have to show and hide elements according to the state of the application, and very often we have to loop through collections and arrays. But that is not what happens, so we have to find a way to make the gists available on the client side as well. In fact, platforms such as Facebook or Twitter give us a way of defining the content of the snippets that are shown when our pages are shared. For example, developers may encounter bugs and open issues, so you have to go through all of them and try to reproduce the problems. Tests help to avoid regressions because they can tell if the new code breaks the old tests. Regarding perceived performance, there is not much difference in this small example, but if you imagine a big application that shows hundreds of list items, this small change can represent a huge win. Finally, we check if the output has a prop with the same value. Testing components, or trees of components, is a less arduous job because every single part of the application has its responsibilities and boundaries. [76] Create Truly Reusable Components Not only React makes it simpler to create reusable components, but there are also tools that can help us building a visual library starting from the code of the components themselves. As you may notice, instead of repainting everything on the screen, React calculated the minimum amount of operations needed to update the DOM. However, there are some particular scenarios where React cannot optimize the process and it's up to the developer implementing specific solutions to make the application run smoothly. It is important to give the users of your library the same option. The same applies to Media queries, which cannot be defined using inline styles and it makes it harder to create responsive web applications. Nowadays, Google seems to be able to run JavaScript, but there are some limitations, and SEO is often a critical aspect of our businesses. Not only that, depending on too many third-party libraries can create problems if any of them are not maintained or have bugs. [14] Everything You Should Know About React Moreover, there is a massive number of React boilerplates on GitHub, each one with tens of dependencies to offer solutions for any problems. Let's delve into the code: const token = 'access_token=123' const connectWithStar = connect(({ id }) => ({ star: () => ({ starResponse: { url: ` {id}/star?${token}`, method: 'PUT', }, }), })) [121] Proper Data Fetching First, we partially apply the collection function, and we use the id prop to compose the URL. The Page Object pattern Let's now move into another common way of writing tests when the tree of components becomes more complex and there are multiple nested children. We have learned how refactoring and designing your components structure in the proper way could provide a performance boost. For example, if we have to change a label when a button is clicked we should not store the text of the label, but we should only save a Boolean flag that tells us if the button has been clicked or not. [272] Anti-Patterns to Be Avoided In that case, it's best practice to make it explicit and give the prop a name that makes your intentions clear, such as initialCount. Luckily, with React, we can render our components on the server-side and serve the content of our applications to the crawlers in such a way that it is easy for them to understand and index the content. If we log the tree in the console, we can see what it looks like: { type: 'input', props: { className: '', type: 'text', placeholder: undefined, autoFocus: 'true', value: '', onBlur: [Function], onChange: [Function], onKeyDown: [Function] }, children: null } Finally, we write the expectation statement checking if the tree matches the previously saved snapshot: expect(tree).toMatchSnapshot() The first time we run the tests with npm test the snapshot is newly created and stored in a __snapshots__ folder. If we change the render method of the List component as follows: render() { return ({this.state.items.map(item => ( ))} +) } In the preceding code, for each Item, we set the key and the item prop to the current value of the item. However, all the checks and warnings come with a cost which we want to avoid in production. The expectation here is that the mock onSave callback gets called with the value of the event. We will use it to show all the features of the CSS Modules. [33] Clean Up Your Code One goal we should always keep in mind is to never add too much logic inside our components. Using start(), we begin storing the information and with stop() we tell the add-on that we have enough data and we are ready to display it. With Webpack this is pretty straightforward: You just have to set the libraryTarget in the output section of the configuration file. A real-life parallel in the imperative world would be entering a bar for a beer, and giving the following instructions to the bartender: Take a glass from the shelf Put the glass in front of the draft Pull down the handle until the glass is full Pass me the glass In the declarative world, instead, you would just say: "Beer, please." The declarative approach of asking for a beer assumes that the bartender knows how to serve one, and that is an important aspect of the way declarative programming works. ESLint does not make any decisions for you so it is up to you to decide which rules to enable. [207] Improve the Performance of Your Applications For that reason, React gives us a special component from which we can inherit and which implements a shallow comparison of all the props and the state attributes for us. The best solution would be to be able to monitor the performance of components without modifying the code and to do that we can use the chrome-react-perf extension for Chrome. The term isomorphic does not fit in this scenario, because in the case of React the applications are exactly the same, and that is why one of the creators of React Router, Michael Jackson, proposed a more meaningful name for this pattern: Universal. To do this, we just have to enclose expression within curly braces: Hello, {variable}. Webpack lets us define a set of externals, which are dependencies that we do not want to add to the bundle. Here is a screenshot of the component in action: Now, this component works and, if we run it inside the browser, we'll see the list of items with the two default values and the form which we can use to add new items to the list. Creating functions inside the render method Constants props Refactoring and good design Tools and libraries Immutability Monitoring tools Babel plugins Summary Chapter 10: About Testing and Debugging The benefits of testing Painless JavaScript testing with Jest Mocha is a flexible testing framework JavaScript testing utilities for React A real-world testing example React tree Snapshot Testing Code coverage tools Common testing solutions Testing Higher-Order Components The Page Object pattern React Dev Tools Error handling with React Summary [iv] 181 181 182 183 183 184 191 194 198 199 200 205 206 208 208 209 211 215 217 223 223 224 225 227 228 229 230 238 242 245 253 255 257 258 262 266 267 269 Chapter 11: Anti-Patterns to Be Avoided Initializing the state using props Mutating the state Using indexes as a key Spreading props on DOM elements Summary Chapter 12: Next Steps 270 270 273 276 279 281 282 Contributing to React Distributing your code Publishing a npm package Summary 283 285 288 290 Index 291 [v] Preface Taking a complete journey through the most valuable design patterns in React, this book demonstrates how to apply design patterns and best practices in real-life situations, whether it's for new or already existing projects. So, let's look at how we can do it, by following the Container and Presentational pattern. It is worth saying that performing a deep comparison of complex objects can sometimes be more expensive than the render method itself. When the user types something in the field, the handleChange function is called and the new value for the field is stored into the state. The data always flows from parents to children, but children can notify the parent and make the tree re-render with the new information. This is really powerful, because it gives us access to the internal instance of a child component, but it is also dangerous and should be avoided. When we test React components we want to make sure that they work properly and that, given different sets of props, their output is always correct. The paradigm enforced by React is not new, and it was not invented by its creators, but React has contributed to making the concept mainstream and, most importantly, popularized it in such a way that is easier to understand for developers with different levels of expertise. Let's see how to use it. That's unfortunately far from being true. Let's now go deeper into the topic of state and see exactly why it is important to use it and find out how it works. The following cheat sheet from Dan Abramov will help you taking the right decision: Prop types Our goal is to write truly reusable components and to do that we have to define their interface in the clearest possible way. Finally, the event handlers are set to the event properties of the element. It is important to know that when setting the ref callback on a non-native component (a custom component that starts with an uppercase letter), the reference we receive as a parameter of the callback is not a DOM node instance, but the instance of the component itself. That is the reason why the developers at AirBnb decided to create Enzyme, a tool built on top of the TestUtils that makes it easy to manipulate the rendered components. In fact, we can pass arbitrary currencies and values to the component without needing a custom parent to provide the values. As mentioned previously, the string we imported was non-random, but it is generated using the hash of the file and some other parameters in a way that is unique within the code base. In this section, we will look at some common tools and solutions to solve re-rendering issues, figuring out which components can be optimized. There are no rules that say that the Presentational component must not have a state. It'll be fun! The first thing to do is to move into a new folder and run: npm init Once the package.json is created we can start installing the dependencies, with the first one being the jest package itself: npm install --save-dev jest To tell npm that we want to use the jest command to run the tests, we have to add the following scripts to the package.json: "scripts": { "test": "jest" }, In order to write components and tests using ES2015 and JSX, we have to install all Babelrelated packages so that Jest can use them to transpile and understand the code. It also has an event handler, which is fired every time the value of the field changes. Again, we use the partial application to specialize the HoC and reuse it multiple times: const withCurrency = getContext({ currency: React.PropTypes.string }) And then we apply it to the component: const PriceWithCurrency = withCurrency(Price) Now, we can replace the old Price component with the resulting one, and it will still work without being coupled with the context. This also means that, during the first render, the list of gists is empty and we might want to show a spinner by applying one of the techniques that we saw in Chapter 2, Clean Up Your Code, which is beyond the scope of the current section. So, we can change the / route as follows: app.get('/', (req, res) => { fetch(') .then(response => response.json()) .then(gists => { const body = ReactDOM.renderToString() const html = template(body) res.send(html) }) }) Here, we first fetch the gists and then we render the App to a string, passing the property. SVG is great, because it is a declarative way of describing vectors and it fits perfectly with the purposes of React. However, you may think that there is too much boilerplate and that you are required to know too many different tools to run a simple Universal application with React. [170] Make Your Components Look Beautiful For example, extract the rule to set the background to red from the rules of the button into a separate block, as follows: .background-red { background-color: #ff0000; } We can then compose it inside our button in the following way: .button { composes: background-red; width: 320px; padding: 20px; border-radius: 5px; border: none; outline: none; } The result is that all the rules of the button and all the rules of the composes declaration are applied to the element. For a simple UI, we could just use createElement and, only when we start building something more complex, we can include a transpiler to enable JSX and convert it into JavaScript. This is pretty powerful, because we can easily access methods on the components, but we should be careful, because it breaks the encapsulation and makes refactoring pretty hard. This is particularly useful if we want to call a function on each event (for example, analytics) or if some events share the same logic. In FP, a function, instead of changing the value of a variable, creates a new variable with a new value and returns it. [266] About Testing and Debugging Props and state can be read, and they can be changed in real time to trigger updates in the UI and see the results straightaway. I have used the word usually because, as we will see, once we have it clear in our mind what the problems with this approach are, we might still decide to use it. [10] Everything You Should Know About React The following is an example of an element: { type: Title, props: { color: 'red', children: 'Hello, Title!' } } Elements have a type, which is the most important attribute, and some properties. [259] About Testing and Debugging Creating a dummy component is pretty common practice when testing HoC because we need a target component to enhance, so we can figure out if all the features are correctly working. But if we do something like: [280] Anti-Patterns to Be Avoided Then React complains because we are applying a non-standard attribute to the DOM element. We do not want to add too many responsibilities to the List component as its role is to display the list; so, we change it to use a subcomponent for each row. If the values of the requests are strings, then the data is fetched as soon as the props are available. The spread attributes is an example of that, and another common example is that JavaScript expressions can be used as attributes values by enclosing them within curly braces: [28] Clean Up Your Code Common patterns Now that we know how JSX works and can master it, we are ready to see how to use it in the right way following some useful conventions and techniques. The presentation is, instead, the part inside the render method where we create the elements to be displayed on the UI. To do so, we can easily add the following plugin to the list of plugins in the Webpack configuration: new webpack.optimize.UglifyJsPlugin() If we run the production version of React and we realize that some parts of our applications are still slow, there are different techniques that we can put in place to improve the rendering of our components. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews. So for example this snippet: () => this.setState() Gets transpiled into the following code with Babel: var _this = this; (function () { return _this.setState(); }); As you can imagine, one possible solution to the autobinding problem is using the arrow function, let's see an example: class Button extends React.Component { handleClick() { console.log(this) } render() { return this.handleClick()} /> } } [56] Create Truly Reusable Components This would work as expected without any particular problems. [143] Write Code for the Browser For a UI library such as React, it is crucial to provide an easy way for developers to create and manage animations. Suppose, for example, that we want to optimize the way we request CSS, loading the CSS related to a particular page only when the users navigate to it. In fact, we are storing the items as well as the value of the form field which are two entirely different things. The SSR should be enabled on when strictly needed. You may have chosen one method according to the examples on the React website or by following the style of the boilerplate you used to scaffold the project. To avoid those linting errors, there is another plugin to install: eslintplugin-jsx-control-statements. Rendering a component into a real DOM would require a browser but Jest comes with a special DOM in which we can render our components using the console. [46] Clean Up Your Code Immutability We have seen how to write pure functions that don't mutate the state, but what if we need to change the value of a variable? A style guide is a visual collection of every single component of the app that can be used across different pages. We will look at how easy it is to implement now. On the other hand, applying this pattern without a real reason can give us the opposite problem and make the codebase less useful as it involves the creation of more files and components. React enforces a very interesting pattern to make data go from the root to the leaves. After that, we'd test if the logic of creating the endpoint from the URL works for both function and static strings. It is easy to think that they are rendered quicker because they do not have an instance, and do not have a state or event handlers, but at the moment that's not the case. The following list represents more or less all the variations and functionalities of the component that are worth testing: The state is initialized with the value coming from the props The placeholder prop is correctly used in the element The right class names are applied following the conditional logic The state is updated whenever the value of the field changes The onSave callback is fired according to the different states and conditions It is now time to start writing code and create a file called TodoTextInput.spec.js with the following import statements: import React from 'react' import { shallow } from 'enzyme' import TodoTextInput from './TodoTextInput' We import React, the Shallow rendering function from Enzyme, and the component to test. Applying TDD means writing the tests first and then writing the code to pass the tests. We first import the main function from the library: import { storiesOf } from '@kadira/storybook' Then we use it to define our stories, as follows: storiesOf('List', module) .add('without text field', () => ( )) [77] Create Truly Reusable Components Using storiesOf, we can define the name of the component and add the stories, and each one includes a description and a function that must return the component to be rendered. Once more, let's start with a simple stateless functional component so that we can add features to it iteratively: const JSONSchemaForm = () => ( ) Now, if we render this component inside the page, we will see a form with the fields we declared in the schema and a submit button. 5. We've solved a performance issue just by refactoring the component and changing the design a bit by separating the responsibilities correctly. However, in a real-world application, we will likely want to load some data instead of a static React component such as the App in the example. In this case, with the transitionAppear prop, we are telling the component that we want to animate the children when they appear on the screen. Babel In order to use JSX (and some features of ES2015) in our code, we have to install Babel. In this chapter, we will cover the following: Common problems with regular CSS at scale What it means to use inline styles in React and the downsides How the Radium library can help fix issues of inline styles How to set up a project from scratch using Webpack and CSS Modules Features of CSS Modules and why they represent a great solution to avoid global CSS Styled Components, a new library that offers a modern approach to styling React components Make Your Components Look Beautiful CSS in JS In the community, everyone agrees that a revolution took place in the styling of React components in November 2014, when Christopher Chedeau gave a talk at the NationJS conference. To create an unordered list to show the users, you can do the following: {users.map(user =>{user.name})} This snippet is incredibly simple and incredibly powerful at the same time, where the power of the HTML and JavaScript converge. So, we tried to solve the problems of Atomic CSS using CSS Modules and we called the technique Atomic CSS Modules. These tools are not able to tell us if our code is going to do what it supposed to do, but they can help us to avoid syntactical errors. To do that, we update the "extends" key in our .eslintrc file as follows: "extends": ["eslint:recommended", "plugin:react/recommended"], Now if we write something wrong, for example, we try to use the same prop twice in a React component, we are going to get an error: The preceding code returns the following: No duplicate props allowed (react/jsx-no-duplicate-props) There are a lot of rules available to be used in our project. This is a common feeling, called JavaScript Fatigue, as described in the introduction to this book. If you liked this chapter, go and give them a star on GitHub: [197] Server-Side Rendering for Fun and Profit Summary The journey through server-side rendering has come to an end. Suppose you need to attach to every component the same className property for some reason. Select the book for which you're looking to download the code files. We first enhance the dummy List that we are passing to the HoC, we then define a prop that we pass to the component and we shallow-render it. React Dev Tools When testing in the console is not enough, and we want to inspect our application while it is running inside the browser, we can use the React Developer Tools. The app.js should have the following content: import React from 'react' const App = () => Hello React export default App Nothing complex here: we import React, we create an App component, which renders the Hello React message, and we export it. Now we can close the first test block: }) The second test block where we test that the onClick callback is fired looks like this: it('fires the onClick callback', () => { We then create the spy, in a similar way before: const onClick = spy() We render the button into a detached DOM using the TestUtils: const tree = TestUtils.renderIntoDocument( ) And we use the tree to find the component by tag: const button = TestUtils.findRenderedDOMComponentWithTag(tree, 'button') The next step is to simulate the button click: TestUtils.Simulate.click(button) [241] About Testing and Debugging Finally, we write the expectation: expect(onClick).to.be.called() Again, the syntax is slightly different but the concept does not change: we check for the spy to be called. [43] Clean Up Your Code The question is, when do we consider a line too long? Testing this kind of component is not as intuitive as testing simple ones, so it is worth looking at some common solutions together. [35] Clean Up Your Code First of all, we have to install it: npm install --save jsx-control-statements Once it is installed, we have to add it to the list of our babel plugins in our .babelrc file: "plugins": ["jsx-control-statements"] From now on we can use the syntax provided by the plugin and Babel will transpile it together with the common JSX syntax. We create a Buttons component to display the increment and decrement buttons, which, instead of firing internal functions when they are clicked on, use the functions received from the props: const Buttons = ({ onDecrement, onIncrement }) => (- +) Buttons.propTypes = { onDecrement: React.PropTypes.func, onIncrement: React.PropTypes.func, } It is a simple, stateless functional component where the onClick event handlers fire the functions received from the props. Why subscribe? Douglas Crockford made linting popular in JavaScript with JSLint (initially released in 2002) a few years ago; then we had JSHint and finally, the de-facto standard in the React world nowadays is ESLint. [288] Next Steps The number in the middle indicates the minor version of the release and it should be changed when new features are added to the library and those new features do not break existing APIs. Finally, the first number on the left represent the major version and it has to be increased when a version containing breaking changes is released to the public. We also learned how React uses the elements returned by the components to display the UI on the screen. It should be clear how this pattern works, so let's look at the advantages of this approach. A popular convention is to name the event handler functions after the event name and prefix them using handle (for example, handleKeyDown). Our target component is going to be the withData HoC created in Chapter 5, Proper Data Fetching. At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks. However, with classes we define our initial state using the state attribute of the instance and setting it inside the constructor method of the class: class Button extends React.Component { constructor(props) { super(props) this.state = { text: 'Click me!', } [54] Create Truly Reusable Components } render() { return {this.state.text} } } These two ways of defining the state are equivalent but, again, with classes we just define properties on the instance without using any React-specific APIs, which is good. The following chapter will be about CSS and inline styles, and it will clarify what it means to write CSS in JavaScript. [290] Index A animations about 143 creating 144, 145 React motion 145 Atomic CSS Modules 172, 173 B Babel about 51 installing 21, 22 plugins 225 big picture reference 284 Browserify 163 C callbacks 109 camelCased 26 children 11, 83, 84, 85 classes, creating about 51 createClass factory, using 51 main differences 52 React.Component, extending 52 stateless functional components 57 code coverage tools 255, 257 code distributing 285, 286, 287, 288 codebase 90 common misconceptions 14, 16 common patterns, JSX conditionals 30, 31, 33, 34 control statements 35, 36 loops 35 multi-line 29, 30 sub-rendering 37, 38 components about 11, 13 communication between 82 composition 170 Container and Presentational pattern 85, 86, 87, 88, 90 Contributor License Agreement (CLA) about 283 reference 283 controlled components about 134 JSON schema 134 Controlled form component 262 core-notes reference 26 CSS Modules about 162

Atomic CSS Modules 172 locally scoped CSS 166, 170, 171 project, setting up 163, 164, 165 React CSS Modules 173, 174 Webpack 162 CSS animations 156 in JS 151 reference 163 currying process 47 D data flow about 107, 108 child-parent communication (callbacks) 109 common parent 110, 111 data fetching 111, 112, 114, 115, 117 fetching, example 191, 192, 193 dead code elimination 152 declarative programming 8, 9 differences, classes autobinding 55, 57 props, defining 52, 53, 54 state 54 Document Object Model 200 DOM elements 280, 281 DOM node 11 domProps 161 E Enzyme 228 error handling React 12 [illegible] ESLint about 38, 205 Airbnb configuration 44, 45 configuration 39, 40, 41 installation 39 React plugin 42, 43, 44 event bubbling 139 event delegation 139 event switch 138 events 136, 137, 138 Express 180, 184 F fetch API 193, 194 fetch polyfill 112 forms about 126 controlled component 131, 132, 133 JSON schema 135, 136 uncontrolled components 126, 127, 128, 129, 130 Function as Child 103, 104 Functional CSS 172 functional programming (FP) about 45 composition 48 currying 47, 48, 49 immutability 47 purity 46 user interfaces 48, 49 G gists reference 117 GitHub reference 177 global keyword 170 good first bug reference 270 H Higher-order Components (HoCs) 94, 95, 96 Higher-order Functions (HoFs) 45, 46 [illegible]

[illegible] ... Whenever you change something in your library and you want to push a new version, you just have to run: npm version <type> Where type is one patch, minor or major. So, every time we need the children to push data into the parent, we can pass callbacks [illegible]

[illegible] linter which rules we are following in our source files. The first thing we have to do is to modify the render method as follows: render() { return ({this.state.items.map(item => ([209] Improve the Performance of Your Applications))} +) } Instead of rendering the single list items inside the map function, we return a custom Item component to which we pass the current item and we use the key to tell React which components existed before the update. 3. Let's see if we can apply the same concept to React components and achieve our goal to sharing functionalities between components while avoiding the downsides of mixins. As we have seen in Chapter 2, Clean Up Your Code, the presets are sets of configuration options that instruct Babel on how to deal with the different types of syntax (for example, JSX). Spinning up a browser for every single test makes tests slower and less predictable, degrading the developer experience; running the tests using the console instead is faster. Next.js is based on conventions, with the most important one being that you can create pages to match the browser URLs. The default page is index, so we can create a folder called pages and put an index.js file inside it. What we have not seen yet is how React can render the same components on the server, giving us a powerful feature called Server-Side Rendering (SSR). Their signature is something similar to the following: const HoC = args => Component => EnhancedComponent What we can do is create a function using the first call and wrap our component into it: const withFlattenUser = flattenProp('user') const ProfileWithFlattenUser = withFlattenUser(Profile) Great! Now suppose for some reason we want to change the attribute username to make this component more generic and reusable. There is also a special property, called children, which is optional and represents the direct descendant of the element. [173] Make Your Components Look Beautiful Also, if we are referencing a class name that does not exist in the CSS file, there is no way to know it, and undefined is added to the list of classes. You may also notice that, if you edit the application inside your editor, you will be able to see the results within the browser instantly without needing to refresh the page. We also set the module loaders using the previous object we created with babel-loader. We can reuse the Uncontrolled class we just created and change the constructor as follows: constructor(props) { super(props) this.state = { firstName: '', lastName: '', } this.handleChangeFirstName = this.handleChangeFirstName.bind(this) this.handleChangeLastName = this.handleChangeLastName.bind(this) this.handleSubmit = this.handleSubmit.bind(this) } We initialize the two fields inside the state and we define an event handler for each one of the fields as well. Let's start with a context-aware component and let's transform it step by step into a reusable one, thanks to HoCs: const Price = ({ value }, { currency }) => ({currency}{value}) Price.propTypes = { value: React.PropTypes.number, } Price.contextTypes = { currency: React.PropTypes.string, [101] Compose All the Things } We have a stateless functional component that receives the value as a property and the currency as the second parameter from the context. However, that is not an easy goal, especially when you are creating an application iteratively and in the first iteration you are not sure exactly how to split the components into smaller ones. Unfortunately, the props are unknown at the moment we apply the HoC, so we need a hook that is called when the props are available and before making the API call. Simply, all the composed class names are applied one after the other on the component in the DOM. You should know that React is maintained by developers at Facebook and hundreds of contributors within the JavaScript community. This approach has many benefits because it simplifies the components' behavior and the relationship between components, making the code more predictable and maintainable. Luckily, React (and the component era), makes testing user interfaces easy and efficient. } The output will be the following: .button { ... JSON schema Now that we know how the forms work in React, we can move into something that helps us automate the form creation, avoid writing a lot of boilerplate, and keep our code much cleaner. The report, in fact, apart from the original error object, gives us the component name and the function name that generated the issue. The latter is terser and easier to read, which makes a huge difference in big code bases, where maintainability is crucial. Let's see how it works: const foo = { id: 'bar' } return The preceding code gets transpiled into the following: var foo = { id: 'bar' }; return React.createElement('div', foo); JavaScript templating Finally, we started with the assumption that one of the advantages of moving the templates inside our components instead of using an external template library is that we can use the full power of JavaScript, so let's start looking at what that means. Next.js You have looked at the basics of server-side rendering with React and you can use the project we created as a starting point for a real app. Testing a single component is great, but it does not guarantee that multiple individually tested components will still work once they are put together. Last but not least, we must install jsdom, which is the package that we use to create the detached DOM and let the TestUtils do their job in the console without a real browser: npm install --save-dev chai chai-spies jsdom [239] About Testing and Debugging We are now ready to write the tests and we'll use the same button.js created earlier. Last but not least, we use the style attribute of the field to set its font-size. The direct consequence is that many developers try to learn React and Redux together, so they never find out how to use the React state correctly. As you can see, that component is not truly reusable because it needs a parent with the currency as child context types to work. React is so powerful because it lets you build complex applications composing small, testable, and maintainable components. : } You can find the ternary condition used in popular repositories such as the Redux realworld example (orld/src/components/List.js#L25), where the ternary is used to show a loading label if the component is fetching the data, or load more inside a button depending on the value of the isFetching variable: [31] Clean Up Your Code {isFetching ? We could define a component in the following way, extending React.Component: class Button extends React.Component { handleClick() { [55] Create Truly Reusable Components console.log(this) } render() { return } } The result would be a null output in the console when the button is clicked. As mentioned at the beginning of the chapter, CSS is global, and that makes it very hard to maintain in large applications. The new parameters are the target, where we specify node to tell Webpack to ignore all the built-in system packages of Node.js, such as fs and externals, where we use the library we imported earlier to tell Webpack to ignore the dependencies. Our goal has always been to write the JavaScript and the HTML in different files. Poor design choices often lead to issues and, in the case of React, if we do not put the state in the right place, the risk is that our components are going to render more than needed. The differences are that the heading, in this case, is the username rather than title and the optional field, that has to be shown only if present, is the bio of the user. Finally, we looked at how living style guides can help us to communicate better with other members of our team, to avoid creating duplicated components and to enforce consistency within the application. You should never need to access the original native event, but it is good to know you can do it if you need to. One way to achieve the same goal without adding complexity is using HoCs, which are functions that take a component and return an enhanced one. We used to do that because server-side rendered applications could not give us the level of interactivity users expect, while a client-side application did not get indexed by search engines. Good job, the component is now covered with tests; and if we need to refactor it, change its behavior, or add some features, the tests will help us to discover whether the new code breaks any of the old functionalities. But if our render methods do complex operations, React will take some time to figure out that no operations have to be done, which is not optimal. Moreover, explicitly importing the CSS dependencies inside our components helps us see clearly which components need which CSS. If our server returns an empty HTML structure for all the URLs, the result is that, when the pages are shared on the social networks, the snippets of our web application are empty as well, which affects their virality. [51] Create Truly Reusable Components Extending React.Component The second way to define a React component is by using the ES2015 classes. Webpack is especially popular in the React world because it offers many interesting and useful features, with the first one being the concept of loaders. Those separate apps usually had to be maintained by two teams of developers with different skill sets. Let's first install it and then we will see what functionalities it provides: npm i --save-dev react-addons-test-utils Now we have everything we need to test our components. Wouldn't it be nice to separate it from the part where the position gets requested and loaded in order to iterate faster on it? Due to the way the library works and its declarative nature, dealing with input fields and other form elements is non-trivial with React, but as soon as we understand its logic, it becomes clear. The reason behind this behavior is that all the objects return a new instance when created and a new array is never equal to another, even if they contain the same values: [] === [] false There is also a table printed on the console, which shows the time that React is wasting rendering the items with unchanged props when there is no need to touch the DOM. The button, for example, does not operate properly without text and the solution is to mark the prop as required: Button.propTypes = { text: React.PropTypes.string.isRequired, } If a developer uses the button inside another component without setting the text property, they receive the following warning in the browser console: Failed prop type: Required prop `text` was not specified in `Button`. The first is fired before the component gets rendered for the first time and the second is fired right after the component is mounted. First published: January 2017 Production reference: 1100117 Published by Packt Publishing Ltd. It is worth noting that we load the bundle on the client side even if the app is server-side rendered. It also provides all the props that the component received. In React, when we need to push information or simply trigger an event from children to the parent, we use callbacks. The result is a simple app that lets users add to-do items and mark them as done. In that way, we are using the state properly, and we can always recalculate different values according to it. The very first test that we can write to be sure that everything is working is the following one (I always start in this way): test('works', () => { expect(true).toBe(true) }) The test function accepts two parameters, with the first one being the description of the test and the second one a function that contains the actual test implementation. Each of the children can hold a local state and use it as a prop for its nested components. In this chapter, we will cover the following: Understanding what a Universal application is Figuring out the reasons why we may want to enable server-side rendering Creating a simple static server-side rendered application with React Adding data fetching to server-side rendering and understanding concepts such as dehydration/hydration Using Next.js by Zeith to easily create a React application that runs on both the server and the client Server-Side Rendering for Fun and Profit Universal applications When we talk about JavaScript web applications, we usually think of client-side code that lives in the browser. Let's define the handleSubmit function where we just log the value. The function type returns the type of the rendered elements, and the function text returns the text being rendered inside the component. [260] About Testing and Debugging Let's now move to the tests that require mounting the component to the detached DOM. It works very well with immutable data structures, which we will see in more detail later in this chapter. Props JSX is very convenient when your DOM elements or React components have props. It is important to say that the React team has a plan to follow and the main features are designed and decided by the core members. Finally, we render the component and check that the value prop of the output element is equal to the given text. It's already been implemented so we will not follow the TDD process this time, but the goal here is to show the main differences between the two frameworks. It is now time to talk about data fetching and one-way data flow. Then, we moved to Radium, which solves the main problems of inline styles, giving us a clear interface to write our CSS in JavaScript. The behavior is the same as we would have without using the key prop at all. However, in some cases, we prefer to be more strict and fix some values to keep consistency. In fact, creating simple and clean components that accept clear props and that are decoupled from the data is the best way to share a library of base components with the rest of the team. Mocha is a flexible testing framework In this section, we will see how it's possible to achieve the same results with Mocha in order to make it clear that, with React, you can use any testing framework. If we run console.log (styles), we can see the following object in DevTools: { button: "_2wpxM3yizfwbWee6k0UlD4" } So, we have an object where the attributes are the class names and the values are (apparently) random strings. The second very interesting implementation detail is again about performance, and it regards the way React attaches the event handlers to the DOM. The next step is telling chai to use the spies package: chai.use(spies) And then we extract the functionalities that we need from the chai package. That is great: using the keys, we helped React to determine the minimum set of operations and improved the rendering performance of our components. We could obviously write all the features we need, but there is an existing library that has a lot of useful functionalities, and it is ready to be used. React Storybook isolates a component so that you can render single components without running the entire app, which is perfect for both development and testing. The only differences are that the hyphenated CSS rules must be camelCased to be JavaScript-compliant, and the values are strings, so they have to be wrapped into quote marks. Downloading the example code You can download the example code files for this book from your account at acktpub.com. The big difference is that we are writing the markup inside a .js file, but it is important to note that JSX is only syntactic sugar and it gets transpiled into JavaScript before being executed in the browser. Also, if you use Eslint, as we saw in the Chapter 2, Clean Up Your Code, and you enabled the jsx-key rule of the eslint-plugin-react, the linter will complain if the keys are missing. The first step to creating a reusable List component is to abstract it a little and decouple it from the data it has to display and we do that by defining a generic collection property. You could go and change all the render methods adding the className prop to each of them, or you could write a HoC such as the following one: const withClassName = Component => props => ( ) The preceding code can be a little difficult to understand initially; let's try to understand it. To use the plugin, we first have to install it: npm install --global eslint-plugin-react Once it is installed, we instruct ESLint to use it by adding the following line to the configuration file: "plugins": ["react"] As you can see, it is pretty straightforward, and it does not require any complex configuration or set up. There is also a rejected property that we can use in case of errors. In this chapter, we will cover the following topics: What JSX is and why we should use it What Babel is and how we can use it to write modern JavaScript code The main features of JSX and the differences between HTML and JSX Best practices to write JSX in an elegant and maintainable way How linting, and ESLint in particular, can make our JavaScript code consistent across applications and teams The basics of functional programming and why following a functional paradigm will make us write better React components Clean Up Your Code JSX In the previous chapter, we saw how React changes the concept of separation of concerns, moving the boundaries inside components. In any case, Facebook developers care a lot about the DX (developer experience), and they listen carefully to the community. [111] Proper Data Fetching At the time of writing, it is natively implemented in Chrome and FireFox and if you need to support different browsers, you must use the fetch polyfill by GitHub: We are also going to use the public GitHub APIs to load some data and the endpoint we will use is the one that returns a list of gists, given a username: username/gists Gists are snippets of code that can be easily shared between developers. [236] About Testing and Debugging We first create a new test block: test('fires the onClick callback', () => { With the title saying that we are going to check for the onClick callback to work properly. Sub-rendering It is worth stressing that we always want to keep our components very small and our render methods very clean and simple. A function is pure when there are no sideeffects, which means that the function does not change anything that is not local to the functions itself. The first one is the package name, which defaults to the folder name, and then the version number. From unit to end-to-end tests, the fact that the interfaces depend on browsers, user interactions, and many other variables makes it difficult to implement an effective testing strategy. Let's start with a very simple snippet: const Button = React.createClass({ render() { return }, }) With the code above we created a button, and we can reference it inside other components in our application. We should at least try to keep the renderIf method as clean as possible, and to do that, we can use another utility library called react-only-if, which lets us write our components as if the condition is always true by setting the conditional function using a Higher-Order Component. ESLint can be configured using a .eslintrc file that lives in the root folder of the project. First of all, if you move into an empty directory and write in your terminal: npm init A new package.json is created and some questions will be displayed. If you are interested in knowing what are the next steps the library will take, you can find some of them under the label big picture on GitHub: ure%22 That said, if you have some good ideas about features that should be added to the library, the first thing to do is to open an issue and start talking with the React team. However, comparing two trees of elements is not free either and React makes two assumptions to reduce its complexity: If two elements have a different type, they render a different tree Developers can use keys to mark children as stable across different render calls The second point is interesting from a developer perspective because it gives us a tool to help React render our views faster. For example, if you are creating a TO-DO list, you could have a story to represent a checked item and another story to describe an unchecked one. For example, let's create a .eslintrc and disable the semicolon: { "rules": { "semi": [2, "never"] } } This configuration file needs a bit of explanation: "semi" is the name of the rule and [2, "never"] is the value. The method is called shouldComponentUpdate and if it returns false, the component and all its children's render methods are not called during an update of its parents. In this chapter, we will go through some of the most popular composition patterns and tools. [243] About Testing and Debugging Your test should now look like this: test('renders with text', () => { const text = 'text' const button = shallow() expect(button.type()).toBe('button') expect(button.text()).toBe(text) }) This is way more concise and clean than before. Props The first difference is in how we can define the props that a component expects to receive and the default values for each one of the props. We will start with a basic button without styling, and we will add some basic styling, as well as pseudo classes and Media queries, to it in order to learn the main features of the library. With getInitialState, the state gets initialized with the initial innerWidth of window: getInitialState() { return { innerWidth: window.innerWidth, } }, [91] Compose All the Things Now we want to keep track of the changes, so when the component mounts, we start listening to the window resize event: componentDidMount() { window.addEventListener('resize', this.handleResize) }, We also want to remove the event listener as soon as the component unmounts, which is critical freeing the memory and not leaving unused listeners attached to the window: componentWillUnmount() { window.removeEventListener('resize', this.handleResize) }, Finally, we define the callback to be called every time the window resize event is fired. render() { return ({this.state.items.map(item => ( ))} +) } If we try the component again in the browser, we see that the console now contains no message, which means that the items do not re-render themselves unnecessarily when the new element is added. All this information should be enough to write a test, reproduce the issue, and fix it quickly. We then have to set the name for each field, and we are going to do it in the render method: render() { return (Submit) } That's it; we can now add as many fields as we want without creating additional handlers. [122] Proper Data Fetching This is the final result in the browser window: You may have noted that, thanks to react-refetch, we can keep our components stateless and unaware of the actions that they are firing. In that case, it makes sense to remain on the same line and avoid adding noise to the markup, as follows: {message} Close Always remember to wrap your elements inside parentheses when you write them in multiple lines. Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. Let's start importing the library we will use to set our node externals. [281] 12 Next Steps React is one of the most amazing libraries that have been released in the last years, not only because of the library itself and its great features but, most importantly, due to the ecosystem that has been built around it. Since styles are global, any style from any file can be applied to any element, and losing control is very easy. Examples are also useful for you to test new versions of the component and see if there are unexpected breaking changes. The npm test now tells us that four tests have passed. DOM elements and React components With JSX, we can create both HTML elements and React components; the only difference is whether or not they start with a capital letter. Similarly, thanks to tests, it becomes easier to refactor the code because tests guarantee that the functionalities do not change during the rewrite. First-class objects In JavaScript, functions are first-class objects, which means that they can be assigned to variables and passed as parameters to other functions. He currently works mostly in React, but has proficiency with other frameworks and libraries, such as Angular, Backbone, and jQuery. Developers tend to focus on the feature they are currently implementing and sometimes it is hard to know if other parts of the application are affected by those changes. The code is verbose and, most importantly, it's coupled with the structure of the markup. Depending on the severity of it, they might fix it or ask you to fix it. Suppose, in fact, that you need to rename the reset function for some reason; you have to check all the parent components that are using it and change them as well. If we look into it we don't find the React element object, but a human-readable version of the rendered output: exports[`test snapshots are awesome 1`] = ` `; [254] About Testing and Debugging If we now go back to the test, add the editing prop to the component, and run npm test again, we get the following response in the console: FAIL ./TodoTextInput-snapshot.spec.js snapshots are awesome expect(value).toMatchSnapshot() Received value does not match stored snapshot 1. When the promise is resolved, we are able to return the props object with the gists. Calling this.setState() or any other functions would work as expected. The shouldComponentUpdate method passes two parameters which we can use to implement those checks: the first parameter represents the nextProps while the second parameter represents the nextState. When we install and run it for the first time, we do not see any errors because it is completely configurable and it does not come with any default rules. DOM elements and components can be nested with each other, to represent the render tree: { type: Title, props: { color: 'red', children: { type: 'h1', props: { children: 'Hello, H1!' } } } } When the type of the element is a function, React calls it, passing the props to get back the underlying elements. The solution is to write our scripts in JSX and ES2015, and when we are ready to ship, we compile the sources into ES5, the standard specification implemented in major browsers today. Now we can pass images, labels, and even other React components inside the Button and they will be rendered as its children. When the fix is ready and all the tests are green, you can submit a Pull Request and wait for the core team members to review it. Whenever you call createClass, extend Component, or simply declare a stateless function, you are creating a component. However, it is crucial to know how the internals work before adding too many layers, and that is why we started with the manual process before learning Next.js. In this case, we'll define our stories in list.js. The code is pretty straightforward: const Uncontrolled = () => (Submit) If we run the preceding snippet in the browser, we can see exactly what we expect: an input field in which we can write something, and a clickable button. [94] Compose All the Things We declare a withClassName function that takes a Component and returns another function. The workaround here is to define animations globally and use them inside the style attribute of the elements. First of all, we have to install the library: npm install --save @kadira/react-storybook-addon Once the package is installed, we can start creating a story. [112] Proper Data Fetching We will then use the componentDidMount hook so that we can be sure that the API endpoint is called on the browser only. [26] Clean Up Your Code Consider the following snippet: foo bar baz In the browser, which interprets HTML, this code would give you foo bar baz, which is exactly what we expect. The ReactCSSTransitionGroup applies a class with the name of that property to the child element so that we can then use CSS transitions to create our animations. Let's see what it looks like: const FunctionAsChild = ({ children }) => children() FunctionAsChild.propTypes = { children: React.PropTypes.func.isRequired, } As you can see, FunctionAsChild is a component that has a children property defined as a function and, instead of being used as a JSX expression, it gets called. In the following section, we will look at what it means to use inline styles with React and the benefits and downsides of it. Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. If the CSS related to this last page has some rules that also apply to the elements of different pages, the fact that it has been loaded last could affect the styling of the rest of the app. As shown in the following screenshot, the items shift down while the input elements remain in the same position in such a way that their value does not match the value of the items anymore: [278] Anti-Patterns to Be Avoided To investigate and figure out the cause of the problem, we can install the Perf Add-on and import it inside the component: import Perf from 'react-addons-perf' We can then use two lifecycle hooks to store the information about the way React manipulates the DOM and print them in the console: componentWillUpdate() { Perf.start() } componentDidUpdate() { Perf.stop() Perf.printOperations() } Running the component, clicking +, and checking the console should give us all the answers. We can use the object to set the class name attribute of our button, as follows: const Button = () => (Click me!) If we go back to the browser, we can now see that the styles we defined in the index.css have been applied to the button. In fact, as we will see in the coming sections, rendering components it is not the only task that needs to be done to create server-side rendered applications. Then the time will come to split our application into separate modules and organize our files in the right way. Different paths have been followed for writing CSS in JavaScript, and many solutions have been tried, so now the time is ripe for a library that takes all the learning and then builds something on top of it. You can download the code files by following these steps: 1. We fire console.log when the onClick event is fired inside the Item and we now have a new prop which represents the possible statuses of the item. First of all, we have to create a function that receives a component: const withInnerWidth = Component => (class extends React.Component { ... But let's do an experiment. A declarative solution would be as follows: const toLowerCase = input => input.map(value => value.toLowerCase()) The items of the input array are passed to a map function, which returns a new array containing the lowercase values. Finally, we render the form: render() { return (Submit) } If we render the component inside the browser and type the word React into the form field, we will see something like the following inside the console: R Re Rea Reac React The handleChange listener is fired every time the value of the input changes. [145] Write Code for the Browser To use it, we first have to install it: npm install --save react-motion Once the installation is successfully completed, we import the motion component and the spring function. A PromiseState object is a synchronous representation of a Promise and it has some useful attributes such as pending or fulfilled, which we can use to show a spinner or a list of data. Here are some examples of these styles and an explanation of their meaning. For example, it could keep a UI state inside it. The first thing we must do is to inject the gists in the template after we fetched them on the client. A popular solution is the react-jsonschema-form, maintained by mozilla-services, and the first thing we must do is install it using npm: npm install --save react-jsonschema-form Once the library is installed, we import it inside our component: import Form from 'react-jsonschema-form' And we define a schema as follows: const schema = { type: 'object', properties: { firstName: { type: 'string', default: 'Dan' }, lastName: { type: 'string', default: 'Abramov' }, }, } We will not go into the details of the JSON Schema format in this book: the important bit here is that we can describe the fields of our forms using a configuration object instead of creating multiple HTML elements. What we want to do here is pass a function to the button using the onClick property and verify that the function gets called when the user clicks it. There are different techniques that we can use to write tests, and one of the most popular ones is Test Driven Development (TDD). To do so, we change the Todos component to store only the list of items, which is the part of the store that is shared between the list and the form. [194] Server-Side Rendering for Fun and Profit Luckily, Facebook developers and other companies in the React community are working very hard to improve the DX and make the life of developers easier. Make sure to discuss your approach and the path you want to take with them before starting coding so that you do not have to rewrite the code multiple times. To see Jest in action we are going to create a project from scratch, installing all the dependencies and writing a component with some tests. The first method that we are going to implement is componentWillUpdate, which is fired right before the component is updated and re-rendered: componentWillUpdate() { Perf.start() } Inside this lifecycle hook we start monitoring the perf, using the Perf add-on's start() function. For years we used to have completely different applications for the server and client: for example, a Django application to render the views on the server, and some JavaScript frameworks, such as Backbone or jQuery, on the client. Your base generic and reusable components represent your palette of ready-touse components that you can share with other developers and designers on your team. It also reduces the boilerplate a lot. The component we are targeting is the text input used to add and edit items. The way it works is that when the value changes, we store the new value of the field inside the state. This section is our opportunity to make it clear how we can use the state in the right way and understand why, in some cases, we do not need any external libraries. We are going to create a very simple button, similar to the one we built in the example earlier in this chapter. In those cases, we can tell React to avoid reconciling some parts of the tree. [92] Compose All the Things We can use the mixin in many components at a time and also use multiple mixins for each component. [198] 9 Improve the Performance of Your Applications The effective performance of a web application is critical to providing a good user experience and improving conversions. These are the reasons we should prefer SVG for our web applications. For example, if we go back to the first button we created: const Button = ({ text }) => {text} Button.propTypes = { text: React.PropTypes.string, } And then run: react-docgen button.js We get the following object in return: { "description": "", "methods": [], "props": { "text": { "type": { "name": "string" }, "required": false, "description": "" [70] Create Truly Reusable Components } } } Which is a JSON object that represents the interface of our components. We then define an object of requests, where the key is star, and the value is a function that, again, returns an object of requests. We start by creating a class that extends the Component function from React: class Counter extends React.Component [107] Proper Data Fetching The class has a constructor where the counter is initialized to 0, and the event handlers are bound to the component itself: constructor(props) { super(props) this.state = { counter: 0, } this.handleDecrement = this.handleDecrement.bind(this) this.handleIncrement = this.handleIncrement.bind(this) } The event handlers are simple, and they just change the state, adding or removing a unit from the current counter: handleDecrement() { this.setState({ counter: this.state.counter - 1, }) } handleIncrement() { this.setState({ counter: this.state.counter + 1, }) } Finally, inside the render method, the current value is displayed, and the buttons with their onClick handlers are defined: render() { return ({this.state.counter} - +) } [108] Proper Data Fetching Child-parent communication (callbacks) There are no major issues with this component, apart from the fact that it does multiple things: It holds the counter value into the state It is responsible for showing the data It contains the logic for incrementing and decrementing the counter It is always a good practice to split components into smaller ones, each with a very specific behavior, to improve the maintainability of the app and make it flexible when requirements change. The first thing to do is move to an empty folder and run the following: npm init This will create a package.json with some defaults. Thanks to that, we can easily describe how our components have to look at a given point in time in a declarative way and let the library do the rest. This way of working with data is called immutability. Suppose for example you created a React component that solves a complex problem and you think that other developers would benefit from using it instead of investing time in building their solutions. Let's now look at the different ways in which we can define our components with React and the reasons why we should use one or other technique. Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? Now, if we just want the input field to have a default value and then be able to change it by typing, we can use the defaultValue property: const Controlled = () => (Submit) In this way, the field is going to show Hello React when it is rendered, but then the user can type anything inside it and change its value. After those lines, we can see the output of the Perf add-on which measures the time we are wasting firing the render method of the Item components when the props are unchanged. Events Events work in a slightly different way across the various browsers. The library is smart enough to figure out the minimum set of operations required to apply the expected changes on the screen. First of all, with Enzyme it is possible to simulate events even with shallow-rendered elements, which we couldn't do with the TestUtils. We call the new component Gist, and we are going to use it inside the loop in the following way: const List = ({ gists }) => (gists.fulfilled && ({gists.value.map(gist => ( ))})) We just replaced the element with the Gist component, and we spread the gist object to it so that it receives single properties and becomes easier to test and maintain. Uncontrolled components Let's start with a basic example: displaying a form with an input field and a submit button. You can also review for us on a regular basis by joining our reviewers' club. Purity An important aspect of FP is to write pure functions. [147] Write Code for the Browser The props are defined in the following way: Circle.propTypes = { x: React.PropTypes.number, y: React.PropTypes.number, radius: React.PropTypes.number, fill: React.PropTypes.string, } This is great, because it makes working with SVGs and their properties more explicit so that the interface is clear and we know exactly how to configure our icons. Before going into the details, let's try to understand what it means to create applications that render both on the server and on the client. Remember that heavy packages have a bad impact on performance. Let's now move back to button.js and implement the event handler, modifying the render as follows: render() { return ({this.props.text}) } If we now run npm test again the tests are now green: PASS ./button.spec.js renders with text (10ms) fires the onClick callback (17ms) Test Suites: 1 passed, 1 total Tests: 2 passed, 2 total Snapshots: 0 total Time: 1.401s, estimated 2s Ran all test suites. We will start with a simple component that has no children, and we will transform it into a cleaner and structured component step by step. One solution would be to create a new separate handler and attach it to the button using the onDoubleClick attribute, as follows: Click me! [137] Write Code for the Browser Remember that we always aim to write less boilerplate and avoid duplicating the code. [284] Next Steps Distributing your code Contributing to the React ecosystem does not only mean pushing code into the React repository. Consider that your component is receiving a user object from an API, and this user object has many attributes. Let's start from the beginning by creating a component to test. And, most importantly, the object returned by the function is not a simple React element but rather ShallowWrapper, a special object with some useful properties and functions, which we'll look at next. Let's now create a new file, called geolocation.js, where we define the stateless functional component as follows: const Geolocation = ({ latitude, longitude }) => (Latitude: {latitude} Longitude: {longitude}) Stateless functional components are an incredibly elegant way to define UIs. They are pure functions that, given a state, return the elements. So, what the library does is apply the fade-appear class (where fade is the value of the transitionName prop) to the component as soon as it gets rendered. Also, it is good to learn the main differences between Jest, which is an integrated test framework and tries to automate all operations to provide a smooth developer experience, and Mocha, which does not make any assumptions on the tools you need. As discussed in Chapter 1, Everything You Should Know About React, React has a declarative programming approach that makes our code more readable. Every time the state changes, React renders the component again with the new state, which is why documentation often says that a React component is similar to a state machine. Common testing solutions In this last section about testing, we will go through some common patterns that are useful to know when testing complex components. It is also very useful for eliminating dead code, because when we delete a component for any reason, we can tell exactly which CSS it was using. It should be avoided because, in most cases, there is a more React-compliant solution to achieve the same result, but it is important to know that we have the possibility to do it and to know how it works so that we can make the right decision. It would be great to add the linting process inside our editor to get immediate feedback as soon as we type. Let's start with a very simple example: const Button = ({ text }) => {text} Button.propTypes = { text: React.PropTypes.string, } In the snippet above, we created a stateless functional component that receives a text prop of type string. Suppose we want to load Dan Abramov's gists on the server and return the list of items from the Express app we just created. For that reason, we have to find a way to load the data beforehand and pass it to the component as props. This rule is not strict, but it is best practice widely used in the React community to append Container to the end of the Container component name and give the original name to the Presentational one. There is a library which is incredibly useful for understanding that, and it is called reactlumberjack. At that point, we should start thinking about using a module bundler. For example, to render an HTML button, we use , while to render our Button components we use . For example, if we want to define a transition on webkit, we should use the WebkitTransition attribute, where the webkit prefix begins with a capital letter. The second problem that occurs when the state is mutated directly is that, whenever setState is called in any other part of the component, the mutated state gets applied unexpectedly. Now, suppose that we need to render a similar list but this time, we want to display a list of users received from the props rather than the state (to make clear that we can serve different scenarios): const UserList = ({ users }) => ({users.map(user => ({user.username} {user.bio && {user.bio}}))}) Given a collection of users, the code above renders an unordered list very similar to the posts one. In this case, as we said, it is easy to figure out which attribute we are passing and remove it but, if we use the spread operator, as in the following example: const Spread = props => We cannot control which properties are passed from the parent. Chapter 10, About Testing and Debugging, makes you realize that we all want our applications to be stable and handle all the edge cases: tests help with that. That is another feature of Next.js, which enables hot module replacement. With composition, we can reference classes from the same file or external dependencies and get all the styles applied to the element. [128] Write Code for the Browser This seems reasonable, and it works very well for a single field. Babel is a popular JavaScript compiler widely adopted within the React community. The first one is by using JavaScript functions, and the second one is by using JSX, an optional XML-like syntax. For example, with inline styles, it is not possible to use pseudo selectors (for example, :hover) and pseudo elements, which is a pretty significant limitation if you are creating a UI with interactions and animations. As we have seen in the previous section, when the attributes are too many or too long, it is a good practice to write them on a new line. Also, what if two sibling components need to share data with each other? www.PacktPub.com For support files and downloads related to your book, please visit www.PacktPub.com. We can think about a UI as a function to which is applied the state of the application, as follows: UI = f(state) [48] Clean Up Your Code We expect this function to be idempotent, so that it returns the same UI given the same state of the application. It can be installed in the browser by following this URL: hmhmm The extension adds a panel in the DevTools which we can use to start and stop the Perf add-on in an easy and convenient way without writing any code. A common practice that leads to fewer bugs is not to pass entire JavaScript objects down to children by reference, but to use their primitive values, which can be easily validated, making components more robust and error-proof. ISBN 978-1-78646-453-8 www.packtpub.com Credits Author Copy Editor Michele Bertoli Safis Editing Reviewer Project Coordinator Clay Diffrient Ulhas Kambali Commissioning Editor Proofreader Ashwin Nair Safis Editing Acquisition Editor Indexer Shweta Pant Rekha Nair Content Development Editor Graphics Onkar Wani Abhinash Sahu Technical Editor Production Coordinator Rashil Shah Aparna Bhagat About the Author Michele Bertoli is a frontend engineer with a passion for beautiful UIs. Born in Italy, he moved to London with his family to look for new and exciting job opportunities. As you may note here, splitting this component is not really needed because it just renders a h1 element. Let's see how it works. As discussed in the previous chapter, it is very helpful to indent our JSX following the tree structure of the elements, to improve the readability. Every time we run the tests, Jest creates new pictures and it compares them with the previous ones to check if something has changed. This is OK, and it works, but we want to fully control the value of the component and to do so, we should transform the component from a stateless functional one to a class: class Controlled extends React.Component As usual, we start defining a constructor where we initialize the state, this time with the default values of the fields. [249] About Testing and Debugging We could check the classes separately to make sure that one is not affecting the other but you should get the point. Let's start with a very simple example to understand what an enhanced component looks like. For the Media queries to work properly, especially with server-side rendering, Radium will inject the rules related to the media query in a style element inside the DOM, with all the properties set as !important. The last number of the package on the right represents the patch, and it should be increased when a new version of the library that contains bug fixes is pushed to npm. We use a class here because we want to store the current list inside the state and we also need an event handler for when the button is clicked: class List extends React.Component The List component has a constructor where we initialize the list and we bind the event handler: constructor(props) { super(props) this.state = { items: ['foo', 'bar'], } this.handleClick = this.handleClick.bind(this) } We then define the event handler that appends a new item to the list and stores the resulting array into the state: handleClick() { this.setState({ items: this.state.items.concat('baz'), }) } Finally, we specify the render method where we loop through the items displaying every single element of the list and declaring the button with its onClick event handler: render() { return ({this.state.items.map(item => {item})} +) } [201] Improve the Performance of Your Applications The component is ready and if you add it to your application (or if you create a new application using create-react-app to try it), you will see that the foo and bar items are shown on the screen and that clicking the + button adds baz at the end of the list. [117] Proper Data Fetching The library is called react-refetch, and it is maintained by developers from Heroku. With the Page Object the code looks much cleaner and has no unnecessary repetitions. Don't forget to avoid the common mistakes that make the PureComponent less effective, such as generating new functions inside the render method, or using constant as props. Developers tend to think carefully about size when they have to decide whether to use a library or not. As we saw in the first chapter, you can easily use React by just adding a script tag pointing to unpkg.com. It represents a nice example because it has various props, its class names change according to the props it receives, and it has three event handlers that implement a bit of logic that we can test. However, there are some particular cases where the optimizations are not enough and we want to help the library to give us the best possible experience. Here comes the magic and the simplicity of the server-side rendering of React. However, pushing the code to GitHub is only a small action within a big process and it comes with some responsibilities. Now, we can easily fix the issue by modifying the extends statement of the Item component from extends React.Component to: class Item extends React.PureComponent If we now open the browser and run the application again, clicking the + button we see that nothing is logged into the console which means that we are not rendering any Item for which the props are not changed. [114] Proper Data Fetching Note that the property we use to store the data is now called data because we are building a generic component and we do not want it to be tied to a particular object shape or collection: constructor(props) { super(props) this.state = { data: [] } } Inside the componentDidMount hook, the fetch function is fired and the data returned from the server is converted to JSON and stored into the state: componentDidMount() { fetch(url) .then(response => response.json()) .then(data => this.setState({ data })) } It is important to note that now the URL is set using the first parameter received from the HoC. To use JSX without any problems or unexpected behaviors, it is important to understand how it works under the hood and the reasons why it is a useful tool for building UIs. Our goal is to write clean and maintainable JSX code, and to achieve that, we have to know where it comes from, how it gets translated to JavaScript, and which features it provides. This represents a powerful feature, but it should be used carefully because it breaks some of the conventions that make React easy to work with. We create a form with an input element inside it and we define a function on its ref attribute. Frontend developers tend to use pre-processors to be able to split their CSS into submodules, but in the end a big, global CSS bundle is generated for the browser. With React, we should always be explicit to avoid confusion. In this case, we will learn how to avoid a common error when using the PureComponent, which makes it less effective. The type is important because it tells React how to deal with the element itself. The first one was the so-called Container and Presentational pattern, which helps us separate the logic from the presentation and create more specialized components with a single responsibility. Then, we create a couple of variables that we will use within the tests: const data = 'data' const List = () => This is the mock data that we use to check whether the information is correctly passed from the fetch to the component, and a dummy List component. That is why the classic separation of concerns ended up being more a separation of technologies, which is of course not a bad thing, but it does not solve any real problems. It is important to say that we should never optimize an application before having measured its performance and understood where the bottlenecks are. In JSX, we can use an inline condition: {isLoggedIn && } This works because if the condition is false, nothing gets rendered, but if the condition is true, the createElement function of the LoginButton gets called and the element is returned to compose the resulting tree. The last one is the blur handler: handleBlur = e => { if (!this.props.newTodo) { this.props.onSave(e.target.value) } } This fires the onSave prop function with the value of the field if the prop newTodo is false. In this small example, this does not create any visible performance issue but, if you think about a real-world application with hundreds of elements, this could become a major problem. A new developer joining the company can just look at the existing stories to figure out if there is any need to create a new component or if an existing one already provides a solution to a particular problem. When the componentDidMount hook gets fired, we call the fetch function against the GitHub APIs. The fetch function returns a Promise and, when it gets resolved, we receive a response object with a JSON function that returns the JSON content of the response itself. This uncontrolled behavior is something we want to avoid as well. Finally, we have seen what the new create-react-app CLI is, and we are now ready to start writing some real code. For example: const Button = React.createClass({ render() { return }, }) const component = ReactTestUtils.renderIntoDocument() In this case, the component represents our Button. FP and user interfaces The last step to take is to learn how we can use FP to build UIs, which is what we use React for. What really happens is that nothing is displayed on the screen. The following test is a very common one: test('updates the text on change', () => { const value = 'value' const wrapper = shallow() wrapper.simulate('change', { target: { value } }) expect(wrapper.prop('value')).toBe(value) }) This test checks that the controlled input works properly and, if you think about the forms we discussed in Chapter 6, Write Code for the Browser, this is a must-have for all the forms in your application. We also define the prop types and the context types for both values. [64] Create Truly Reusable Components First of all, we should always keep in mind that only the minimal amount of data needed should be put into the state. JSX has been built in such a way that it only abstracts the creation of the elements, leaving the logic parts to real JavaScript, which is great except that sometimes, the code becomes less clear. SEO is certainly one of the main reasons, but social sharing and performance are important factors as well. This allows us to introduce the concept of Higher-order Functions (HoF). Summary Knowing all the best practices is always a good thing, but sometimes being aware of antipatterns help us avoid taking the wrong path. One solution we can use to solve the problem is to create a prop called domProps that we can spread safely to the component because we are explicitly saying that it contains valid DOM properties. We assume that the title field is always present, and we show it inside an while the excerpt is optional, and we show it inside a paragraph only if it exists. It is not only wrong on principle, but it also leads to many errors in big codebases, and it makes maintainability very hard in the long term. What does it mean for a function to be pure? Let's now go through the problems of mixins and, in the next section, we will see which is the best technique to achieve the same result without all the issues. ESLint is an open-source project released in 2013 that became popular thanks to the fact that it is highly configurable and extensible. Let's see how they work. Testing components only in the console can sometimes give unexpected behaviors when they are rendered in a real browser but, in my experience, this is rare. [289] Next Steps Summary In the last stop in this trip around the React world, we have seen some of the aspects that make React great: its community and its ecosystem, and how to contribute to them. A good practice is to avoid external dependencies as much as we can to keep our bundle smaller, but it may be worth it in this particular case because improving the readability of our templates is a big win. Chapter 9, Improve the Performance of Your Applications, informs that on the Web, performance is one of the most important factors to engage users. The buttons are now dumb, and they are only able to notify their owner when they are clicked on. Finally, there is the render method, where the input element is defined with all its properties: render() { return ( ) } [247] About Testing and Debugging To apply the class name, the classnames function is used; it comes from a very useful package, authored by Jed Watson, which makes it simple to apply classes using a conditional logic. Finally, you have learned how the React Developer Tools help debugging and how to approach error handling in React. It renders a console at the bottom of the page from which we can start and stop monitoring. Let's dive into a real-world example. The rules relate to the type of the property as well as to whether the property is optional or required. This is one of the reasons why co-locating the styles within our components and applying them using inline styles on the elements could make sense in React. We create a class because we have to store the state, and we need an event handler: class FontSize extends React.Component The class has a constructor, where we set the default value for the state, and we bind the handleChange handler, which listens to the onChange event of the input field: constructor(props) { super(props) this.state = { value: 16, } this.handleChange = this.handleChange.bind(this) } We implement a simple change handler, where we use the target attribute of the event to retrieve the current value of the field: handleChange({ target }) { this.setState({ value: Number(target.value), }) } Finally, we render the input file of type number, which is a controlled component because we keep its value updated by using the state. So, here is an example of how ESLint can be useful to help everyone in the team follow a style guide without having to memorize it. Since the spread operator hides the single properties we are spreading, it makes even harder to figure out what we are passing to the element. At this point, we enhance our component with the function we just created, as follows: const ListWithGists = connectWithGists(List) We now have to make small changes to the initial component to make it work well with the new HoC. Jest is an all-in-one testing framework solution, maintained by Christopher Pojer from Facebook and contributors within the community, and aims to give you the best developer experience; but you can decide to create a custom setup with Mocha as well. When React has to display a component, it calls its render method and the render methods of its children recursively. Please contact us at with a link to the suspected pirated material. By logic, we refer to anything that is unrelated to the UI, such as API calls, data manipulation, and event handlers. In this case, we can generate a function inside the render because there are no pure children. Let's see what solutions we can adopt and in which scenario we should prefer one or another. In this way, we avoid polluting our component with conditionals so that it is easier to understand and reason about. Let's look at an example to clarify the concept. Sharing code also gives you some responsibilities and it could be time-consuming. As the name suggests, React Storybook lets you write stories which represent the possible states of the components. With React, there are different ways of doing it and each one tackles the problem from a different perspective. React is great because it gives us a declarative API to use in all cases, but we can also access the underlying DOM nodes and components instances in case we need them to create more advanced interactions and complex structures. Suppose you are working along with the designer on the UI side of the component, where latitude and longitude are displayed to the user. The sixth issue concerns the non-deterministic resolution of CSS. For example, we want to add a click event handler on each item so that, if the item gets clicked, we log its value into the console. [123] Proper Data Fetching Summary The journey through data fetching in React has come to an end and now you know how to send and retreive data to and from API endpoints. It is a really useful way to exchange information with members of the team who have different skills, keeping the style consistent as time passes and as the number of components increases. For example, we can recalculate some CSS values on the client at runtime, which is a very powerful concept, as you will see in the following example. During componentDidMount, the API call gets fired, and as soon as the data is available, the posts are stored in the state. [139] Write Code for the Browser Suppose we want to create a simple form with an input element and a button, and we want it to behave in such a way that when the button is clicked, the input field gets focused. Jest is an all-in-one testing framework and it gives us all the tools we need to write a proper test. When creating a build for the server, in fact, we do not want to add to the bundle all the node packages that we use; we just want to bundle our server code. There are two lifecycle hooks where we can put the data fetching: componentWillMount and componentDidMount. Animations When we think about UIs and the browser, we must surely

because React is split into two packages: react, which implements the core features of the library, and react-dom, which contains all the browserrelated features. We first change the component to declare single properties, as follows: const Profile = ({ username, age }) => (Username: {username} Age: {age}) Profile.propTypes = { username: React.PropTypes.string, age: React.PropTypes.number, } Then, we enhance it with the HoC: const ProfileWithFlattenUser = flattenProp('user')(Profile) You may have noted here that we are using the HoC in a slightly different way. Greater confidence in writing new features leads to faster releases. This is an example of an Uncontrolled Component, where we do not set the value of the input field, but we let the component manage its internal state. Looking at the component, it is pretty easy to figure out which are the important parts to cover. [174] Make Your Components Look Beautiful Styled Components There is a library which is very promising, because it takes into account all the problems the other libraries have encountered in styling components. It is almost impossible to do that using client-side-only applications, because the engine that extracts the information from the pages uses the markup returned by the server. However, sometimes it helps just to keep the render method cleaner. We are now ready to write some code, and we will start from the React application, which we are more familiar with. It is also crucial to understand how unconnected siblings can share their data. Preface Chapter 5, Proper Data Fetching, instructs that any client-side application has to deal with data at some point; it takes you through the different techniques and approaches that can be used to fetch data in the React-way. With Webpack, it is pretty straightforward because we can have multiple configuration files that can be used in the different stages of our application life cycle. Suppose we want to add a prop to our Item component which represents a list of statuses that the item can have. If something changes in the component, instead of updating multiple tests, we can just modify the way the Page Object works in a transparent and easy way. They make the applications faster, optimizing parts of the code at build-time. [233] About Testing and Debugging As we said, we want to test that the component renders correctly given some props, and that the event handlers do their job. Also, if you are debugging a list where every single item has the same style attribute, and if you modify one on the fly to check the result in the browser, you will see that you are applying the styles only to it and not to all the other siblings, even if they share the same style. Using it is pretty straightforward; we just install it: npm install --global babel-cli [21] Clean Up Your Code If you do not want to install it globally (developers usually tend to avoid this), you can install Babel locally to a project and run it through an npm script, but for the purposes of this chapter, a global instance is fine. With react-refetch, we can do the same thing; so, first of all, we install the library: npm install react-refetch --save Then, we import the connect function inside our module: import { connect } from 'react-refetch' Finally, we decorate our component using the connect HoC. First of all, we should rename our Geolocation component to GeolocationContainer: class GeolocationContainer extends React.Component We will also change the filename from geolocation.js to geolocation-container.js. The component receives the description and, for now, the only difference from the previous markup is that it has a +1 button, to which we will add some functionalities soon: const Gist = ({ description }) => ({description} +1) Gist.propTypes = { [120] Proper Data Fetching description: React.PropTypes.string, } The URL of the endpoint to star a gist is the following: id/star?access_token=:access_token Here, :id is the id of the gist that we want to star, and the access token is the authentication token required to run the action. React Design Patterns and Best Practices Build modular applications that are easy to scale using the most powerful components and design patterns that React can offer you right now Michele Bertoli BIRMINGHAM - MUMBAI React Design Patterns and Best Practices Copyright © 2017 Packt Publishing All rights reserved. They are scoped locally as well, so we have all the advantages of CSS Modules. It is not very intuitive the first time you see it. The PureComponent decides if the component should be updated by checking if the values of its props and state are changed but, in this case, we passed the same array again. Mutating the state React comes with a very clear and straightforward API to mutate the internal state of components. This approach will let us see the best pattern to apply in each phase in order to let the data flow across the tree. As you might have already experienced, most other UI libraries work in the opposite way: they leave the responsibility of keeping the interface updated to the developer, who has to manage the creation and destruction of the DOM elements manually. Not only does a linter help us to make fewer errors, or at least, find those errors sooner, but it enforces some common coding style guides, which is really important, especially in big teams with many developers, each one with their favorite coding style. In a real-world scenario you could send the data to an API endpoint or pass it to another component. The button we will start with is as follows: const Button = () => Click me! First, we have to install Radium using npm: npm install --save radium Once the installation is complete, we can import the library and wrap the button into it: import radium from 'radium' const Button = () => Click me! export default radium(Button) The radium function is a Higher-Order Component (see Chapter 4, Compose all the Things), which extends the functionalities of our Button, returning a new enhanced component. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy. [18] 2 Clean Up Your Code This chapter assumes that you already have experience with JSX and you want to improve your skills to use it effectively. That is why the PureComponent should be used only when it is needed and only once the performance have been measured and you have figured out which components are taking too much time to be executed. This is where the Page Object pattern come to the rescue. Whenever you can, you should keep the package small and add as few dependencies as you can. Being able to share the logic between the client and the server makes it easy to apply any changes on both sides without doing the work twice, which most of the time leads to fewer errors and fewer problems. This is a very common data fetching pattern, and we will see the other possible approaches in Chapter 5, Proper Data Fetching. There is one cool feature in Webpack that lets us export an array of configurations as well so that we can define both client and server configurations in the same place and use both in one go. In this way, the component is unaware of the context, and it can be easily reused in different parts of the application. We also call the onSubmit prop function received from the parent, passing it the value of the fields concatenated. If we then pass the schema to the Form component we imported from the library, a form will be generated for us automatically. The field fires a callback on the common parent to update the list when the form is submitted. In this chapter we will see the following: How reconciliation works and how we can help React do a better job using the keys How using the production version of React makes the library faster Improve the Performance of Your Applications What shouldComponentUpdate and PureComponent can do for us and how to use them Common optimization techniques and common performance-related mistakes What it means to use immutable data and how to do it Useful tools and libraries to make our applications run faster Reconciliation and keys Most of the time, React is fast enough by default and you do not need to do anything more to improve the performance of your application. [206] Improve the Performance of Your Applications React is not able to figure out which components do not need to be updated but we have a function that we can implement to tell the library when to update a component or not. We change the render method in the following way, adding an input field near each item. Again, we set the opacity to 0.0.1 to hide the element and start the fade. For those who think that inline styles are a bad solution, we moved into the world of CSS Modules, setting up a simple project from scratch. [116] Proper Data Fetching That is pretty straightforward to do; we just have to change the componentDidMount hook, as follows: componentDidMount() { const endpoint = typeof url === 'function' ? Most likely, we want to do something with the value of the element when the submit button is clicked. For example, if we have an initial state as the following one: this.state = { text: 'Click me!', } And we run setState with a new parameter: this.setState({ cliked: true, }) The resulting state is: { cliked: true, text: 'Click me!', } Every time the state changes React runs the render function again, so there's no need for us to do anything other than setting the new state. The following rule applies a height of 100px: const style = { height: 100, } Using inline styles works, and we can also do things that are hard to implement with regular CSS. Alternatively, you can go a step further and, as we have just seen, use getters: get price() { return `${this.props.currency}${this.props.value}` } {this.price} Going back to conditional statements, there are other solutions that require using external dependencies. In this chapter we will see: The different ways we can follow to create React components and when we should use one rather than the other What the stateless functional components are, and what's the difference between functional and stateful ones How the state works and when to avoid using it Why it is important to define clear prop types for each component and how to generate documentation dynamically from them with React Docgen A real example of transforming a coupled component into a reusable one How we can create a living style guide to document our collection of reusable components using React Storybook Create Truly Reusable Components Creating classes We have seen in the first chapter how React uses elements to display the components on the screen. Testing the main functionalities of an application makes the codebase more solid and, whenever a new bug is found, it can be reproduced, fixed, and covered by tests so that it does not happen again in the future. With Jest it is very easy to isolate and mock the dependencies. It is worth stressing that the technique used by this library should be avoided, and it could create some problems with your application. Returning false all the time is far from useful and so developers usually check if the props or the state are changed inside that method. In fact, the variable user does not exist before the transpilation, nor is it wrapped into a function. Being able to work in parallel on the same component is a big win for teams, especially for those companies where building interfaces is an iterative process. We often need to know the height of a header, for example, to recalculate the position of other elements that depend on it. The best way to learn is by seeing examples, and we will do that by starting from a component which serves a single purpose and transforming it into a reusable one. We can easily fix this warning by setting the children prop to accept the following prop types: Button.propTypes = { children: React.PropTypes.oneOfType([React.PropTypes.array, React.PropTypes.element,]), } Container and Presentational pattern In the last chapter, we saw how to take a coupled component and make it reusable step by step. The withData function has the following signature: const withData = URL => Component => (...) It takes the URL of the endpoint where the data has to be loaded, and it provides such data to the target component. However, Packt Publishing cannot guarantee the accuracy of this information. There are some languages that can be converted into JavaScript at build time, but the concept does not change. Rendering only one level means that we can test our component in isolation and, even if it has some complex children, they do not get rendered and they do not affect the results of the test if they should change or fail. If we write a function that returns an array inside our JSX template, each element of the array gets compiled into an element. For example, using Open Graph, we can tell Facebook that, for a particular page, we want a certain image to be shown and a particular title to be used as the title of the post. Now, if we want to use the mixin in our component, we just have to assign it to the array of mixins, which is a property of the object: const MyComponent = React.createClass({ mixins: [WindowResize], render() { console.log('window.innerWidth', this.state.innerWidth) ... Now, create-react-app does not support SSR yet, but there's a company called Zeit who created a tool called Next.js, which makes it incredibly easy to generate Universal applications without worrying about configuration files. All the dependencies are now available and we can write a simple test: test('snapshots are awesome', () => { In the first line we render the component using the renderer that we previously imported: const component = renderer.create({}} />) [253] About Testing and Debugging What we receive back is the instance of the component, which has a special function called toJSON that we call in the following line: const tree = component.toJSON() The resulting tree is the React element returned from the original component that Jest will use to generate the snapshot to be compared with the next ones. A common solution to get the same output, is putting a space explicitly between the elements: foo {' '} bar {' '} baz As you may have noticed, we are using an empty string wrapped inside a JavaScript expression to force the compiler to apply the space between the elements. Sharing your code, in fact, on the one hand forces you to follow best practice and write better code. SEO One of the main reasons we may want to render our applications on the server-side is Search Engine Optimization (SEO). Props are the way a parent component can pass its data down the tree to every component that needs it (or part of it). Let's look at a simple example where we draw a blue circle, creating a React component which wraps an SVG element: const Circle = ({ x, y, radius, fill }) => ( ) As you can see, we can easily use a stateless functional component which wraps the SVG markup, and it accepts the same props as the SVG does. In the Functional Programming section of Chapter 2, Clean Up Your Code, we mentioned the concept of Higher-order Functions (HoFs), which are functions that, given a function, enhance it with some extra behaviors, returning a new one. One is what we have done until now: run it from the command line and get the list of errors. Now we create an Express application: const app = express() We tell the application where our static assets are stored: app.use(express.static('dist/public')) As you may have noticed, the path is the same that we used in the client configuration of Webpack as the output destination of the client bundle. However, there might be some cases where you need to access the underlying DOM nodes to perform some imperative operations. We have to set up and maintain a server with its routes and its logic, manage the server data flow, and so on. [214] Improve the Performance of Your Applications Constants props Let's keep on improving our list example and see what happens when we add new features. Stateless functional components Another concept that is sometimes counter-intuitive for beginners is the fact that stateless components do not give us any benefits regarding performance. When this functionality is enabled we can use our application and visually see which components get updated when we perform a particular action. Once the recommended rules are enabled and we run ESLint again, we should not receive an error for the semicolon (which is not part of the recommended configuration), but we should see the linter complaining about the fact that the foo variable has been declared and never used. Again the Operations and Payload columns are the most interesting ones and we can see that React has applied three changes: It has replaced the text of the first item with baz, the new value It has replaced the text of the second item with foo, the value of the former first item It has inserted a new child at index 2, at the bottom of the list React, instead of just adding the new element at the top of the list and shifting the other ones down, has changed the values of the previous ones and added a new item at the end of the list. Those advanced features make our code cleaner for developers, but the browser cannot understand and execute it. Choose from the drop-down menu where you purchased this book from. Congratulations! Your first test on a component written using TDD passed. Let's create a new form with first and last name fields. So, we should consider carefully before enabling it for one of the preceding reasons and check if our team is ready to support and maintain a Universal application. You'll learn the difference between Shallow rendering and full DOM rendering with both TestUtils and Enzyme, how Snapshot Testing works, and how to collect some useful code coverage information. However, if we load a huge JavaScript bundle using a mobile device with a 3G connection, users have to wait for a little while before interacting with the application. We have seen how to test components using mocks and writing expectations. In the constructor, the initial state gets defined, and the handleResize callback is bound to the current class: constructor(props) { super(props) this.state = { innerWidth: window.innerWidth, } this.handleResize = this.handleResize.bind(this) } The lifecycle hooks and the event handler are identical to the mixin's: componentDidMount() { window.addEventListener('resize', this.handleResize) } componentWillUnmount() { window.removeEventListener('resize', this.handleResize) } handleResize() { this.setState({ innerWidth: window.innerWidth, }) } [96] Compose All the Things Finally, the original component gets rendered in this way: render() { return } As you may note here, we are spreading the props as we saw before, but we are also spreading the state. First of all, let's move into an empty folder and create a new project: npm init When this is done, we can install the library: npm install --save next Now that the project is created, we just have to add an npm script to run the binary: "scripts": { "dev": "next" }, Perfect, it is now time to generate our App component. We also saw how using the wrong key attribute can produce bad effects on the reconciliation algorithm; and, finally, we learned why spreading non-standard props to DOM elements is considered an anti-pattern. The different browsers and environments, and real user data, are all variables that we cannot control and sometimes our code will fail. The first makes it easy to test the behavior of the functions inside a test. What you learned in this section is how to test Higher-Order Components using a dummy child and how to isolate the current component mocking external dependencies. For example, if we change the constructor of the Counter component in the following way: constructor(props) { super(props) this.state = { count: props.initialCount, } this.handleClick = this.handleClick.bind(this) } And then we use it like this: It is clear that the parent only has a way to initialize the counter but any future values of the initialCount prop will be ignored. The function used to validate the condition receives the props, state, and context of the component. In fact, our is translated into React.createElement('div') when we run Babel, which is something we should always keep in mind when we write our templates. In fact, if we serve an empty HTML skeleton to the crawlers of the main search engines, they are not able to extract any meaningful information from it. To start and stop tracking and display the results, we can simply use two lifecycle hooks that React provides. As the name suggests the shallow function provides Shallow rendering functionalities and it has some special features. State The name stateless tells us clearly that the stateless functional components do not have any internal state, and the fact that this does not exist enforces it. Please notice that setTimeout has been used in the example only to show the behavior of React but you should never write event handlers in that way. Maintaining a repository requires commitment because the more popular it gets and the more people use it, the higher the number of questions and issues will be. [4] Preface Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of: WinRAR / 7-Zip for Windows Zipeg / iZip / UnRarX for Mac 7-Zip / PeaZip for Linux The code bundle for the book is also hosted on GitHub at ishing/React-Design-Patterns-and-Best-Practices. From the previous section, we have a List component, which is a stateless functional component that can receive a collection of gists; it displays the description for each one of them: const List = ({ data: gists }) => ({gists.map(gist => ({gist.description}))}) List.propTypes = { data: React.PropTypes.array, } By wrapping this component inside the withData HoC, we are able to provide data to it in a transparent way through props. When we click on one of those stories, we can see the components being rendered in the right area. We started from a deep study of the basics and seeing the differences between stateful and stateless components, and we saw an example of how to make a tightly coupled component reusable. Piracy Piracy of copyrighted material on the Internet is an ongoing problem across all media. To give back to the community and help developers, you can create packages, blog posts, answer questions on Stack Overflow, and many other activities. The key prop uniquely identifies an element in the DOM and React uses it to check if the element is new or if it has to be updated when the component props or state change. If multiple components need to keep track of the same information, we should consider using a state manager like Redux at the application level. ESLint helps us to maintain consistency with the jsx-max-props-per-line rule so that every component is written in the same way. The second thing to do is define the style of the element we are creating, and to do so, we use a ES2015 feature called Tagged Template Literals, which is a way of passing template strings to a function without them being interpolated beforehand. To do this, we have to change the template a bit: export default (body, gists) => ` ${body} window.gists = ${JSON.stringify(gists)} ` [193] Server-Side Rendering for Fun and Profit The template function now accepts two parameters: the body of the app and the collection of gists. Importing the CSS files into our components makes the dependencies clear, and scoping the class names locally avoids clashes. To do that, there is a handy JSFiddle template provided by the React team: This operation has two key benefits: It helps you to be 100% confident that the bug is effectively a React bug and not just an issue with your application code It helps the React team understand the problem quickly without having to delve into your application code, making the process faster The Fiddle uses the latest version of React and this is important because, if you find a bug in an older version of the library, it may already have been fixed in the current one. [144] Write Code for the Browser On the next tick, the class fade-appear-active is applied so that we can fire our animation from the initial state to the new one, using CSS. Inside the Express route, we just have to change the line where we generate the template passing the body, as follows: const html = template(body, gists) Last but not least, we have to use the gists attached to window inside the client.js, which is pretty easy: ReactDOM.render(, document.getElementById('app')) We read the gists directly and we pass them to the App component that gets rendered on the client. The problem is that sometimes is not easy for new developers to find out if components already exist or if new ones are needed. The test starts like this: test('renders with text', () => { As a first operation we define the text variable that we pass as a prop and that we will use to check if the right value is rendered: const text = 'text' Now it is time for Shallow rendering, which is implemented in these three lines: const renderer = TestUtils.createRenderer() renderer.render() const button = renderer.getRenderOutput() The first one creates the renderer, the second one renders the button passing the text variable, and finally we get the output of the rendering. However, if you open your DevTools and try to force the :hover status in the Styles panel, you will see that nothing happens. before onClick() { return console.log(item); } after onClick() { return console.log(item); } [212] Improve the Performance of Your Applications } Item.props.onClick Value is a function. [151] Make Your Components Look Beautiful The second problem with CSS regards the definition of the dependencies. We will also see some common mistakes that can harm our applications and make them slower. handleSubmit(e) { e.preventDefault() console.log(this.state.value) } This handler is pretty straightforward: we just log the value currently stored in the state. The first row tells Mustache to loop through a collection of items. The Container knows everything about the logic of the component and it's where the APIs are called. First of all, it is important to clearly understand the problems it can solve for us and why we need to add a step to our process. [146] Write Code for the Browser It is also interesting to compare the two different approaches of the transition group and react-motion to be able to choose the right one for the project you are working on. 7. Since the Presentational component is usually logic-less, we can create it as a functional stateless component. We used it in all the examples in this book and you have seen how easy it is to install a package: It is just a matter of running npm install package, and that is it. To start with, we'll create only the skeleton, and we will write the implementation following the tests, using the TDD approach. A common code base We do not have many options on the client: our applications have to be written in JavaScript. Moving the component away is not a problem because we have seen how components in different parts of the application can receive the data they need from their parents. For example, if we use our WindowResize mixin in a component where we also define a componentDidMount hook, both will be executed in order. This is how the render method of a React component looks: render() { return (Click me!) } We all agree that it looks a bit weird in the beginning, but it is just because we are not used to that kind of syntax. [13] Everything You Should Know About React That is why, even if the idea of mixing JavaScript and HTML sounds weird in the beginning, it is important to give React five minutes. The only change is the value of the input element and that is going to be the only mutation applied to the DOM; but, to figure out what operations are needed, React has to render the entire component and all its children and rendering a big list of items many times over is expensive. This is great, and it is an incredibly useful tool. Let's look at how it works: the API is pretty similar; we just have to create a new attribute on our style object and nest the styles that must be applied when the media query matches inside it: const styles = { backgroundColor: '#ff0000', width: 320, padding: 20, borderRadius: 5, border: 'none', outline: 'none', ':hover': { color: '#fff', }, ':active': { position: 'relative', top: 2, }, '@media (max-width: 480px)': { width: 160, }, } There is one thing we must do to make Media queries work, and that is wrapping our application into the StyleRoot component provided by Radium. The examples in this section use the fetch function to make web requests, which is a modern replacement for XMLHttpRequest. Christopher Pojer, a Facebook engineer, said on Twitter that they enabled the server-side rendering on Instagram only for SEO reasons because, in the case of a highly dynamic content website such as Instagram, SSR is not useful to improve the user's perception of the speed: A basic example We will now create a very simple server-side application to look at the steps that are needed to build a basic Universal setup. You can even choose to force a non-indentation using 0 as a second parameter. To do that we cannot render the components using ReactDOM, because it needs the browser's DOM. The collected props are spread and a className property with the "my-class" value is passed to it. So, the SVG is just a template, and we can use the same Circle multiple times in our application, with various props. If you realize that your application takes a lot of time to fully load and you have already done all the optimization (see the following chapter for more about this topic), you can consider using server-side rendering to offer a better experience to your users and improve the perceived speed. The best way to solve it is to bind the function inside the construction in a way that it doesn't ever change even if the component renders multiple times: class Button extends React.Component { constructor(props) { super(props) this.handleClick = this.handleClick.bind(this) } handleClick() { console.log(this) } render() { return } That's it, problem solved! Stateless functional components There is one more way to define our components, and it is very different from the previous two. After the events, we will jump into refs to look at how we can access the underlying DOM nodes in our React components. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title. It has many cool features, such as the server-side rendering that gives you the power to write Universal applications. [110] Proper Data Fetching In general, our goal is to make components unaware of the data source so that we can reuse them with different sources in various parts of the application. This is great, not only for SEO, but also for social sharing services. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book. Compose All the Things Communication between components Reusing functions is one of our goals as developers, and we have seen how React makes it easy to create reusable components. Let's modify the render method of PostsLists as follows: render() { return ( ) } And the UserList function as follows: const UserList = ({ users }) => ( ) We went from a single-purpose component to a reusable one using the props to create a generic and well-defined interface. This is great and it is more than enough for the majority of the use cases. [285] Next Steps If you decide to ask people to co-maintain the project and help you with issues and Pull Requests you have to coordinate with them to share your vision and make decisions together. */ text: React.PropTypes.string, } If we run the command again, the result is: { "description": "A generic button with text.", "methods": [], "props": { "text": { "type": { "name": "string" }, "required": false, "description": "The text of the button." } } } We can now use the returned object to create the documentation and share it across our team or publish it on GitHub. Small components with a clean interface can be composed together to create complex applications that are powerful and maintainable at the same time. Now, suppose we want to apply some styles to the button: a background color, the size, and so on. It's important to know which libraries are available and how they work in order to choose the right one. Another significant opportunity that open source can give you is letting you get in touch with smart and passionate developers from all around the world. This coupling between mixins makes it very hard to refactor the components and scale the application. As shown in the example, templates highly depend on the models they receive from the logic layer to display the information. We can enable and disable every single rule manually, or we can enable the recommended configuration in one go by putting the following code into our .eslintrc: { "extends": "eslint:recommended" } The extends key means that we are extending the recommended rules from the ESLint configuration, but we can always override single rules manually inside our .eslintrc using the rules key, as we have done before. In fact, one of the reasons why React is so powerful is because it enforces a declarative programming paradigm. [191] Server-Side Rendering for Fun and Profit Now we have to change the server to retrieve the gists and pass them to the component. [223] Improve the Performance of Your Applications One way to solve this issue is using immutable data: Data that, once it gets created, cannot be mutated. This is very powerful because we have the full power and expressivity of the CSS combined with the advantages of having locally scoped class names and explicit dependencies. Spreading props on DOM elements There is a common practice that has recently been described as an anti-pattern by Dan Abramov; it also triggers a warning in the console when you do it in your React application. Another pretty common operation that is non-trivial with regular CSS is sharing constants between the styles and the client application. 4. [2] Preface What you need for this book We will need a computer with a terminal, a node.js/npm environment, and a browser. The Button we used in the previous example was great and we should always try to make our components as simple as possible; but sometimes we need to implement different kinds of logic and state, which makes the tests more challenging. As you may notice, this does not scale very well when there are lots of fields, but it is important to clearly understand the problem before moving to a more flexible solution. We pass the condition as the first parameter of the onlyIf function and when the condition is matched, the component is rendered. To understand where it comes from, you just have to check the implementation of the render method: render() { return ((this.element = element)} /> Focus) } Here comes the core of the logic. That is great, and it looks easy, but if we check the DevTools console we can see the following error: Cannot read property 'map' of undefined The reason we see the error is that, on the client, we are rendering the App again but without passing any gists to it. We then use the getDefaultProps function to define the values that the props are going to have by default and which will be overwritten by the props passed from the parent, if they are present. The gists prop is not the actual data, but it is a particular type of object, called PromiseState. In this case we describe the behavior of the button: describe('Button', () => { And then we write the first test where we expect the type and the text to be the correct ones: it('renders with text', () => { [240] About Testing and Debugging We define the text variable that we will use to check the validity of the expectation: const text = 'text' We then shallow-render the components as we did before: const renderer = TestUtils.createRenderer() renderer.render() const button = renderer.getRenderOutput() And we finally write the expectations: expect(button.type).to.equal('button') expect(button.props.children).to.equal(text) As you can see, here the syntax is slightly different. You should avoid spending time writing code and submitting a Pull Request before asking them because the feature you have in mind might not fit into their plans or might conflict with other functionalities they are working on. We can change the snippet to use plain JavaScript as follows: const Button = React.createClass({ render() { return React.createElement('button') }, }) We can run the code everywhere without needing to use Babel for transpiling, which is a good way to start with React, avoiding the effort of learning different tools in the React ecosystem. The best way to get started with a new technology is to try it in a small side project and see how it goes. Finally, we describe our view in the render method: render() { return ({this.state.items.map(item => {item})} +) [218] Improve the Performance of Your Applications } We have the unordered list, where we loop through the items collection and we print every single item inside a element. Jest is a fully-featured tool while Mocha lets you customize your experience. In the first section, we will do a little step back, but please bear with me because it is crucial to master the basics in order to apply the best practices. The first component that we will build is a simple list of the gists created by the user, called gaearon (Dan Abramov). In our case, we could do something like this: shouldComponentUpdate(nextProps, nextState) { return this.state.items !== nextState.items } We return true only if the items are

changed, while we stop the rendering in any other case. The reason it does that is because, in React, we declare what we want to see on the screen, and setting a fixed value attribute ends up always rendering that value, no matter what other actions are taken. A real-world testing example We have a working test set up and a clear understanding of what the different tools and libraries can do for us. The value of the body variable is something like the following: Hello React As you can see, it represents what we defined in the render method of our App, except for a couple of data attributes that React uses on the client to attach the client-side application to the server-rendered string. In this book, we have been very lucky. It is to create applications using React components and how they work within the browser. Inside the .storybook folder we create a config.js file to load our stories: import { configure } from '@kadira/storybook' function loadStories() { require('../src/stories/index') } configure(loadStories, module) We first load the configure function from the library and then we define a function to load each single story using their paths. Starting with a boilerplate that has one hundred dependencies and tens of paces of packages of which we know nothing is the best way to get lost. Since styles are declared using JavaScript objects, it is also not possible to use style fallbacks: display: -webkit-flex; display: flex; JavaScript objects, in fact, cannot have two attributes with the same name. Using the same language on both sides not only makes it easy for developers to reuse their knowledge, it also enables different ways of sharing code between the server and the client. Here is where the Choose component comes in handy: if else if else Please note that the preceding code gets transpiled into multiple ternaries. [87] Compose All the Things In this case, we just need a component to display the latitude and longitude, so we are going to use a simple function. There are also different callbacks, such as onChange, which is fired every time the value of a field changes, and onError, which is fired whenever the form is submitted using invalid data. We can now wrap any component of our application and provide data to them from any URL. With a single class, we cannot easily create a proper animation, and that is why the transition group helps us for multiple classes according to the state of the animation. Using keys is always a good idea, if you don't do it, React gives a warning in the console (in development mode). Let's look at a simple example where there is a function for adding two numbers and it gets enhanced with a function that first logs all the parameters and then executes the original one: const add = (x, y) => x + y [45] Clean Up Your Code const log = func => (...args) => { console.log(...args) return func(...args) } const logAdd = log(add) This concept is pretty important to understand because in the React world, a common pattern is to use Higher-order components (HoC), treating our components as functions, and enhancing them with common behaviors. React motion As soon as the complexity of the animations grows, or when we need animations that depend on other animations, or which is more advanced, when we need to apply some physics-based behavior to our components, we realize that the transition group is not helping us enough, so we may consider using a third-party library. The prop containing the response from the API is called data, which is generic and not useful, but we can easily change it, thanks to ES2015, and give it a more meaningful name. The result is the same: a comparison between the two values. The component is implemented using a class: class Counter extends React.Component It has a constructor where we initialize the state using the count prop and we bind the event handler: constructor(props) { super(props) this.state = { count: props.count, } this.handleClick = this.handleClick.bind(this) } The implementation of the click handler is pretty straightforward: we just add 1 to the current count value and store the resulting value inside the state: handleClick() { this.setState({ count: this.state.count + 1, }) } Finally, in the render method, we describe the output, which is composed by the current value of the count, and the button to increment it: render() { return ({this.state.count} +) } Now let's render this component, passing 1 as the count prop. [271] Anti-Patterns to Be Avoided It works as expected: each click on the + button increments the current value. Let's see what the version of Recompose looks like: const withGists = ... This is pretty bad in big codebases where it's hard to find what's going on and the reason why things happen. To view the previously submitted errata, go to ... and enter the name of the book in the search field. In this chapter, you will see how to set up a server-rendered application, and by the end of the relevant sections, you will be able to build a Universal application and understand the pros and the cons of this technique. It is now time to enhance our component: const GistWithStar = connectWithStar(Gist) Also, it is time to use the star function inside it to fire the request: const Gist = ({ description, star }) => ({description} +) Gist.propTypes = { description: React.PropTypes.string, star: React.PropTypes.func, } As you can see, it is very simple; the component receives the star function, which star is the name of the request that we defined in the connect function. This is a very powerful feature and it works in a fascinating way. Or a parent has to be updated when its children's state changes? The problem we will try to solve is a common task in web development: showing a map with a marker. [227] 10 About Testing and Debugging React, thanks to components, makes it easy to test our applications. Possibly available re-render! [52] Create Truly Reusable Components With React Components, we can declare the props inside the object we have passed as a parameter to the function, and we use the getDefaultProps function to return the default values: const Button = React.createClass({ propTypes: { text: React.PropTypes.string, }, getDefaultProps() { return { text: 'Click me!', } }, render() { return {this.props.text} } }) As you can see, we pass the propTypes attribute to list all the props that we can pass to the component. Whenever we use the id attributes, we are describing to React the behavior we want to achieve, but the library does not add the actual event handler to the underlying DOM nodes. We will also define the configuration for the Babel loader, which we will use for both the client and the server: const nodeExternals = require('webpack-node-externals') const loaders = [{ test: /\.js$/, exclude: /node_modules/, loader: 'babel', query: { presets: ['es2015', 'react'], }, }] In Chapter 7, Anti-Patterns to Be Avoided, we looked at how we had to provide a configuration object for the Babel loader, [185] Server-Side Rendering for Fun and Profit loader: 'babel', query: { presets: ['es2015', 'react'], }, }] In Chapter 7, Anti-Patterns to Be Avoided, we looked at how we had to provide a configuration object for the Babel loader. For example, a function that returns an element, a function that touches external entities, such as the DOM, is considered impure. It could be great to expose the buttons after the user has finished writing in the field and the buttons are visible. The question is: if we move the buttons away from the component, how do we know when they get clicked on so that we can update the state? If you've ever tried to write end-to-end tests for the Web, you'll know how complex it is to get consistent results and how the results are often affected by false negatives due to different factors, such as the network. In fact, JSX always gets replaced by functions, and functions written on a new line can give you an unexpected result because of automatic semicolon insertion. Jest looks into the source folder to find files that end with .spec, .test, or the ones inside a __tests__ folder, but you can change this configuration as you wish to fit the needs of your project. You may have noticed that the TestUtils, even if they provide a useful tool to help testing components, are verbose and sometimes it is not easy to find the right approach to get the reference to the elements and their properties. However, React does not force you to use a particular test framework and you can use your favorite one without any problems. The process is called transpilation, because it compiles the source into a new source rather than into an executable. For example, we cannot use it easily in our Style Guide, passing a fake currency as a prop. The question is, What should we do if we need more features? As soon as the app grows a bit more, we may need a router to handle different pages and views, and we can include that as well. It is very hard to create an abstraction, and it makes sense to know where and how you can avoid duplication and when it is not needed. The CSS loader lets you import the CSS files into your JavaScript modules, and when the modules are activated, all the class names are locally scoped to the module they are imported into. The function returns a class component with its own state and event handler: class extends React.Component It has a constructor to set the initial empty state. The style loader takes the result of the CSS Modules transformation and it injects the styles inside the head section of the page. It is important to notice that the state of the Form is composed by all the fields and their values. To put an element at the first item of an array we can use JavaScript's unshift and apply it to a copy of the array to make sure we do not mutate the current state. [118] Proper Data Fetching Our callback function returns an object where the keys are the identifiers of the request, and the values are the URLs. The URL is not limited to being a string; we'll see later how we can define more complex requests. Let's see how it works: The method returns a Operation, which is the result of the operation. The first method will receive two parameters: "search" and Payload, which shows { "Infinite": "Cancel", "LT" }. Sharing JavaScript code is pretty straightforward, while attaching the CSS is not as easy as you may think. The ability to use the same language on the server represents a significant win regarding maintainability and knowledge sharing across the company. Reusable components can be shared across multiple domains of your application to avoid duplication. It extends the PureComponent as well so that it never gets re-rendered from the parent like the controlled vs. uncontrolled form with fields: class Controlled extends React.Component We define a constructor in which we initialize the state and bind the handler for the controlled input: constructor(props) { super(props) this.state = { value: '', } this.handleChange = this.handleChange.bind(this) } The implementation of the change handler is very similar to any other controlled input we have seen until now: handleChange({ target }) { this.setState({ value: target.value, }) } It receives the target element, which is our input, and saves its value into the state. To attach an event listener to a node and get the object when the event is fired, we can use a simple convention which recalls the way events are attached to the DOM nodes. You might have noticed the key attribute of . These immutable objects are much simpler compared to the components and their instances, and contain only the information that is strictly needed to represent the interface. Now comes the hardest part and the most powerful one: get.mock('.get.json', () => { ...). It is used to create an external component to fetch the data. One way of using a mixin is to write it once and share it across the different components. When we apply the idea of HoFs to components, we call it Higher-order Components (HoCs) for brevity. Inside our index.js where we defined the button component, we can add the following line: import styles from './index.css' The result of this import statement is a styles object, where all the classes defined in the index.css. Our list can show items with title and text, so we need the add attribute to our only collection: const posts = [{ id: 1, title: 'Create Readable Code', excerpt: 'Your code should be readable' } ...] 'Mutation', excerpt: 'How do I share the code between components'. In this way, we are going to show the following snippet right after the one we previously created: .add('with text field', () => ( )) If we now go back to the browser, the page is automatically refreshed, and we see two stories in the left sidebar. Let's start with the UI and then add the real API call, thanks to react-refetch. FAIL ./button.spec.js In the onClick callback expect(jest.fn).toBeCalled() Expected mock function to have been called. We could spread the properties to the elements to avoid writing every single one manually, as follows: This works only well and it gets transpiled into the following code for Babel: React.createElement(Component, props); However, when we spread props into a DOM element, we run the risk of adding unknown HTML attributes, which is a bad practice. We want to add the parameter to the element where we pass the render function and an hint to tell Jest that the component we're testing is good to go. Working closely with new people who have different backgrounds and skill sets is one of the best ways to keep your mind open and improve yourselves continuously. Code Button = () => (Hello World!) = React.createElement(hello) You are now able to create a server-side rendered application with React, and it should be clear why it can be useful for you. In general, we have a setup React with hot loading: import { expect } from chai It is time to start the thing: a new React.Component, now we render TestUtils.renderIntoDocument(). If we run the test with npm test in the terminal, we can see that Button is not defined. The first thing we have to do is to install a couple of the packages. It uses the DOM to keep the user updated on the state of the form, which is something that we do not recommend. In this case, the component is null and one solution is to wrap our component inside a follows: const component = ReactDOM.render(componentElement, ...) Optimization One thing we should keep in mind when we are dealing with components. If, in the previous example, the Gist component needed to change. Even if we had the functions to pass the right implementation of a stateless functional component without a state, at the time of writing, they perform a little bit less well. The second is chai-spies, which provides the functionality that we can use to inspect the onClick callback and check if it has been called. What if, for example, we have to create a decorator that deals with a data fetched to do it? This can change the HoC to accept two kinds of props: a string, as we have seen before and a function, which receives the component's props and returns a URL that depends on the received parameters. We still want our application to be fast and we also want to be able to create highly dynamic applications. But we only need to create a complex selector when clicking on the button, and this is done in the following way: => { if this.input.focus(); this.input.focus() (input = el)} /> Focus } } [59] Create Truly Reusable Components No reference to component Accessibility is an indicator of the stateless functional components, whenever we render them using the ReactTestUtils (we will cover tests extensively in Chapter 10, About Testing and Debugging), or we do not receive back any reference to the component. We also have to change the render method and remove all the UI part of it, as follows: render() { return ( ) } As you can see in the snippet, instead of creating the HTML elements inside the render method itself, we just use the Presentational one (which we will create next), and we pass the state to it: the render method of the component is pretty similar to the event type: render() { return ({this.props.gists.map(gist => ({gist.description}))}) } [196] Server-Side Rendering for Fun and Profit However, if it uses this.props.gists, which we can also use as an inside a class component. Instead of creating a new component, the function renders a specific result. This is a common practice to write a single end-to-end component, which can trigger different actions according to the event type. Then, the function loops through all the elements of the input array and invokes the functional component that takes the input as the array. The first basic test that we can write is checking if a children is rendered as a child of the button element. The next step is to store the user and make it available when the user clicks the submit button. Firing a function inside the render multiple times, even if it is not optimal, it is not a problem by itself. The state has the latitude and longitude properties, which are null by default, and they get updated with the real position of the user when the browser has fired the callback. This is one of the most important features of React. We have to use the PropTypes to validate the props that we receive, and that helps in debugging. The correct function of each component accepts a function as the second argument, we need to create it as soon as the getJSON function returns the data. This signature is the reason why the component does not receive the server bundle. Adding this component we get a list that we can use to correctly show every single part of the application. Fully searchable across every book published by Packt Copy and paste, print, and bookmark content On demand and accessible via a web browser Customer Feedback Thank you for purchasing this Packt book. [12] Everything You Should Know About React Some of the same problem applies to styles: they are defined in a different file, so they have to be referenced in the templates and the CSS selectors follow the structure of the markup, so it is almost impossible to change one without breaking the other, which is the definition of coupling. Let's look at how it is possible to create the same button we saw in the previous section, and check if all the CSS features we are interested in (for example, pseudo classes and Media queries) work with Styled Components. The application will consist of two parts: The server-side, where we will use Express to create a basic web server and serve an HTML page with the server-side rendered React application The client-side, where we will render the application using react-dom. The first thing to do is to change app.js to accept a list of gists as a prop, and loop through it to show the gists description: const App = ({ gists }) => ({gists.map(gist => ({gist.description}))}) App.propTypes = { gists: React.PropTypes.array, } Applying the concept we have learned in the previous chapter, we define a stateless functional component, which receives gists as a prop and loops through the elements to render a list of items. First of all, we have to create a class: class TodoTextInput extends Component The propTypes are defined as a static prop of the class: static propTypes = { onSave: PropTypes.func.isRequired, text: PropTypes.string, placeholder: PropTypes.string, editing: PropTypes.bool, newTodo: PropTypes.bool } The state is initialized in the constructor, as it is done in the previous sections. We need to add a new line. To make this possible, we need a library plugin called transform-class-properties: npm install --save-dev babel-plugin-transform-class-properties Then we have to add this plugin in the list of plugins in our .babelrc: "plugins": ["transform-class-properties"] Now, we have to create a test, and the first thing we have to do is make sure that the initial value of the input is set properly. This makes the syntax to simulate an event is slightly different from the TestUtils but it is still intuitive: button.simulate('click') ShallowWrapper has a simulate function that we can use to trigger events for our components. [231] About Testing and Debugging We have to pass the handler function as a callback, passing the name of the events and some parameters as a second argument. We simulate the change event passing a text string and then we check the value of the output when the input event is triggered. It is possible to include the relevant code in our tests on CSS. This will be used to inject the file for each test. Once we are declaring too many props for a single component, and they are not related to each other, it may be a signal that it is the time to introduce multiple vertical components, each one with fewer props and responsibilities. If you have ever set the element with the DevTools open, you can see that the style changes and the color gets added to it dynamically: Click me! The way Radium works is by adding an event handler for each one of the events that can trigger the behavior of pseudo-classes, and listening to them. The method is called getInitialState, which is a component method and it is used to set the initial state of the component. We do not need to update the state because we have a component that needs to update the state every time it is executed in this case. The DOM operations that have been applied in the browser. In his talk, he went through all the problems related to CSS at scale when he was working at Facebook. We will finally see a real-world example using Components: They are more concerned about the behavior. Their response for Presentational components, on the other hand: They are more concerned with the visual representation They render the HTML markup (or other components) They receive data in the form of props from the parents. They may make API calls and manipulate data They define event handlers They are written as classes Presentational components: They are often written as stateless functional components [90] Compose All the Things Mixins are great to achieve reusability, but what if different components in different domains share the same behavior? A new type of component has been introduced in React which lets us declare a component in the form of a function. Let's look at an example: title: { margin: 0 } } Then, Hello Now! Look here is great, because you will see the styling option for each of the states. Dependencies. The form has some CSS Modules features and reusable components are built is what we will see in this chapter. What we write inside the JSX file becomes the JavaScript, and we can use it to add the fields: this.props.json. This.setState({ title: '' }) } If the URL is a function, we fire it by passing the props as parameters, while if it is a string we use it directly. This is very useful information that is sometimes not easy to find. React elements React elements are a form of abstraction that you are familiar with components and instances, but there is another object you should know if you want to use React effectively: the Element. First, you may use function that returns data that receive a value of the props. Keys and the stateless components uses the function as a child pattern (see Chapter 4, Compose All the Things), so it is much easier to write than the best practices and define a clear description of the components that need to be rendered. [50] Create Truly Reusable Components Fortunately, we can use propTypes to declare the properties that the component needs: Geolocation.propTypes = { latitude: React.PropTypes.number, longitude: React.PropTypes.number, } If you run the component in the browser, you can see something like this: Following the Container and Presentational pattern, we created a dumb reusable component that we can put in our Style Guide, passing fake coordinates to it. What we actually have to do is to create a folder called stories as long as you define the config. Now, if we are in the topic of the logic inside the item that has been clicked. As long as you have some ... that's why we use the form, and the property the style is stored. For example, the reason why we use ESLint, without any rules it does not do anything, but we can enable the recommended configuration to activate a basic set of rules. One example is a rule to enforce the correct prop types, or, if we run npm test now, we can get the following output: ✓ renders with text (5ms) ✓ fires onSave on blur if not new (1ms) ✓ fires onSave on enter (1ms) ✓ does not fire onSave on key down (1ms) ✓ clears the value after save if new (3ms) ✓ updates the text on change (1ms) ✓ fires onSave on blur if not new (2ms) Test Suites: 1 passed, 1 total Tests: 8 passed, 8 total Snapshots: 0 total Time: 2.271s Ran all test suites. Then we search for the List component inside the mounted wrapper and check that its data property is the same as the data returned from the fetching. Finally, we write the expectation: expect(onClick).toBeCalled() Here, we have simply said that we expect the mock function to be called. In fact, the SSR is only half of the job that React does to improve the rendering of our application. We can use the class attribute on the element (as it is usually a string), and we can use it to make a new instance. It is important to think carefully about when it makes sense to use it and when it is better to rethink your structure. Classes are defined in CSS files, but they are composed in the views, and every time you have to modify the style of an element, you need to change the class. The props the UI should display, so we need to know how to use the render props in sequence of how it works. It's written and published in Node.js environment. Alternatively, instead of setting the new value using the constant and the CSS class that is composed in the component to call a function, something like getInnerWidth in our example, to get the actual value. The following screenshot shows the source of the page: [190] Server-Side Rendering for Fun and Profit A data fetching example The example in the previous section should explain clearly how to set up a Universal application in React. This is something that makes JavaScript and the browser a completely different environment compared to any other language or platform. Consider that we have a Text property representing the text of the button: const Button = ({ text }) => ({text}) Button.propTypes = { text: React.PropTypes.string, } It can be used in the following way: And it renders the following code: Click me! Now, suppose we want to use the same button with the same class name in multiple parts of our application, and we also want to be able to apply a single string. An excellent example of a highly customizable library that uses Storybook to show all the variations is react-dates from Airbnb. [286] Next Steps To show that your component is highly customizable you can add one or more examples to the repository to make it easy for everyone to understand how it works and which props it accepts. However, developers make mistakes, and this chapter is all about the common patterns we should avoid when using React. Last but not least, an example is a sort of template, a piece of content that you can put in your code to explain a function. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "Let's begin updating the tests, starting with the renders with text ones." [3] Preface Warnings or important notes appear in a box like this. Also, we must not forget to add the data on componentDidMount and apply some lazy loading patterns instead. The most useful one is printWasted which prints out the times that the components have spent executing their render methods and returning the previous execution. At the end of the chapter, you'll be able to create a test environment from scratch and write tests for your application's components. The TodoMVC example is a standard way of creating a real-world application with the different frameworks in order to compare their features and make it easier for developers to choose. As you probably know already, the new ES2015 arrow function automatically binds the current this to the body of the function. The first one is the global keyword. We should be able to move components across the application, and they should provide the same result regarding both logic and UI, no matter where they get rendered. Testing components should now be familiar to you, and you should have all the information to start writing tests for your applications. Unfortunately, if we run the component in the browser, we notice some new logs in DevTools: first of all the whyDidYouUpdate library is telling us that there is a possibly avoidable re-render for the onClick function and this is the same: item.props.onClick Value is a function. They may decide to reject it or to ask you to make some changes. Then there is the controlled input field and we set its current value as a controlled of the component. Given that, every developer that comes across our codebase knows how to use it in the right way. We saw how React tried to solve the problem of sharing functionalities between components with mixins. [67] Create Truly Reusable Components It was quite a journey, but hopefully, you now understand that for the components to be reusable, it is crucial to make sure that our components and their interactions are well-defined and straightforward to use. As you can see, we are making a convention: render() { return ( ) } [155] Make Your Components Look Beautiful Rendering the preceding component, we can see an input field that, if clicked, changes its font size according to its value. As developers, is that something we must not ignore? However, is not easy to figure out the best strategy to test, for example, Higher-Order Components. [204] Improve the Performance of Your Applications However, if we open the DevTools and look at the console, we can see that the Perf add-on has logged something different from the previous execution. The two tests start getting more complex, and you will learn how to go through some of them to see how they can help us to follow the best practices. For example, in the Redux realworld example, a sub-render method is used to render the load more button. [203] Improve the Performance of Your Applications Everything works as expected but if we look inside the component we have defined a state object on the mount data with two buttons, they are a very important tool to compare the previous and the next props and the state. Using the first seems to be a good way to go, like something as simple as a string to the value of the element that we want to maintain. As we have seen, so it is crucial to make sure that we only make it work. Some of these rules are quite strict and it is worth to mention. It is a component that will be used to render the list. Unfortunately, all these features do not come for free and one important aspect is also the performance. onSave property in the component: test('sets the text prop as value', () => { const text = 'text' const wrapper = shallow() expect(wrapper.prop('value')).toBe(text) }) The code is pretty straightforward: We create a text variable, which we then pass to the component as a text prop when we shallow-render it. Our goal is basically to use a single change handler so that we can add an arbitrary number of fields without creating new listeners. We have learned how Snapshot Testing can make it even easier to test, and we have seen what is the best way to test a component. The mixin can be defined as an object literal that has the same functions and attributes of a component: const WindowResize = { ... } To communicate with the component, mixins usually use the state. Suppose we want to show a logout button only if the user is currently logged into our application. The reason it must be used is that we have to use features of the language that have not yet been added in the browser, our target environment. The other questions help you to explain the problem in detail and describe the current and the expected behaviors. First of all, if you want to publish your React component you have to write a comprehensive set of tests. This is a very common pattern because we may think that providing any key is always the best solution but it is not like that. What I mean by common is that by using JavaScript as a template language is a very good idea. [83] Compose All the Things However, if we realize that our Button could be just a wrapper, and we want to be able to render any element inside it, we can use the children property. We can see how data flows work in React and why the approach it enforces can make our applications simpler and cleaner. In this chapter, we will go through the basic concepts of performance of React and we will learn how to use some APIs to help the library find the optimal path to update the DOM without degrading the user experience. In production, we do not need class names like this, and we are more interested in performance, so we may want shorter class names and hashes. If we render the button inside the browser, we will not see anything in particular at the moment, because we are not applying any styles to it. On the other hand, for the class component, we can use the Class.propTypes and the Class.defaultProps. In our case, we have a button example that shows how to reproduce and solve it. This makes it easier, and also we can change the implementation of the HoC without modifying the subcomponent. Then we'll render the component and finally we'll simulate the key down event passing a specific key code. For this reason, we could create two props: one called titleKey where we specify the name of the attribute to be displayed and one called textKey for which we can specify the optional field. Now, we implement the new handlers: handleChangeFirstName({ target }) { this.setState({ firstName: target.value, }) } handleChangeLastName({ target }) { this.setState({ lastName: target.value, }) } [129] Write Code for the Browser We also have to change the submit handler a little bit so that it displays first and last name when it gets clicked: handleSubmit(e) { e.preventDefault() console.log(`${this.state.firstName} ${this.state.lastName}`) } Finally, we describe our elements structure inside the render method: render() { return (Submit) } We are ready to go: if we run the preceding component in the browser we will see two fields, and if we type Dan into the first one and Abramov into the second one we will see the full name displayed in the browser console when the form is submitted. In our Counter component we read the value of the count prop and we stored it into the state. Creating functions inside the render method Now, let's keep on adding features to the List as we would do in a real-world application and see whether, at some point, we manage to break the benefits given by the PureComponent. Why did you update? Unfortunately, there is no way for us to know the list of methods that has to be implemented. [208] Improve the Performance of Your Applications There are some tricky situations where extending the PureComponent does not give the expected results; usually, the cause is that our props or state attributes are changing even if we think they are not. Then we can see that the Motion component has two attributes: the first one is defaultStyle, which represents the initial style. The first and most important thing is to create a small reproducible example of the problem. This could sound counter-intuitive in the beginning because we might think that React is smart enough to use the gists rendered within the server-side string on the client. The second prop is the children element, which we want to make sure contains the right value. Only in that very moment, should we introduce a package manager. Usually, we read the value in the client using the JavaScript APIs, but the optimal solution would be to share constants and avoid doing expensive calculations at runtime. Then, we went through some of the most interesting composition patterns in React. Some of them will require a bit of it, but we should try to keep them as simple and dumb as possible so that we can easily spot and fix errors. There's a package that helps with that, and we can simply apply it to the external entry in our Webpack configuration to exclude all the modules: npm install --save-dev webpack-node-externals Great, it is now time to create an entry in the npm scripts section of the package.json so that we can easily run the build command from the terminal: "scripts": { "build": "webpack" }, We now have to create the configuration file, called webpack.config.js, to tell Webpack how we want our files to be bundled. [75] Create Truly Reusable Components The benefits of using a reusable component over a component which is coupled with the data it handles are many. To use the fetch API on the server-side, we have to install a library called isomorphicfetch, which implements the fetch standards. Then, we pass the function to the configuration method, and we are ready to go. You could receive Pull Requests from developers, which could be long and complex, and they need to be reviewed. We can concatenate multiple HoCs to keep our components as simple as possible. The reason it does that is because React suggests you organize your applications by composing small bricks called components. This is because, when a component has a single child, React optimizes the creation of the elements and avoids allocating an array for performance reasons. As you learned in Chapter 3, Create Truly Reusable Components, using props is always a good solution to enforce reusability. This technique is called event delegation and is used for memory and speed optimization. This means that if we want to set an attribute to false, we have to declare it explicitly as false: React.createElement("button", { disabled: true }); The following is another example: React.createElement("button", { disabled: false }); [27] Clean Up Your Code This can be confusing in the beginning because we may think that omitting an attribute would mean false, but it is not like that. If nothing changes, no changes will be made in the DOM, which is great. [100] Compose All the Things Context HoCs come in very handy when we have to deal with context. The keywords static and async applied to a class method mean that the function can be accessed outside the instance of the class and that the function yields the execution of the wait instructions inside its body. Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Inside the loop, there is some conditional logic to check if the #first and the #link properties exist, and depending on their values, a different piece of HTML is rendered. Passing single primitive props helps us to find whether a component surface is too wide and whether it should be split into smaller surfaces. In the production version of React, the propTypes validation is disabled for performance reasons. For example, we may want to enforce one of the best practices we have seen in the previous section and keep our templates consistent across developers and teams. The fact that React offers a declarative approach makes it easy to use, and consequently, the resulting code is simple, which often leads to fewer bugs and more maintainability. We are using the spread attribute operator, which we saw in Chapter 2, Clean Up Your Code; it is a convenient way to pass the attributes of the state, avoiding writing prop by prop manually. We will see HoCs and other patterns in Chapter 4, Compose All the Things. We surely want our components to be simple and we should avoid doing expensive operations inside the renderer. The reason why HoCs usually spread the props they receive on the component is because they tend to be transparent and only add the new behavior. Suppose, for example, you are returning JSX from your render method, which is how you create UIs in React. We use the partial application technique to specialize the function and reuse it: const connectWithGists = connect(({ username }) => ({ gists: ` {username}/gists`, })) The preceding code needs a bit of explanation. Summary In this chapter, we learned a great deal about how JSX works and how to use it in the right way in our components. One of the core members will get in touch with you. What we are doing in the callback is storing the reference of the element to be able to use it in the future (for example, when the handleClick method is fired). Again, one of the best ways to learn new concepts is by seeing practical examples, so let's delve into some code. The biggest argument against Atomic CSS is usually that you are moving the styling logic from the CSS to the markup, which is wrong. There are different methods which can be used to display the information we've previously gathered in the browser console. Let's start by creating a simple button with a basic styling: const Button = styled.button backgroundColor: #ff0000; width: 320px; padding: 20px; borderRadius: 5px; [175] Make Your Components Look Beautiful border: none; outline: none; ` This kind-of-weird syntax returns a proper React component called Button, which renders a button element and applies to it all the styles defined in the template. The first button is transpiled into the following: React.createElement('button') The second one is transpiled into the following: React.createElement(Button) The difference here is that in the first call we are passing the type of the DOM element as a string, while in the second call we are passing the component itself, which means that it should exist in the scope to work. To do that, we have to create a special function called mock (or spy, depending on the framework) that behaves like a real function but has some special properties. This idea is so widespread and shared among people that it has been clearly defined, and has been given the name JavaScript Fatigue. Reader feedback Feedback from our readers is always welcome. It is incredibly useful in development mode. However, the information contained in this book is sold without warranty, either express or implied. Here is where the compose function of recompose comes handy. Reasons to implement Server-Side Rendering SSR is a great feature, but we should not jump into it just for the sake of it: we should have a real and solid reason to start using it. A nice feature of mixins is that they can merge lifecycle methods and the initial state. The Presentational component is where the UI is defined, and it receives data in the form of props from the container. Before starting, it is important to have a clear idea about what we want to test and why. Get the most in-demand software skills with Mapt. Stories usually go into a folder called stories that you can create inside your components folder or wherever it fits better in your folder structure. Last but not least, we have to export the configurations as an array: module.exports = [client, server] The configuration is done. If we want to ask for information from our users within the browser, forms are the most common solution. For example, we can enable a rule to enforce the sorting of the prop types into alphabetical order, a rule to give us an error when we are using a prop that has not been declared, a rule to force us to prefer stateless functional components over classes (we will see the difference in detail in Chapter 3, Create Truly Reusable Components), and so on. Write something like the following: ESLint complains, as follows: Expected indentation of 2 space characters but found 0 (react/jsx-indent) A similar rule regards the way we indent our attributes when we write them on a new line. For example, we can add CSS rules like the following: .button:hover { color: #fff; } .button:active { position: relative; top: 2px; } @media (max-width: 480px) { [168] Make Your Components Look Beautiful .button { width: 160px } } This will be transformed into the following code and injected into the document: ._2wpxM3yizfwbWee6k0UlD4:hover { color: #fff; } ._2wpxM3yizfwbWee6k0UlD4:active { position: relative; top: 2px; } @media (max-width: 480px) { ._2wpxM3yizfwbWee6k0UlD4 { width: 160px } } The class names get created and they get replaced everywhere the button is used, making it reliable and local, as expected. This section assumes that all the concepts regarding JavaScript build tools, such as Webpack and its loaders, are clear, and it requires a little bit of knowledge of Node.js. The static string is pretty easy to test: test('uses the string url', () => { const url = ' const withGists = withData(url) const ListWithGists = withGists(List) mount() expect(getJSON).toHaveBeenCalledWith(url) }) We define a URL and we use the partial application to generate a new function; we then use it to enhance the dummy List. The TestUtils library has functions to shallow render components or render them into a detached DOM outside the browser. We can start with Webpack: npm install --save-dev webpack After it is done, it is time to install the Babel loader and the presets that we need in order to write an ES2015 application using React and JSX: npm install --save-dev babel-loader babel-core babel-preset-es2015 babel-preset-react We also have to install a dependency, which we will need to create the server bundle. Painless JavaScript testing with Jest The most important way to learn how to test React components in the right way is by writing some code, and that is what we are going to do in this section. However using the hooks and polluting the codebase for tracking the component's performance can be overkill. Components can be composed to form the final UI, which is a property of FP. First of all, we have to activate the rule: "rules": { "react/jsx-indent": [2, 2] } The first 2 means that we want ESLint to raise an error in case the rule is not followed within the code, and the second 2 means that we want every JSX element to be indented with two spaces. Chapter 6, Write Code for the Browser, states that our applications live in the browser and we should know how to use it properly. The first version of a package is usually the 0.1.0. To publish an npm package you must have an npm account, which you can easily create by running the following command in the console: npm adduser $username Where $username is the name of your choice. Wouldn't it be nice to remove the duplication and move the selectors to one single place so that it is easier to change them if the form changes? HoCs are functions that take a component and return an enhanced version of it. In our example, we began from a single component, and we realized that we could separate the API call from the markup. We appreciate your help in protecting our authors and our ability to bring you valuable content. Living style guides Creating reusable components with a clear API is great to avoid duplicating code across the application, but that is not the only reason why you should focus on reusability. We can now use the HoC in the following way: const withGists = withData(props => ` {props.username}/gists`) Where the username for loading the gists can be set in the props of the component: React-refetch Now, our HoC works as expected and we can reuse it across the code base without any problems. In fact, there are many different paths you can take to provide it: from adding a CSS file to the package to using inline styles. A React component to show a map on a page would look like this instead: In declarative programming, developers only describe what they want to achieve and there's no need to list all the steps to make it work. Building software together is great to improve both your skills and your packages and that is why I strongly believe in open source. You can just download the distribution bundle and host it yourself (or use unpkg.com), and you are ready to get started with React and its features in a few minutes. Variables are wrapped into curly braces. Another feature of CSS it is not possible to emulate using inline styles is Animations. This has been the case for the last few years, and it is still the way to go for a huge number of applications. - Snapshot + Received @@ -1,8 +1,8 @@ { if (!this.props.newTodo) { this.props.onSave(e.target.value) } } When the to-do is not new, the onSave prop function is fired with the current value of the field; when the to-do is new nothing happens. Running the preceding code in a browser will show the form with the field and the button, and clicking on the button will focus the input field, as expected. Let's see an example of that: const Button = React.createClass({ getInitialState() { return { text: 'Click me!', } }, render() { return {this.state.text} }, }) The getInitialState method expects an object with the default values for each one of the state properties. We have to create a :hover property inside our style object, and Radium will do the rest: const styles = { backgroundColor: '#ff0000', width: 320, padding: 20, borderRadius: 5, border: 'none', outline: 'none', ':hover': { color: '#fff', }, } [159] Make Your Components Look Beautiful If you apply this style object to your button and render it on the screen, you can see that passing the mouse over the button results in a button with white text, as opposed to the default black one. First of all, if we mutate the state without using setState, two bad things can happen: The state changes without making the component re-render Whenever setState gets called in future, the mutated state gets applied If we go back to the counter example and change the click handler to: handleClick() { this.state.count++ } [273] Anti-Patterns to Be Avoided We can see how clicking + does not affect the rendered value in the browser but, if we look into the component using the React Developer Tools, the value of the state is correctly updated. The most difficult thing that happens working with inline styles is debugging. To do that, we will go through all the reasons why regular CSS may not be the best approach for styling components, and we will check out various alternative solutions. Inside the templates, it is also possible to use the props that the component received and change the style accordingly. As you can see, there is a props attribute which has our text property of type string defined inside it. As you can see, they use Storybook to show the different options of the component: [287] Next Steps Last but not least, you not only want to share your code, but you may also want to distribute your package. A common way to create SVGs in a web app with React is to wrap our vectors into a React component and use the props to define their dynamic values. Babel plugins There are also a couple of interesting Babel plugins that we can install and use to improve the performance of our React applications. You learned how it is possible to load the data on the server and dehydrate it in the HTML template to make it available for the client-side application when it boots on the browser. As you can see, we are passing to the children titles and text props which represent the values of the main attribute and the optional one, respectively. Check them out! Errata Although we have taken every care to ensure the accuracy of our content, mistakes do happen. You may also have noticed that we are setting the presets in there as well. For example, we could run the linting at test time in a such way that if the code does not pass the linting rules, the whole test step fails. Not just that, there are conferences and meetups where you can talk to people in real life and build new relationships, blog posts that you can read to improve your skills and learn more, and many other ways to become a better developer. We have now mastered the techniques that let us abstract the logic away from components so that we can make them as dumb as possible, improving their testability. Finally, we had a quick look at Styled Components, which is a very promising library and is meant to completely change the way we approach the styling of components. When a component passes some props to another component, it is called the Owner, irrespective of the parent-child relation between them. A real world example of components documented using docgen is the great Material UI library, where all the docs are automatically generated from the source code. [251] About Testing and Debugging In the last test we check that the blur event fires the callback only when the To-do item is not new: test('fires onSave on blur if not new', () => { const onSave = jest.fn() const value = 'value' const wrapper = shallow() wrapper.simulate('blur', { target: { value } }) expect(onSave).toHaveBeenCalledWith(value) }) We set up a mock, create the expected value, simulate the blur event using the target, and then we check that the onSave callback has been called with the given value. The version number is composed of three numbers separated by a dot and they all have a meaning. The library is conceived and maintained by two popular developers in the JavaScript community: Glenn Maddern and Max Stoiberg. Let's now create a client.js, which is actually responsible for rendering the App inside the DOM: import React from 'react' import ReactDOM from 'react-dom' import App from './app' ReactDOM.render(, document.getElementById('app')) Again, this should sound familiar, since we import React, ReactDOM, and the App we created earlier, and we use ReactDOM to render it in a DOM element with the ID app. The following screenshot shows the default page of an application created with create-react-app: We will use this tool throughout the book to run the examples for each chapter which are also available on GitHub at the following address: [17] Everything You Should Know About React Summary In this first chapter, we have learned some basic concepts that are very important for following the rest of the book, and which are crucial to working with React daily. There are some important differences to note: the former example is less elegant and it requires more effort to be understood. [73] Create Truly Reusable Components Duplicating the code is usually not the best solution so let's see how React can help us to keep our code Don't Repeat Yourself (DRY). Avoidable re-render! before ["open", "close"] after ["open", "close"] Item.props.statuses Value did not change. You may consider that you can execute the fetch again on the client. In fact, we will use the value attributes of the input fields to set their initial values as well as the updated one: render() { return (Submit [133] Write Code for the Browser) } The first time the form is rendered, React uses the initial values from the state as the value of the input fields. It is important to know that, when the component gets unmounted, the same callback is called with a null parameter, to free the memory. We start by just logging it, because it is important to proceed with small steps, but we will store the value into the state soon. The basics of functional programming Apart from following the best practices when we write JSX and use a linter to enforce consistency and find errors earlier, there is one more thing we can do to clean up our code: follow a Functional Programming (FP) style. If a piece of code is well-tested (and tests are well-written) developers can be sure that it works and is ready to ship. Now run the following again: npm run build && npm start If we point the browser window to , the error is gone, and if we inspect the App component using React DevTools, we can see how the client-side App component receives the collection of gists. It is very easy to think that all these tools are required to start using React, but this is far from the truth. [25] Clean Up Your Code The style attribute does not accept a CSS string as the HTML parallel does, but it expects a JS object where the style names are camelCased: Root One important difference with HTML worth mentioning is that since JSX elements get translated into JavaScript functions and you cannot return two functions in JavaScript, whenever you have multiple elements at the same level, you are forced to wrap them into a parent. Hover the mouse pointer on the SUPPORT tab at the top. One of the motivations why you may want to contributes to improve your skills as a developer. GitHub has a lot of ready-made templates to choose from. [60] Create Truly Reusable Components The state We have seen how to create a component with the factory, extending the React class or using stateless functional components. Style fallbacks should be avoided, but it is always good to have the ability to use them if needed. It is now possible to reuse this component as many times as we need in our application and every developer can easily understand how to implement it thanks to the prop types. Our list item has a required title attribute and an optional text, so we can create at least two stories to represent those states. We used to use icon fonts to create icons, but they have well-known problems, with the first being that they are not accessible. For example, in the previous section we created a generic list with title and text, and since it is decoupled from the data it shows, we can use it many times within the app just by passing the right props. [257] About Testing and Debugging Testing Higher-Order Components As we have seen in previous chapters, we can use Higher-Order Components to share functionalities between different components across the application. Let's now see how this works differently with classes, and what we can do to create the same behavior. Chapter 2, Clean Up Your Code, teaches that one of the most important aspects of writing maintainable code is to keep it clean and follow a coding style guide. }) You may have spotted a pattern in the way HoCs are named. The only possible way to achieve this result would be by monkey-patching the render method and wrapping it into a try...catch block. The second event handler is the change handler: handleChange = e => { this.setState({ text: e.target.value }) } Apart from its being defined as a class property you should be able to recognize the method that keeps the state updated for a controlled input element. As mentioned previously, in general, we should try to avoid using refs because they force the code to be more imperative, and harder to read and maintain. We also bind the event handlers we need to make the form work. JSX, in fact, handles the spaces between text and elements differently from HTML, in a way that's counter-intuitive. This can be counter-intuitive in the beginning, especially if you are not used to working with immutable data structures. If you purchased this book elsewhere, you can visit om/supportand register to have the files e-mailed directly to you. The first one is the submit handler: handleSubmit = e => { const text = e.target.value.trim() if (e.which === 13) { this.props.onSave(text) if (this.props.newTodo) { this.setState({ text: '' }) } } } [246] About Testing and Debugging The function receives the event; it trims the value of the target element, then it checks if the key that fired the event is the Enter key (13) and, if it is, it passes the trimmed value to the onSave prop function. If we now run the component in the browser and we click the + button to add the new item, which causes the List component to render, we can see that we are not wasting time anymore. For example, you can load JSON files, as well as images and other assets, inside the bundle. Last but not least, if we render our application on the server side (we will cover this topic in Chapter 8, Server-Side Rendering for Fun and Profit), the size of the page is bigger when using inline styles. Let's look at how we can optimize it a little bit. We can now run: npm run storybook And point the browser to We can access the Storybook interface where we can see the list of stories on the left. It then clones the array and adds the value as its first element. This makes components use the state even if not needed, which is bad because we have seen that we should avoid using it as much as we can to improve reusability and maintainability. Whenever you compose components, as in the preceding example, you share data between them using props. The next thing to do is to update the tests related to the onClick event handler. ESLint We always try to write the best code possible, but sometimes errors happen, and spending a few hours catching a bug due to a typo is very frustrating. Checking for each variation of text, value, and class name is laborious and it requires much code to cover all instances. There are two main errors: We have a duplicated source of truth If the count prop passed to the component changes, the state does not get updated If we inspect the Counter element using the React Developer Tools, we notice that Props and State hold a similar value: Props count: 1 State count: 1 This makes it unclear which is the current and trustworthy value to use inside the component and to display to the user. CSS Modules are regular CSS, so we can use pseudo classes, Media queries, and animations. This helps users of the package but it also avoids people opening issues and asking questions about how the library works and how it should be used. [184] Server-Side Rendering for Fun and Profit Both sides of the application will be transpiled with Babel and bundled with Webpack before being run, which lets us use the full power of ES2015 and the modules both on Node.js and on the browser. React is one of the most popular libraries for creating user interfaces and it is well-known to be fast, thanks to its smart way of touching the DOM. Common parent Now, the Counter looks a bit better, and the Buttons are reusable. Media queries are crucial for creating responsive applications, and Radium again uses JavaScript to enable that CSS feature in our application. When the array is updated, it gets added back to the state. In fact, our UI consists of buttons with text, buttons with text and icons, and buttons with text and labels. [211] Improve the Performance of Your Applications Let's start with the first one: render() { return ({this.state.items.map(item => (console.log(item)} />))} +) } We added a onClick prop to the Item component and we set it to a function that, when called, logs the current item into the console. React is open source which means that its source code is public and anyone who's signed the Contributor License Agreement (CLA) can help to fix bugs, write documentation, or even add new features. In this way developers will be more likely to use it because it can be adapted to their custom solutions. In the case of the List, for example, we should check if the items array has been modified or not, and return a value accordingly. Consider that we need the same plus and minus buttons in another part of the application. Click on Code Download. [172] Make Your Components Look Beautiful In fact, when all the base rules have been generated, applying those classes to the elements and creating new styles is a very quick process, which is good. Done! If we now run the newly created Todos component in the page, we see that the behavior is the same as before, but the list and the form have two separate states and they render only when their props change. If you come from a static language such as C#, you are used to getting that kind of warning inside your IDE. For example, we can set the state in the following mode: const obj = this.state.obj obj.foo = 'bar' this.setState({ obj }) Even if the value of the foo attribute of the object is changed, the reference to the object is still the same and the shallow comparison does not recognize it. He takes joy in creating software that is accessible for all people. We should avoid optimizing our components for the sake of it and it is important to apply the techniques that we will see in the following sections only when they are needed. Get in touch with us at for more details. One solution is to write all the attributes on the same line, but this would lead to very long lines, which we do not want in our code (see the following section for how to enforce coding style guides). We have also seen how it is flexible and how we can extend it with configuration and plugins. For years, we used to write two applications: a server-side rendered one for the crawlers and another one to be used on the client side by the users. Currying is the process of converting a function that takes multiple arguments into a function that takes one argument at a time, returning another function. Once you've found your bottlenecks, you can apply one the techniques we have seen in this chapter to fix the issue. However, in some case is unavoidable to pass objects and in those cases, we should declare our propType using shapes. Another common solution for this kind of situation is to storing the request as a private member of the component instance: componentDidMount() { this.request = API.get(...) } componentWillUnmount() { this.request.abort() } In that way, the request is encapsulated into the component without affecting the state, so it does not trigger any additional rendering when the value changes. Conventions In this book, you will find a number of text styles that distinguish between different kinds of information. Notice the /server path in the import statement of ReactDOM. Finally, we wrap the component and get a new one: const ListWithGists = withGists(List) We can now put the enhanced component anywhere within our application, and it just works. Suppose we have tens of different fields? Separating markup, styling, and logic into different files when they are tightly coupled and where cannot work one without the other is just an illusion. The first thing we have to do is import all the dependencies: import import import import import chai from 'chai' spies from 'chai-spies' { jsdom } from 'jsdom' React from 'react' TestUtils from 'react-addons-test-utils' Button from '../button' As you may notice here, there are way more compared to Jest because Mocha gives you the freedom to choose your tools. That is pretty easy, but we can create more complex animations, and we can also animate different states of the component. [156] Make Your Components Look Beautiful For example, the button that we created earlier in this section is rendered in the following way: Click me! By itself, it does not seem very hard to read, but if you imagine you have hundreds of elements and hundreds of styles, you realize that the problem becomes very complicated. Let's see if we can do even better adding comments: /** * A generic button with text. Let's now see how to use the withClassName HoC in our components. That is how we expect it to work but, to have a clear idea of what React is doing, we need to install a new tool which helps us store and display performance-related information. [68] Create Truly Reusable Components It gives us also some utilities, like oneOf, that accept an array of types which are valid for a particular property. Mapt gives you full access to all Packt books and video courses, as well as industry-leading tools to help you plan your personal development and advance your career. Great; if we run npm test we see that the first test passes. The next step is installing React and ReactDOM: npm install --save react react-dom And the TestUtils, which we will need to render our components in the test environment: npm install --save-dev react-addons-test-utils The basic functionalities for testing with Mocha are ready but to get feature parity with Jest we need three more packages. The HoCs that the library offers are small utilities that we can use to wrap our components, moving away some logic from them and making them more dumb and reusable. On each iteration of the spring function, the child function receives the interpolated style for the given point in time and, just by applying the received object to the style attribute of the component, we can see the transition of the opacity. I would like to thank my wife and my son for making my days better with their smiles. The React documentation says that at Facebook they use Jest to test their components. Data fetching In the preceding section, we saw the different patterns we can put in place to share data between components in the tree. First of all, we have to type the following command in the terminal: npm install --save-dev why-did-you-update And add the following snippet after the import statement of React: if (process.env.NODE_ENV !== 'production') { const { whyDidYouUpdate } = require('why-did-you-update') whyDidYouUpdate(React) } We are basically saying that, in development mode, we want to load the library and patch React using the whyDidYouUpdate method. Let's now see how we can declare our elements inside our components. Let's start with a simple example: const Button = React.createClass({ handleClick() { console.log(this) }, render() { return }, }) With createClass, we can set an event handler in this way and rely on the fact that this inside the function refers to the component itself. The first thing we have to do to create an event handler is transform the stateless functional component into a class: class JSONSchemaForm extends React.Component Inside the constructor, we bind the event handler: constructor(props) { super(props) this.handleSubmit = this.handleSubmit.bind(this) } In the preceding example, we just log the form data into the console, but in a real-world application you may want to post the fields to an endpoint. Moreover, they might need new features and they could help you implement them so that everyone, yourself included, can benefit from it. At that point, we realize that typing into the field becomes laggy. It is important to keep in mind that we should always try to pass primitive props to components because they are simpler to validate and to compare (we will see the benefits in Chapter 10, About Testing and Debugging). To do that we create a folder called .storybook in the root folder of our application. This is a pretty common mistake when using React and it can be easily fixed by refactoring the components a bit. Now that we have the SSR representation of our app, we can use the template function to apply it to the HTML template and send it back to the browser within the Express response. ESLint can parse JSX without any plugins (just by enabling the flag), but we want to do more. There are many different tools that we can use to create tests with React and we'll cover the most popular ones to understand the benefits they provide. [283] Next Steps Once the issue is filed, you have to wait for one of the core contributors to examine it and tell you what they've decided to do with the bug. Other use cases are numbers: they can be written without quotes or units of measurement and, by default, they are treated as pixels. Again, this works fine, and we can do some interesting things in this way, but it does not handle complex scenarios without requiring us to write a lot of boilerplate code. It is important to bear in mind common solutions when it comes to testing complex components such as Higher-Order Cmponents or forms with multiples nested fields. With Webpack, in fact, you can potentially load any dependencies other than JavaScript, if there is a loader for it. The render method of a component returns a tree of React elements, which React uses to decide which DOM operations have to be done to update the UI. Let's go through all of them in detail so you can have all the information you need to choose the best way for the needs of your team and your projects. Let's start by moving into an empty folder and running the following to create a new package: npm init When the package.json has been created, it is time to install the dependencies. We just attach our handler to the onSubmit callback of the Form component provided by the library and we have created a working form easily. Let's first take a step back and revisit the basic concepts, so that we can move forward and create a living style guide of components by the end of this chapter. However, sometimes we simply cannot manage this and our applications become slow, even if the DOM is not touched at all. React has figured out that the change needed to append a child at the end of the current list is to insert a new LI element as a child to the existing list at the index number two (third element). Then some static attributes such as type and autofocus are set, and the value to control the input is set using the text property, which is updated on each change. Let's begin updating the tests, starting with the renders with text ones. [102] Compose All the Things That is a big win because we did not have to change the parent, we can use the context without worrying about future API changes, and the Price component is now reusable. We just have to create an object where the attributes are the CSS rules and the values are the values we would use in a regular CSS file. Style A pretty significant difference is the way the style attribute works. We will see how event handlers work in Chapter 6, Write Code for the Browser. One of the easiest is to install a third-party library that can provide the information automatically. Log in or register to our website using your e-mail address and password. The last thing to do is to close the test block: }) If you now switch to the console and type: npm test You should see something like this: FAIL ./button.spec.js renders with text expect(received).toBe(expected) Expected value to be (using ===): "button" Received: "div" The test fails, and this is something we expect when we run the tests for the first time doing TDD because the component has not been implemented yet. He currently works on Instructure's flagship product, Canvas, an open source learning management system. Here is the examples section of the official React.js website: [20] Clean Up Your Code To begin with, JSX is one of the main reasons why people fail to approach React, because looking at the examples on the home page and seeing JavaScript mixed with HTML for the first time seems strange to most of us. If we select the first one, we see the list of titles only, while if we selected the second one, we see the list of titles and excerpts. Livery Place 35 Livery Street Birmingham B32PB, UK. The most visible one is that the code for filling the fields is duplicated, apart from some variables. The component we are going to test is TodoTextInput from the Redux TodoMVC example:

/TodoTextInput.js You can easily copy it into your Jest project folder. My favorite feature of CSS Modules is composition. Check the render method of `List`. Now, create a file with the following content: var foo = 'bar'; (Note that we are using var here because ESLint does not know yet that we want to write code in ES2015.) If we run eslint index.js we get the following: Extra semicolon (semi) This is great; we set up the linter and it is helping us follow our first rule. This is particularly helpful when the codebase grows and we have multiple developers working on different components and styles. Let's start with a very simple example, where we set the background color, the padding, the size, and a couple of other CSS properties. The CSS to make an element look like a button is pretty standard, and so we are not going to go through it. [269] 11 Anti-Patterns to Be Avoided With this book, you have learned how to apply best practices when writing a React application. Using JavaScript for both logic and templating not only helps us separate our concerns in a better way, but it also gives us more power and more expressiveness, which is what we need to build complex user interfaces. It is astonishing how, in the JavaScript world, as soon as a specification is announced or drafted, someone in the community implements it as a transpiler plugin or a polyfill, letting everyone else play with it while the browser vendors agree and start supporting it. You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Other available placeholders are [path], which represents the path of the CSS file, and [name], which is the name of the source CSS file. Using the jest.mock function we are telling the test runner to replace the external module with the function we passed as a second parameter. We will see the following: How components communicate with each other using props and children The Container and Presentational pattern and how it can make our code more maintainable The problem mixins tried to solve and why they failed What HoCs are and how we can structure our applications in a better way, thanks to them The recompose library with its ready-made functions How we can interact with the context and avoid coupling our components to it What is the Function as Child component pattern and what are its benefits? However, adding the property only sometimes is not enough because it does not tell us if the component works without the props. All the examples in this section are available in a branch on GitHub. We can create the mock of the import statement in the DOM and it is particularly helpful to show which is the source of the component when... [1] ... which runs like this: npm install eslint --save-dev ... This will ... For example, if we receive the currency and the price from the props, and we always show them together, we may think that it would be better to store it in the state and use the state value inside the render as follows: class Price extends React.Component { constructor(props) { super(props) this.state = { price: `${props.currency}${props.value}` } } } 65 Create Truly Reusable Components render() { return {this.state.price} } } This would work if we create it like this in the parent component: The problem is that if the currency or the value change during the lifetime of the Price component, the state never gets recalculated because the constructor is called once and the application shows the wrong price. Consider the following example: On Recompose As soon as we become familiar with HoCs, we realize how powerful they are and how we can use them to get the most out of them. With inline styles, all the styles of the items are listed in their style attribute, which makes it very hard to read and debug the result. The function can simply inspect the object returned from the renderIntoDocument function; instead, we have to use a function from the TestUtils library to get the button element that we are looking for: const button = TestUtils.findRenderedDOMComponentWithTag(tree, 'button') As the function name suggests, it searches for a component with the given tag name inside the tree. In this case, it's testing legacy code that we may inherit from other teams, or code that we might find when joining a new company. Writing an isomorphic application means building an application that looks the same on the server and the client. Let's look at how to apply a simple fade-in effect to a text with the React add-on, and then we will perform the same operation using react-motion, a third-party library that makes creating complex animations even easier. Props are the way to decouple the components we need and create a clean and well-defined interface. If a new list of categories has to be shown, the user can always load them and it's done. You will encounter this concept very often in the React ecosystem, especially if you look into libraries such as Redux. [69] Make Your Components Look Beautiful Activating the previous configuration option, the result we have in the browser is as follows: Click me! This is way more readable and easier to debug. Edge cases are states that the component can assume when, for example, all the props are in use to its layout. Rendering a component in the DOM and it is tricky to predict the final style of an element just by knowing the class names applied to it, because styles are not isolated and other rules in other parts of the application can affect the final result. The Perf add-on also gives us interesting information related to React operations in the DOM and if we should do some work to improve the performance of our applications. The result of the following code is something like the following: Hello React! Here, title, and h1 are all applied automatically to the element. Functional Programming is a declarative paradigm, where side-effects are avoided and data is considered immutable to make the code easier to maintain and to reason about. Typically with Mocha, you have two functions to define your test: the first one is describe, which wraps a set of tests for the same module, and then there is it, which is where tests are implemented. To do so, we have to run webpack in a new terminal and use it to check and debug the result. The function can simply inspect the object returned from the renderIntoDocument function; instead we have to use a function from the TestUtils library to get the button element we are looking for. const button = ... As a general rule, it is always better to use the component and the tools that support it... If we compile the following code... With Babel, we can transform the JSX into JavaScript... The bundle is... In React, it's crucial to keep our components clean and well-defined... It is important to note that every prop we pass to a component is a legacy... Children JSX allows you to define children to describe the tree of elements and compose complex UIs. A basic example is a link with text inside it, as follows: Click me! This would be transpiled into the following: React.createElement("a", { href: "", }, "Click me!") Our link can be enclosed inside a div for some layout requirements, and the JSX snippet to achieve that is as follows: Click me! The JavaScript equivalent is as follows: React.createElement("div", null, React.createElement("a", { href: "", }, "Click me!")); [24] Clean Up Your Code It should now be clear how the XML-like syntax of JSX makes everything more readable and maintainable, but it is always important to know the JavaScript parallel of our JSX to have control over the creation of elements. [6] We create a Page object that represents the elements of the page and hides the selectors, and we use it to fill the fields and submit the form, we'll get many benefits and avoid duplicated code. If you are used to using binders, you might wonder why the linter is not complaining about that code. Finally, in the render method, we define the button with the onClick attribute to which we attach our event listener: render() { return (Click me!) } Now, suppose we want to attach a second handler to the button which listens to the double click event. [6] Everything You Should Know About React Hello, readers! This book assumes that you already know what React is and what problems it solves for you. It is a common practice to prefix HoCs that provide some information to the components they enhance using the with pattern. These concepts are pretty advanced, and they are not part of the scope of this chapter, but if you are interested in them, you should check out the ECMAScript proposals. To see the warning in the console, a basic operation we can do in this section is the following: const Spread = () => <div /> The message we get looks like the following: Unknown prop `foo` on tag. Then we create a separate list which only receives the items and a form which lets the user to add new items to the list. Also, using props instead of state for the controlled component, you might as well say that the type... The components are the building blocks of a tree of DOM nodes, which is the desired result of... in the screen. The last thing to do is change the... any complex logic and just passing down the props. It keeps on performing the same operation recursively on the result until it gets a tree of DOM nodes, which is ready to be rendered on the screen. The last thing to do is change the list component to remove the display logic from it. Publishing a npm package The most popular way of making a package available to developers is publishing it to npm, the popular package manager for Node.js. Also, in production, we may want to extract the CSS file instead of injecting it into the browser from the bundle so that we can have a lighter bundle and cache the CSS on a CDN for better performance. Sometimes it is not easy to figure out which component to re-render or we do not know exactly which one is causing the problem, and it is not easy to quickly identify which components are taking... Now you have noticed how to load all the classes of the CSS, and because JavaScript does not support hyphenated attributes, we are forced to use a camel-cased class name. So, we add a second event handler to the button, as follows: constructor(props) { super(props) this.state = { value: '', } this.handleChange = this.handleChange.bind(this) this.handleSubmit = this.handleSubmit.bind(this) } We may also want to initialize the value property of the state as an empty string, in case the button gets clicked before any change event is triggered. As a final example, let's see how to create a mixin for React to be declarative. Then we run the function and we set the state of the end goal is to encapsulate every single technology used to create our components and separate the concerns according to their domain and functionalities. Using React for the first time usually requires an open mind because it brings a new way of designing web and mobile applications. With public code and many people contributing to it, tests are very helpful for many reasons: They make the code more robust They help other developers understand what the code does They make it easier to find regression when new code is added They make other contributors more confident in writing the code The second important thing to do is add a README with description of the component, an example of its use, and documentations of the APIs and props that can be used. Each file inside that folder represents a snapshot. Components can be defined in the right way, if they are pure and aim for composability and reusability, they can be tested as simple functions. We went through some of the most common patterns to make child and parent communicate using callbacks. In a {function()}. So, in order to test a browser event, we have to render our component in a detached DOM. In fact, it is true that React is smart enough to merge lifecycle callbacks, it so cannot do anything if two mixins define or change the same attribute in the state. The bundle is... Anti-Patterns to Be Avoided, explains that bad developers often try to use shortcuts and creative solutions, but in some cases these workarounds can be dangerous for their applications, especially with big teams and large codebases. Function as Child There is a pattern that is gaining consensus within the React community, called Function as Child. In general, the right approach is to always be ready to unlearn everything and change your mindset if the long-term benefits are worth it. Writing a comprehensive set of tests is vital to creating rock-solid and maintainable code. We learned how inline styles work in React and what the implications are. Stated previously, one of the main reasons we should prefer JSX over React's createElement is because of its XML-like syntax and because balanced opening and closing tags are perfect to represent a tree of nodes. [230] About Testing and Debugging In the next section, you will learn how it's possible to test components using Mocha. We can replace the line where we used the renderIntoDocument method with the following one: const tree = shallow( ) Calling the shallow function returns a shallow rendered component that can be accessed more easily, as shown in the previous test. Atomic CSS for a bit and we found out that it makes prototyping incredibly fast. Now that we are JSX power users, it is time to move on and see how to follow a style guide within our code to make it consistent. In fact, in CSS, the order matters and if the CSS is loaded on-demand, the order is not guaranteed which leads to the wrong styles being applied to the elements. For example, we can ask our users to enter some information using forms, and we will look at how, in React, we can apply different techniques to achieve the same result. Here are the URLs to be included in the HTML to start using React: We include the core React library, which can use is not a standalized language supported by the browser; but the whole point is to start with the bare minimum set of features and add more functionalities as soon as they are needed. Once the tests are written, we can be pretty confident that the component behaves as expected. We can now implement the component, extending React.Component: class Items extends React.Component At the moment it only implements the render method, which loops through the items received from the parent and displays a list. Those are the most interesting in the time when rendering unchanged components, rather than the DOM operations that have been executed: componentDidUpdate() { Perf.stop() Perf.printWasted() } Nice! If we now enter the application and type... [242] About Testing and Debugging Let's see what it means to convert our Jest tests to use Enzyme instead of TestUtils. [154] Make Your Components Look Beautiful Let's look at how this looks in a way that is to use inline styles instead. In fact, when we configure ... we use Webpack to ... a bit more than a ... of using the DefinePlugin, since the plugin can... that sets the environment variable inside the bundle... To improve the Performance of Your Applications a second time... not only wanted to consider the bundle with the production flag activated but we also should keep in mind that... we want to write code... Recommended configurations to activate a set of rules that... a source of true is... The main reason... define them. It is maintained by the great developers at Formidable Labs, and it is still one of the most popular solutions. For example, we may want to print out a message... this is... as simple as it looks: the children function is defined in the render method of the parent and it... Hello, World! Instead, it must be a... a different... frustration and help us improve our code... tell us what we need... Luckily, there are some very useful presets of configurations, which we can easily install via npm... Now, the first JavaScript code... one of... the value of innerWidth of the window will be available in the state of our component and the component will re-render with the updated value anytime innerWidth changes. Again, being able to perform operations declaratively is incredibly powerful, and it makes the code more easier to reason about and keep the code clean. Then we create a function that we have to be in a function that is... the browser to draw icons and graphs in Scalable Vector Graphics (SVG). We are using a new property, ... const Gists = ... the gist.description ... [115] Proper Data Fetching render() { return ({this.props.gists.map(gist => ... }) } ... Thanks to what we can do on the client side... The state of the application... Reconciliation... [153] Make Your Components Look Beautiful First, let's look at an example of a DOM element... React uses the renderToString function... The following code... [15] Everything You Should Know About React Now that we have the basics, let's see how to build a button that works... The property... the element... React DOM and React Native to create the user interfaces of their respective platforms. The updated components are highlighted with colored rectangles and it becomes easy to spot possible optimizations. We use the constant; instead, passing a function to it as a parameter. For example, writing a simple function that, given an array of uppercase strings, returns an array with the same strings in lowercase: toLowerCase(['FOO', 'BAR']) // ['foo', 'bar'] ... The main difference... output.push(input[i].toLowerCase()) } return output } First of all, an empty array to contain the result gets created. Vice versa; if you find a problem with the latest version of the library that was not in the previous versions, in a regression and it is going to have a higher priority because it may affect a massive number of users. The easiest way to approach the problem is to show the application to the user... The element... an elegant way and it is a good idea to create an... const obj = { foo: 'bar' } ... This structure is more concise than the previous one and it is more readable. ... ES2015 and Babel... there is another way to express the same concept in a way that works for us with immutable objects and there is also a popular library called immutable.js which has more powerful features but it... For example, we can create... Circle.defaultProps = { fill: 'red', } This is pretty powerful when we build UIs... Shallow rendering Mounting the components... at first... are the same... We can see how all the... [28] ... title: 'Create Apps with No Configuration', }, { id: 2, title: 'Mixins Considered Harmful', },] Before running storybook and navigating to our visual collection of components and stories we have to change it. As we saw in Chapter 3, Create Truly Reusable Components, React Storybook can help you create living style guides, which are easier for you to maintain and for the consumer of your package to navigate and use. Currying A common technique in FP is currying. Install the plugin with: npm install --save-dev babel-plugin-transform-react-inline-elements Next, you can add the plugin to the .babelrc file: { "plugins": ["transform-react-inline-elements"] } Both plugins should be used only in production because they make debugging harder in development mode. With React CSS Modules we can write our component in the following way: import React from 'react' ... that gets compiled into JavaScript. Another solution is to add the linting before opening a pull request so that we have the chance to clean up the code before our colleagues start reviewing it. This works, but it is not very convenient to run it manually all the time. Instead, we can create a helper function inside our component and use it in JSX to verify the condition: canShowSecretData() { const { dataIsReady, isAdmin, userHasPermissions } = this.props return dataIsReady && (isAdmin || userHasPermissions) } {this.canShowSecretData() && } As you can see, this change makes the code more readable and the condition more explicit. If you do not like using functions, you can use an object's getters, which make the code more elegant. First of all, mixins sometimes use internal functions to communicate with the component. This section does not aim to be an exhaustive guide to creating animations and beautiful UIs; the goal here is to provide you some basic information about the common solutions we can put in place to animate our React components. The great thing is that we can use the class name as a string (for example, "button"): const Button = () => Click me! If we now render the EnhancedButton into the DOM, we will see that nothing has really changed from before, which means that the library works. As we have seen in Chapter 3, Create Truly Reusable Components, in fact, one of the reasons why we use the state object is to store values that are needed inside the render method. CSS Modules If you feel that inline styles are not a suitable solution for your project and your team, but you still want to keep the styles as close as possible to your components, there is a solution for you, called CSS Modules. What it does instead is attach a single event handler to the root element, which listens to all the events, thanks to the event bubbling. Since we are using Webpack to create the bundle, we will use the Babel loader to transpile our ES2015 code within Webpack itself: npm install --save-dev babel-loader babel-core babel-preset-es2015 babel-preset-react Finally, we install the style loader and the CSS loader, which are the two loaders we need to enable the CSS Modules: npm install --save-dev style-loader CSS-loader There is one more thing to do to make things easier, and that is install the html-webpackplugin, which is a plugin that can create an HTML page to host our JavaScript application on the fly, just by looking into the Webpack configuration and without us needing to create a regular file: npm install --save-dev html-webpack-plugin Last but not least, we install react and react-dom to use them in our simple example: npm install --save react react-dom Now that all the dependencies are installed, it is time to configure everything to make it work. In general, the right path to follow is starting with a single component and splitting it only when the logic and the presentation become too coupled where they shouldn't. One of the most popular configurations used in the React world is without any doubt the Airbnb one. Other than npm, some developers may need to add your component as a global dependency and use it without a package manager. The only requirement is to use a node.js/npm environment and install the CLI tool globally: npm install -g create-react-app When the executable is installed, we can use it to create our application passing a folder name: create-react-app hello-world Finally, we move into the folder of our application with cd hello-world and we just run: npm start [16] Everything You Should Know About React Magically, our application is running with a single dependency, but with all the features needed to build a complete React application using the most advanced techniques. Higher-order Components In the previous section, we saw how mixins are useful for sharing functionalities between components and the problems that they bring to our applications. It is not hard to understand the reasons behind this. [34] Clean Up Your Code Loops A very common operation in UI development is to display lists of items. Running npm test in the console gives us a green bar with six passed tests. What renderToString does is return a string representation of the DOM elements generated by our App component, the same tree that it would render in the DOM if we were using the ReactDOM render method. For now, we are only interested in the way they are bound to the components we are defining. If the condition has an alternative, (the classic if...else statement), and we want, for example, to show a logout button if the user is logged in and a login button otherwise, we can use JavaScript's if...else, as follows: let button if (isLoggedIn) { button = } else { button = } return {button} Alternatively, and better, we can use a ternary condition, which makes the code more compact: {isLoggedIn ? In this case, any element that we wrap inside the Button component will be rendered as a child of the button element with the btn class name. [88] Compose All the Things In this case, our function receives the latitude and longitude from the owner, and it returns the markup structure to display it. [183] Server-Side Rendering for Fun and Profit For this reason, my suggestion is to build the client-side version first, and only when the web application is fully working on the server should you think about improving the experience by enabling SSR. before onClick() { return console.log(item); } after onClick() { return console.log(item); } Secondly, we can see that the Perf add-on is informing us that we are wasting some time rendering the List > Item component. The props of the new reusable List are the following: List.propTypes = { collection: React.PropTypes.array, textKey: React.PropTypes.string, titleKey: React.PropTypes.string, } Since the List is not going to have any state or function, we can write it as a stateless functional component: const List = ({ collection, textKey, titleKey }) => ({collection.map(item =>)}) The List receives the props, and iterates over the collection, mapping all the items into another component (that we are going to create next). First of all we have to create a Page object using a class: class Page The class has a constructor that receives the root wrapper from Enzyme and stores it for future usage: constructor(wrapper) { this.wrapper = wrapper } Then we define a generic function to fill the fields that accepts name and values and fires the change event: fill(name, value) { const field = this.wrapper.find(`[name="${name}"]`) field.simulate('change', { target: { name, value } }) } [265] About Testing and Debugging Then we implement a submit function to abstract the part where we look for the button and simulate the browser event: submit() { const form = this.wrapper.find('form') form.simulate('submit', { preventDefault() {} }) } We are now able to rewrite the previous test in the following way: test('submits the form with the page object', () => { const onSubmit = jest.fn() const wrapper = shallow() const page = new Page(wrapper) page.fill('firstName', 'Christopher') page.fill('lastName', 'Chedeau') page.submit() expect(onSubmit).toHaveBeenCalledWith('Christopher Chedeau') }) As you can see, we created an instance of the Page Object and we used its function to fill the fields and submit the form. [79] Create Truly Reusable Components By clicking on each one of them we can see the component updating on the right. This seems like a complex concept, but it is not. For example, we can create a class to set the margin bottom to zero: .mb0 { margin-bottom: 0; } We can use another one to set the font-weight to six hundred: .fw6 { font-weight: 600; } Then, we apply all those atomic classes to the elements: Hello React This technique is controversial, and particularly efficient at the same time. Most of the time, refactoring the components and putting the state in the right place can greatly assist the optimization of the application. [127] Write Code for the Browser We just have to change the implementation of the handler to store into the state instead of logging it, as follows: handleChange({ target }) { this.setState({ value: target.value, }) } Getting notified of when the form is submitted is very similar to listening to the change event of the input field; they are both events called by the browser when something happens. One solution is to split it into smaller functions in a way that lets us keep all the logic in the same component. Second, if we open DevTools and we have the React extension installed, we can see that the App component has been booted on the client as well. Let's create a class called Focus with a constructor where we bind the handleClick method: class Focus extends React.Component We will listen to the click events on the button to focus the input field: constructor(props) { super(props) this.handleClick = this.handleClick.bind(this) } Then we implement the actual handleClick method: handleClick() { this.element.focus() } As you can see, we are referencing the element attribute of the class and calling the focus method on it. The following code is wrong because we are storing a value in the state to use it later but we do not access it in our render method, and we fire an unnecessary render when we set the new state: componentDidMount() { this.setState({ request: API.get(...) }) } componentWillUnmount() { this.state.request.abort() } [66] Create Truly Reusable Components In a scenario like the previous one, it would be preferable to keep the API request stored in an external module. If you find a mistake in one of our books-maybe a mistake in the text or the codewe would be grateful if you could report this to us. Webpack Before diving into CSS Modules and learning how it works, it is important to understand how it was created and the tools that support it. Avoidable re-render! {item: "bar"} [210] Improve the Performance of Your Applications after Object {item: "bar"} In fact, even if React does not touch the DOM for the foo and bar items, their render methods are still being called with the same props, which makes React perform an additional job.) } } As a result of this, if you open the DevTools, you can see that Radium injected the following style into the DOM: @media (max-width: 480px){ .rmq-1d8d7428{width: 160px !important;}} The rmq-1d8d7428 class has been applied to the button automatically, as well: Click me! If you now resize the browser window, you can see that the button becomes smaller for small screens, as expected. The framework should not tell you how to separate the concerns, because every application has its own, and only the developers should decide how to limit the boundaries of their apps. We are now ready to learn the various techniques we can put in place to compose our components. One thing that we want to avoid is loading real data and, most importantly, we do not want our test to fail if the external module fails for some reasons. [140] Write Code for the Browser The callback we defined is called when the component gets mounted, and the element parameter represents the DOM instance of the input. In the case of inline styles, unfortunately, the problems are many. We can now move into the part where the posts are displayed: render() { return ({this.state.posts.map(post => ({post.title} {post.excerpt && {post.excerpt}}))}) } Inside the render method, we loop through the posts, and we map each one of them into a element. Suppose you are using createClass in your application and you find yourself needing to write the same code into different components. Its creator, Ryan Florence, is the co-creator of one of the most popular React libraries: react-router. If the problems exist, then you have to write the fix and publish a new version of the library. Various templating solutions have been created to help developers achieve this. Tips and tricks appear like this. A common solution is to write each attribute on a new line, with one level of indentation, and then align the closing bracket with the opening tag: Conditionals Things get more interesting when we start working with conditionals, for example, if we want to render some components only when certain conditions are matched. In order to use it, you must first install some dependencies: npm install --global ^2.9.0 ^1.2.0 ^1.7.0 ^5.0.1 [44] Clean Up Your Code Then add the following configuration to your .eslintrc: { "extends": "airbnb" } Try to run ESLint again against your React source files and you will see if your code follows the Airbnb rules and if you like them. If you're interested in joining, or would like to learn more about the benefits we offer, please contact us: Table of Contents Preface Chapter 1: Everything You Should Know About React Declarative programming React elements Unlearning everything Common misconceptions Summary Chapter 2: Clean Up Your Code JSX Babel Hello, World! DOM elements and React components Props Children Differences with HTML Attributes Style Root Spaces Boolean attributes Spread attributes JavaScript templating Common patterns Multi-line Multi-properties Conditionals Loops Control statements Sub-rendering ESLint Installation Configuration React plugin Airbnb configuration The basics of functional programming First-class objects 1 7 8 10 11 14 18 19 20 21 22 23 23 24 25 25 25 26 26 27 28 28 29 29 30 30 35 35 37 38 39 39 42 44 45 45 Purity Immutability Currying Composition FP and user interfaces Summary 46 47 47 48 48 49 Chapter 3: Create Truly Reusable Components Creating classes The createClass factory Extending React.Component The main differences Props State Autobinding Stateless functional components Props and context The this keyword State Lifecycle Refs and event handlers No reference to component Optimization The state External libraries How it works Asynchronous React lumberjack Using the state Derivables The render method Prop types React Docgen Reusable components Living style guides Summary Chapter 4: Compose All the Things 50 51 51 52 52 52 54 55 57 58 59 59 59 59 60 60 61 61 61 62 64 64 65 66 67 70 72 76 80 81 Communication between components Children Container and Presentational pattern Mixins [ii] 82 83 85 91 Higher-order Components Recompose Context Function as Child Summary 94 98 101 103 105 Chapter 5: Proper Data Fetching 106 Data flow Child-parent communication (callbacks) Common parent Data fetching React-refetch Summary 107 109 110 111 117 124 Chapter 6: Write Code for the Browser 125 Forms Uncontrolled components Controlled components JSON schema Events Refs Animations React motion Scalable Vector Graphics Summary 126 126 131 134 136 139 143 145 147 149 Chapter 7: Make Your Components Look Beautiful CSS in JS Inline styles Radium CSS Modules Webpack Setting up a project Locally scoped CSS Atomic CSS Modules React CSS Modules Styled Components Summary 150 151 153 158 162 162 163 166 172 173 175 178 Chapter 8: Server-Side Rendering for Fun and Profit Universal applications 179 180 [iii] Reasons to implement Server-Side Rendering SEO A common code base Better performance Don't underestimate the complexity A basic example A data fetching example Next.js Summary Chapter 9: Improve the Performance of Your Applications Reconciliation and keys Optimization techniques Should component update Stateless functional components Common solutions Why did you update? The API is nicer, similar to jQuery, and it provides many useful utilities to interact with components, their states, and their properties. [245] About Testing and Debugging It is worth going quickly through the implementation of the component so that we can understand what it does before testing. We will also go through all the features of CSS Modules and look at the problems they can solve. If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at www.packtpub.com/authors. [71] Create Truly Reusable Components Reusable components We have seen what are the best ways to create components and the scenarios where it makes sense to use a local state. At the end of the chapter we talked about immutability and we've seen why it's important not to mutate data to make shouldComponentUpdate and the shallowCompare do their job. If you find any errata, please report them by visiting selecting your book, clicking on the Errata Submission Form link, and entering the details of your errata. Run the following command in the console and point the browser to . React utilizes different techniques to optimize the rendering of the components on the screen. If we follow the rules, we can easily figure out whenever a component should be stateless or stateful and how to deal with the state to make our component reusable across the application. In this chapter, we covered four different ways of using components that can harm the performance and behavior of our web applications. There are two important things to note here: if we use the View Page Source feature of the browser, we can see the source code of the application being rendered and returned from the server, which we would not see if the SSR were not enabled. If we run the component and click the + button, we can see the operations are listed in a table in the DevTools console. We may also want to cover all the various states that a component can have. I would also like to thank Packt Publishing for giving me this opportunity. It also worth saying again that React already implements some techniques to make our applications quicker and smoother and so most of the time we do not need anything more than this. The best thing to do is to publish it to GitHub and make it available for everyone to read and use. The way they usually work is that the server returns an empty HTML page with a script tag to load the application. We do not want duplicated code in our applications, and React gives us a tool that we can use when we want to share functionalities across various components: mixins. Our component is fully tested and correctly implemented. React helps us by creating well-defined components and building a style guide, so it does not require too much effort. React components typically contain a mix of logic and presentation. Therefore, our function is called once for each typed character. Consequently, to master React, it is important to understand what declarative programming means and what the main differences between imperative and declarative programming are. Working with big teams, it is non-trivial to know if a particular class or element has already been styled, and most of the time we tend to add more classes instead of reusing existing ones. Don't consider the following section an exhaustive guide to functional programming; it is only an introduction to get started with some concepts that are commonly used in React and of which you should be aware. Unfortunately, we cannot define the logging function once in the parent because we need to know which child has fired it; so that is why creating it inside the loop could seem the best solution. Another tool which helps us easily collect information about our component's performance is react-perf-tool which is a component that we can install, import and add inside our application to get a nice interface to manage the Perf add-on within the browser window. In case we need it, React gives us access to the actual DOM nodes in a way that we can perform imperative operations with them, which is useful if we need to integrate React with an existing imperative library. For this example, we will use the Controlled form component created in Chapter 6, Write Code for the Browser: class Controlled extends React.Component Let's go quickly through its function to remind ourselves how it works, and then we can talk about testing. Then we create the onClick mock, using the jest function: const onClick = jest.fn() Here comes the part where we fully render the component into the DOM: const tree = TestUtils.renderIntoDocument( ) If we log the tree, we can see that we get back the real component instance and not just the React Element. For example, we can change the value of the loader as follows: loader: 'style!css?modules&localIdentName=[local]--[hash:base64:5]', Here, localIdentName is the parameter and [local] and [hash:base64:5] are placeholders for the original class name value and a five-character hash. The main concept is that, instead of passing a child in the form of a component, we define a function that can receive parameters from the parent. As a final task, we can create a second story. Other than Shallow rendering, which we can use to cover most scenarios, there is a mount method that can be imported from the library and renders the tree into the DOM. There are methods to help make assertions and expectations on props, and functions to inject arbitrary state and context into components. These tools are called module bundlers, and what they do is load all the dependencies of your application into a single bundle that can be executed in the browser, which does not have any concept of modules (yet). We can change the preceding function to make it immutable using concat, which returns a new array without modifying the given one: const const const const add3 = arr => arr.concat(3) myArr = [1, 2] result1 = add3(myArr) // [1, 2, 3] result2 = add3(myArr) // [1, 2, 3] After we have run the function twice, myArr still has its original value. In most cases, a good solution would be to add multiple parameters to the Button or to create different versions of the Button, each one with its single specialization, for example, IconButton. Consider the following functions: const add = (x, y) => x + y const square = x => x * x These functions can be composed together to create a new function that adds two numbers and then doubles the result: const addAndSquare = (x, y) => square(add(x, y)) Following this paradigm, we end up with small, simple, testable pure functions that can be composed together. Otherwise, we might expect both Nice components to be rendered and the Evil not to be shown. [237] About Testing and Debugging If we run the tests with npm test they will fail because the onClick functionality hasn't been implemented yet. This will render a read-only field. The result of the transformation is then passed to the style loader, which injects the styles into the head of the page. For more complex components, you can add multiple stories and show all the possible states and variations that each component can assume. Inline styles The official React documentation suggests developers use inline styles to style their React components. Let's see an example of a template: {{#items}} {{#first}} {{name}} {{/first}} {{#link}} {{name}} {{/link}} {{/items}} The preceding snippet is taken from the website of Mustache, one of the most popular templating systems. [104] Compose All the Things Summary In this chapter, we learned how to compose our reusable components and make them communicate effectively. In the data fetching examples in Chapter 5, Proper Data Fetching, we looked at how we can use componentDidMount to fire the data loading. Finally, the expectation remains the same: expect(onClick).toBeCalled() The final code is something like this: test('fires the onClick callback', () => { const onClick = jest.fn() const button = shallow() button.simulate('click') [244] About Testing and Debugging expect(onClick).toBeCalled() }) Migrating to Enzyme is relatively easy, and it makes the code more readable. [105] 5 Proper Data Fetching The goal of this chapter is to show the different data fetching patterns that we can put in place in a React application. Anti-Patterns to Be Avoided One of the best ways to learn something is by looking at the code, so we will start by creating a simple component with a + button to increment a counter. React tries to apply the smallest possible number of operations on the DOM because touching the Document Object Model is an expensive operation. It is very hard, in fact, to clearly state that a particular component depends on a specific CSS and that the CSS has to be loaded for the style to be applied. For example, you need to listen to the window resize event to get the size of the window and do some operations accordingly. Inside the loop, there is some conditional logic to check if the #first and the #link properties exist, and depending on their values, a different piece of HTML is rendered. We first install the library using: npm install --save react-component-errors Then we import it inside your component file: import wrapReactLifecycleMethods from 'react-component-errors' And then we decorate your classes in the following way: @wrapReactLifecycleMethods class MyComponents extends React.Component [268] About Testing and Debugging This library not only avoids breaking the entire tree when a single components fails, but it also provides a way to set a custom error handler and get some valuable information when an exception occurs. Again this book assumes that you've already played with React in a small/medium application which means that you must have created some components before. The mixin was simply listening to the window resize event and making the updated innerWidth property of the window available into the state. This feature lets you create event handlers assuming that, when they get called, this references the component itself. It has been proven by the major e-commerce websites that a few milliseconds added to the page load can have an enormous impact on revenues. [119] Proper Data Fetching We also have to update the propTypes and change the name of the received prop and its type: List.propTypes = { gists: React.PropTypes.object, } Now that we have this library in our project, we can add more functionalities to our List component. About the Reviewer Clay Diffrient is a JavaScript enthusiast who is always looking to improve and do more. It is incredibly hard to control all the various combinations of styles, rules, and navigation paths, but again, being able to load the CSS when needed could have a critical impact on the performance of a web application. Let's move back to the TodoTextInput.spec.js and add a new test: test('does not fire onSave on blur if new', () => { const onSave = jest.fn() const wrapper = shallow( ) wrapper.simulate('blur') expect(onSave).not.toBeCalled() }) The test is similar to the last one in the file except that we pass the newTodo prop to the component and we check that the onSave callback does not get called. In our example, we started and stopped monitoring using the lifecycle hooks of the components, as follows: componentWillUpdate() { Perf.start() } [224] Improve the Performance of Your Applications componentDidUpdate() { Perf.stop() Perf.printOperations() } After calling the stop method, we also used the printOperations function to print, in the browser console, the current DOM operation that React has made to apply the required changes. So, for example, a mixin can update a special attribute in the state of a component, then the component re-renders, taking into account the new attribute. We use a class because we need to store an internal state and use the life cycle methods: class Gists extends React.Component Then, we define a constructor, where we initialize the state: constructor(props) { super(props) this.state = { gists: [] } } Now comes the interesting part: data fetching. : null} The If component is great, but if for some reason you have nested conditions in your render method, it can easily become messy and hard to follow. We've looked at the internal state of a component and at when it is better to avoid using it. If you needed to share data between the server-side rendered pages and the client-side application, you could inject some variables inside a script tag. ESLint and its plugins are very well documented and for any single rule, you can find the description of the rule and some examples of when it passes and when it fails. It also deals with data manipulation, and event handling. The most popular library to create animations in React is react-motion, maintained by Cheng Lou. Variables are wrapped into curly braces." A block of code is set as follows: const toLowerCase = input => { const output = [] for (let i = 0; i < input.length; i++) { output.push(input[i].toLowerCase()) } return output } Any command-line input or output is written as follows: npm install -g create-react-app New terms and important words are shown in bold. It has an attribute called loaders, where we set a specific loader for each one of the file types: module: { loaders: [{ test: /\.js$/, exclude: /(node_modules|bower_components)/, loader: 'babel', query: { presets: ['es2015', 'react'], } }, { test: /\.css$/, loader: 'style!css?modules', },], }, We are saying that the files that match the .js regex are loaded using the babel-loader so that they get transpiled and loaded into the bundle. What if we really want to use the prop's value to initialize the component and we know for sure that the value does not change in the future? We have to import the config object from the package like this: import { config } from 'react-component-errors' Then, we can set a custom error handler in the following way: config.errorHandler = errorReport => { ... Of course, if our prop types have clear name and types, that should be sufficient for developers to use them, but we can do more: We can automatically create documentation for our components starting from the definition of the prop types. The next step is adding a function to the onClick event of the button to make the API call with the ID of the gist. Unfortunately, creating a style guide is not always easy in web applications because often the concerns are not well defined, and some elements get duplicated to achieve small variations in requirements. For example, Dan Abramov, an active member of the community and a Facebook employee recently said: "ES6 classes: better the devil that's standardized" while talking about createClass vs extends Component. For example, if we keep on working on the Counter component and we add the following button, which updates the state creating a new foo property: this.setState({ foo: 'bar' })}> Update We can see how clicking the + does not have any visible effect but as soon as we click Update the count value in the browser makes a jump, displaying the current hidden state count value. The reason why this happens is that React knows how to optimize the state update inside event handlers and it batches the operation for better performance. Another aspect worth mentioning is that in the declarative example, there is no need to use variables nor to keep their values updated during the execution. The topic is very hot and highly controversial, so this chapter requires an open mind in order to be understood and followed. If we try to change the styleName property to reference a non-existing class name, as follows: const Button = () => (Click me!) We will see the following error in the console of the browser: Uncaught Error: "button1" CSS module is undefined. Using React, we create our UIs using components we can consider functions, as we will see in the following chapters. In fact, the shouldComponentUpdate function does not exist, and there is not a way to tell React that a functional component should not be rendered if the props (or a particular prop) are not changed. Let's now delve into some more interesting stuff and test whether the class names get applied to the element when the related props are present: test('applies the right class names', () => { const wrapper = shallow( ) expect(wrapper.hasClass('edit new-todo')).toBe(true) }) In this test we add the editing and newTodo props and we check inside the expect function that the classes have been applied to the output. So, if the value of the field is 24, the font size is going to be 24 pixels. It is important not to abuse HoCs because with every abstraction come some problems; in this case, the trade-off is related to performance. Let's create an index.js file, which is the entry we specified in the Webpack configuration, and let's import React and ReactDOM as well: import React from 'react' import ReactDOM from 'react-dom' We then create the simple button. As usual, we will start with a basic version and we will optimize it step by step. This means that the component and the HoC are not coupled, and they can both be reused across the application. In the second part, we looked at how ESLint and its plugins can help us find problems faster and enforce a consistent style guide across our code base. It is also essential to add a LICENSE file to your repository in order to make people aware of what they can and cannot do with your code. [229] About Testing and Debugging Another great power that modern tools bring to us the ability to run tests using Node and the console. JavaScript testing utilities for React At this point, it should be clear how to test basic components with Jest and Mocha and what the pros and cons are of both. Last but not least, we have to start the Express application: app.listen(3000, () => { console.log('Listening on port 3000') }) We are now ready to go: there are only a few operations left. [82] Compose All the Things Children There is a special prop that can be passed from the owners to the components defined inside their render method; it is called children. We learned how to deal with the context without needing to couple our components to it, thanks to HoCs. Finally, we saw how we can compose components dynamically by following the Function as Child pattern. That would work, but it is not optimal because you would end up firing two HTTP calls, one on the Express server and one in the browser. [85] Compose All the Things Creating well-defined boundaries between logic and presentation not only makes components more reusable, but it provides many other benefits, which you will learn about in this section. The constructor function of the class gets called only once when the component is created. However, if we change our code a little: handleClick() { setTimeout(() => { this.setState({ clicked: true, }) console.log('the state is now', this.state) }) } The result is going to be: the state is now Object {clicked: true} This is what we may have expected in the first place, and it's because React does not have any way to optimize the execution and it tries to update the state as soon as possible. This is how the TDD process works. To control the UI flow, React uses a particular type of object, called element, which describes what has to be shown on the screen. What we can do to solve the issue here is to create the array once, and always pass the same instance to every render. Now we want to check if the URL function works: test('uses the function url', () => { const url = jest.fn(props => (` {props.username}/gists`) const withGists = withData(url) const ListWithGists = withGists(List) const props = { username: 'gaearon' } mount() expect(url).toHaveBeenCalledWith(props) expect(getJSON).toHaveBeenCalledWith(') }) We first generate the URL function using a Jest mock so that we can write an expectation on it, then we enhance the List and define the prop that gets passed to the component. It provides a very clean and easy API that gives us a very powerful tool to create any animations. If we change the example above a little bit, in particular the event handler, and instead of appending baz at the end of the list we insert it as a first element, we can see how React behaves in a non-optimal way. In this chapter, we will cover the following topics: The Unidirectional Data Flow of React and how it can make our applications easier to reason about How a child can communicate with its parent using callbacks The way two siblings can share data through their common parent How to create a generic HoC, which can fetch data from any API endpoints How react-refetch works and why it is a useful tool that we can integrate into our projects to make data fetching easier Proper Data Fetching Data flow In the last two chapters, we saw how to create single reusable components and how to compose them together effectively. Summary The journey to learn how to make reusable components has come to the end. This is particularly bad for maintainability because, if a component uses multiple mixins, it ends up implementing different methods, which makes it difficult to eliminate the code when some mixins are removed, or they change their behavior. The following example works fine, because the div is on the same line as the return: return [29] Clean Up Your Code The following, however, is not right: return The reason for this is because you would have the following: return; React.createElement("div", null); This is why you have to wrap the statement in parentheses: return ( ) Multi-properties A common problem in writing JSX comes when an element has multiples attributes. The JavaScript implementation (using the Google Maps SDK) is as follows: const map = new google.maps.Map(document.getElementById('map'), { zoom: 4, center: myLatLng, }) const marker = new google.maps.Marker({ position: myLatLng, title: 'Hello World!', }) [9] Everything You Should Know About React marker.setMap(map) It is clearly imperative, because all the instructions needed to create the map, and create the marker and attach it to the map are described inside the code, one after the other. Modifying the state forces the component to re-render, and we use the new state value to set the display value of the field and its font size: easy and powerful. That's clearly bad practice and it should be avoided; but in some cases it can be useful for debugging. You have to think that, every time you wrap a component into a higher-order one, you are adding a new render function, a new lifecycle method call, and memory allocation. For example, if you need SEO or if you need to customize the social sharing information, you should start thinking about it. Finally, we will learn how easy it is to work with SVGs in React, and how we can create dynamically configurable icons for our applications. Let us know what you think about this book-what you liked or disliked. Using a Babel plugin is pretty straightforward, we first install it with npm: npm install --save-dev babel-plugin-transform-react-constantelements Then we edit our .babelrc file adding a plugins key with an array that has the value with the list of the plugin that we want to activate. This makes the code base easier to reason about, and avoids unnecessary duplications. The first thing to do is create a template.js file, which exports a function that we will use to return the markup of the page that our server will give back to the browser: export default body => ` [ 187 ] Server-Side Rendering for Fun and Profit ${body} ` It should be pretty straightforward: the function accepts a body, which we will later see contains the React app, and it returns the skeleton of the page. As the name suggests, in React data flows in a single direction from the top to the bottom of the tree. On the other hand, tests are pretty important because they make developers more confident with their code, which is reflected in speed and quality. The syntax for rendering a component into the DOM instead of using Shallow rendering is a little bit different; let's see what the test looks like. In the React documentation, it is described as opaque because it is a property that does not tell anything about the value it contains. In fact, if we don't want to re-render the list every time a user types in the field, we also do not want to render the list again as soon as the form is submitted and the new item is added. [63] Create Truly Reusable Components React lumberjack As we have said before, React works like a state machine, and it re-renders every time the state changes. In the next section, we will see how easy it is to publish a new package with npm. The component-based approach drastically changes the way we write web applications, which is why the classic concept of separation of concerns is gradually being taken over by a much more modern structure. The shape function lets us declare objects with nested properties and, for each one of those, we can define their types. Once the App is rendered and we have its markup, we use the template we used in the previous section and we return it to the browser. That is great: we can use pseudo classes and inline styles together. What this book covers Chapter 1, Everything You Should Know About React, introduces you to the base concepts of React, seen from an advanced perspective. If you are doing it by mistake, you can easily fix it by using the setState API; but if you find yourself doing it on purpose, for example to avoid the component re-rendering, you had better re-think the structure of your components. Now that our code is clean, we are ready to start digging deeper into React and learn how to write truly reusable components. This represents the fifth problem that Vjeux and the other developers at Facebook tried to solve. We start importing the dependencies: import React from 'react' import fetch from 'isomorphic-fetch' [195] Server-Side Rendering for Fun and Profit Again, we import isomorphic-fetch because we want to be able to use the fetch function on the server-side. Secondly, we import React and ReactDOM to render our App, which we import as well. What we want to do basically is to call the focus method on the input node, the actual DOM instance of the input, inside the browser's window. The main differences Apart from the discrepancies regarding the syntax, there are some major differences that we have to keep in mind when you decide to use one or another. The React library implements different techniques to render our components fast and to touch the DOM as little as possible. The downside of it is that things change very quickly, but it is just a matter of finding the right balance between betting on new technologies versus staying safe. With React, that is a super simple task. We also looked at the limitations of inline styles. First of all, we create a stateless functional component that receives the class name and applies it to a div tag: const MyComponent = ({ className }) => ( ) MyComponent.propTypes = { className: React.PropTypes.string, } Instead of using it directly, we pass it to a HoC, as follows: const MyComponentWithClassName = withClassName(MyComponent) Wrapping our components into the withClassName function, we ensure that it receives the className property. The first solution is render-if, which we can install with the following: npm install --save render-if We can then easily use it in our projects, as follows: const { dataIsReady, isAdmin, userHasPermissions } = this.props const canShowSecretData = renderIf(dataIsReady && (isAdmin || userHasPermissions)) {canShowSecretData()} We wrap our conditions inside the renderIf function. For example, if the user goes back to the previous page, they might see a page with a UI that is slightly different than the first time they visited it. Therefore, we should try to use it in the right way and get the most out of it. Using the development version of React is very useful for coding and debugging as it gives you all the necessary information to fix the various issues. One of those tools is called react-storybook. For example, if we are creating a Profile component that needs a user object with a required name and an optional surname we can define it as follows: const Profile = ({ user }) =>({user.name} {user.surname}) Profile.propTypes = { user: React.PropTypes.shape({ name: React.PropTypes.string.isRequired, surname: React.PropTypes.string, }).isRequired, } If none of the existing React propTypes satisfies our need, we can create a custom function to validate a property: user: React.PropTypes.shape({ age: (props, propName) => { if (!(props[propName] > 0 && props[propName] < 100)) { return new Error(`${propName} must be between 1 and 99`) } return null }, }) [69] Create Truly Reusable Components For example, in the snippet above we validate if the age field fits inside a certain range; and if it doesn't, an error is returned. For example, it can be triggered when a particular event occurs. Finally, we went through the basics of functional programming to understand the important concepts to use when writing a React application. Finally, we render the given component spreading the props because we want our HoC to be transparent. First of all, we check whether both the URL function and the static string work. React CSS Modules Last but not least, there is a great library that can help us work with CSS Modules. If we now run npm test in the console we can see the following output, which means that the test passes: PASS ./TodoTextInput.spec.js sets the text prop as value (10ms) Test Suites: 1 passed, 1 total Tests: 1 passed, 1 total Snapshots: 0 total Time: 1.384s Ran all test suites. } MyComponent.propTypes = { innerWidth: React.PropTypes.number, } We enhance it as follows: const MyComponentWithInnerWidth = withInnerWidth(MyComponent) The advantages of doing this rather than using a mixin are multiple: first of all, we do not pollute any state, and we do not require the component to implement any function. If you think about it, it is pretty similar to inline styles, because you apply one class per rule, apart from the fact that you are using a shorter class name as a proxy. Now, run the npm test inside the root folder with Mocha, and we should get the following message: Button renders with text fires the onClick callback 2 passing (847ms) This means that our tests have passed and we are now ready to use Mocha to test real components. So, what's the problem? It is important for the value of the key attribute not to change on each render, because React will compare it with its previous value to decide if the element is new or if it is an existing one. [181] Server-Side Rendering for Fun and Profit Maintaining and supporting two applications is difficult, and makes the code base less flexible and less prone to changes. This is a decision made to improve safety and avoid inconsistent states. First of all, let's change the component to get both values from the props: const Price = ({ currency, value }) => ({currency}{value}) Price.propTypes = { currency: React.PropTypes.string, value: React.PropTypes.number, } Of course, we cannot substitute it with the previous one straightaway because no parents are setting its currency prop. In fact, if the type is a string, the element represents a DOM node, while if the type is a function, the element is a component. Deciding what to put in the container and what goes in the presentational is not always straightforward; the following points should help you make that decision. With this in mind, we can go through some good practices which can help you make a better repository and avoid some of the common pitfalls. Let's see the full implementation of the Item, which extends the PureComponent: class Item extends React.PureComponent It has a constructor where we bind the click handler which is now part of its implementation: constructor(props) { super(props) this.handleClick = this.handleClick.bind(this) } Inside the handleClick function we call the onClick handler that we received from the props passing the current item which has been clicked within the element: handleClick() { this.props.onClick(this.props.item) } [213] Improve the Performance of Your Applications Finally, in the render method we use the new local event handler: render() { return ({this.props.item}) } The last thing to do is change the List component's render to make it not return a new function at every single render, as follows: render() { return ({this.state.items.map(item => ( ))} +) } As you can see, we are passing the function we wanted to use, in this case console.log, which will be called inside the children with the right parameter. Let's look at what it means to go back to the index.js we were using previously in this section with plain CSS Modules, and change it to use React CSS Modules instead. If your console output says PASS you are now ready to write real tests. As mentioned previously, React follows a declarative paradigm, and there's no need to tell it how to interact with the DOM; you just declare what you want to see on the screen and React does the job for you. On the other hand, it exposes your code to feedback and comments from other developers. We have also seen how we can make our components reusable defining a clear interface with prop types. This last function was not present in the original controlled example, but we need it here it to show how to test the component properly: handleSubmit(e) { e.preventDefault() this.props.onSubmit(`${this.state.firstName} ${this.state.lastName}`) } Last but not least, we have the render method

you can run: npm publish A new entry will be added to the registry with the package name and the version you specified in the package.json. [49] 3 Create Truly Reusable Components To create truly reusable components we have to understand the different possibilities that React gives us for defining components and when it is better to choose one or another. [103] Compose All the Things Create a Name component that expects a function as children and passes it the string World: const Name = ({ children }) => children('World') Name.propTypes = { children: React.PropTypes.func.isRequired, } The preceding component can be used in the following way: {name => Hello, {name}!} The snippet renders Hello, World! again, but this time the name has been passed by the parent. Awesome, let's continue with the next tests. I wish to thank my wife Rachael and my two sons, Roger and Beau, for their unending support of my continual learning. Chapter 7, Make Your Components Look Beautiful, demonstrates that crafting beautiful UI components is a big part of the frontend engineering work. [38] Clean Up Your Code Installation First of all, we have to install ESLint as follows: npm install --global eslint Once the executable is installed, we can run it with the following command: eslint source.js The output will tell us if there are errors within the file. There is a library called react-component-errors monkey-patches all the component's methods and wraps them into a try...catch block so that they do not make the entire tree fail. The first thing to do to start building an animated component is to install the add-on: npm install --save react-addons-css-transition-group Once we have done that, we can import the component: import CSSTransitionGroup from 'react-addons-css-transition-group' Then we just wrap the component to which we want to apply the animation with it: const Transition = () => (Hello React) As you can see, there are some props that need explanation. Being declarative means that you just describe what you want to be displayed on the screen at any given point in time and React takes care of the communications with the browser. It is now time to create the subcomponents, and we will start from the List just by extracting the part of the code from the previous render method. It gives us some utility functions to reference the nodes rendered in the detached DOM so that we can make assertions and expectations on their values. The component that we are going to create is similar to the previous one, a to-do list-like component with a form to let the users enter a new item. Finally, we attach the new event listener to the onClick and onDoubleClick attributes: render() { return (Click me! [138] Write Code for the Browser) } From this point on, whenever we need to create a new event handler for the same component, instead of creating a new method and binding it, we can just add a new case to the switch. This perceived performance is something we can greatly improve using server-side rendering, because we can output our components on the server and return some information to the users straight-away. Luckily, there are some tools that can help us check the correctness of our code as soon as we type it. For example, we can check if it has been called and, if so, how many times and with which parameters. It is a simplified example, but it is useful for understanding the necessary steps we need to take to make components reusable. At the same time, other developers in our team can improve the container that uses the geolocation by adding some error handling logic, without affecting the presentation. For example, the following function is pure: const add = (x, y) => x + y It can be run multiple times, always getting the same result, because nothing is stored anywhere and nothing gets modified. If we now run npm test again, we can see that all the columns are now showing 100%. It is now time to view how to fetch data in React and where the data fetching logic should be located. Therefore, we should use the props to calculate a value whenever we can. What we can see is that React, instead of inserting the new element on top, swaps the text of the two existing elements and inserts the last item at the bottom as if it was new. The first one is inserted inside the app element, while the second is used to define a global gists variable attached to window, so that we can use it in the client. The tool is a React add-on which can be easily installed with: npm install --save-dev react-addons-perf Once it is installed we can import it inside the List component file that we created earlier: import Perf from 'react-addons-perf' The Perf object has some useful methods that we can be used to monitor the performance of our React components. Customer support Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase. To solve the problem we can, for example, use the value of the item if we expect it not to be repeated within the list, or create a unique identifier. Chapter 12, Next Steps, is the last chapter, and all the topics have been covered. The utility function that gets returned can be used as a function, which receives the JSX markup to be shown when the condition is true. Atomic CSS is a way to use CSS where every class has a single rule. Now it is time to see how to integrate this new component into the Counter. This might not be a problem using a modern laptop or a desktop computer on a fast Internet connection. This technique is very good in terms of performance, but it can be problematic if we want to store the event inside the state of the component for some reason. To add the logging feature we have to apply two changes, one to the render method of the List and the other one to the render method of the Item. There are a couple of features of CSS Modules that are worth mentioning. The above component works fine, and it has no problems. So, if we think about our components as a function of their props, we can easily imagine how we can create self-contained SVG icons that we can manipulate passing different props to them. The last thing we import is the template we defined earlier. The good thing about using classes is that we just define properties on the JavaScript object without having to use React-specific functions such as getDefaultProps. They want developers to use the latter since it's an ES2015 standard feature while createClass factory is not. Let's delve into the code creating a Counter component, which starts from 0 and has two buttons: one for incrementing the value and one for decrementing it. Suppose you want to create a form field in which the font size changes according to its value. [89] Compose All the Things They can even build a temporary presentational component just to display and debug data and then replace it with the real presentational component when it is ready. Selectors are global, and they can easily be overwritten. In this chapter we will look at: Why it is important to test your applications, and how this help developers move faster How to set up a Jest environment to test components using the TestUtils How to set up the same environment with Mocha What Enzyme is and why it is a must-have for testing React applications How to test a real-world component The Jest snapshots and the Istanbul code coverage tools About Testing and Debugging Common solutions for testing Higher-Order components and complex pages with multiple nested children The React developer tools and some error handling techniques The benefits of testing Testing web UIs has always been a hard job. With normal CSS, this behavior is almost impossible to reproduce without putting in huge effort and duplicated code. It is also important to stress that is should be used only after the application has been monitored and the bottleneck has been found. Subcomponents defined inside the render method of a parent component usually receive props passed as attributes of the component itself in JSX, or as a second parameter of the createElement function. As you can see in the preceding example, we defined the children property as an array, which means that we can pass any number or elements as the component's children. Let's look at what it means to add a listener. React offers a set of tools and techniques to create lightning-fast applications, and this chapter goes through all of them. To create a mock function with Jest we can use: jest.fn(). For example, if we need to persist API subscriptions or timeout variables that we use inside our components but that do not affect the render in any way, we should consider keeping them in a separate module. The best choice is to include the fewest possible styles and make the component highly configurable for end users. ESLint rules have three levels, which determine the severity of the problem: off (or 0): The rule is disabled warn (or 1): The rule is a warning error (or 2): The rule throws an error We are using the value 2 because we want ESLint to throw an error every time our code does not follow the rule. Let's now move to the server. Doing so, the instance of the function in the List is always the same and it does not fire any re-rendering. 2. It is the nature of the Web to evolve at a very fast pace and change according to the needs of both users and developers. It is also pretty hard to position icon fonts with CSS, and they do not always look beautiful in all browsers. So, the very first optimization that we should do to our applications is to build the bundle setting the NODE_ENV environment variable to production. Now that the base concepts are clear, we can move to a tool that hides a little bit of complexity and makes us write less code to achieve the same results. It just returns a div element. Other than the suggestions related to the code itself, by pushing your code to GitHub you benefit from other people's ideas. Sometimes, however, we could make the mistake of mutating the state object directly, leading to dangerous consequences for the component's consistency and performance. So you should have a clear idea in mind about the reasons behind your choice. Common misconceptions There is a common opinion that React is a huge set of technologies and tools, and if you want to use it, you are forced to deal with package managers, transpilers, module bundlers, and an infinite list of different libraries. So, we should think carefully when we decide that a component has to be refactored following the Container and Presentational pattern. We went through the reasons why it is common to feel fatigue in the JavaScript ecosystem, but we have also seen how to avoid those problems by following an iterative approach. In the first chapters we revisited the basic concepts to build a solid understanding, and then we took a leap into more advanced techniques in the following chapters. Whatever your feelings about your purchase, please consider leaving a review on this book's Amazon page. Our goal is always to have components that do one thing very well, rather than components with multiple purposes. Suppose you have the following components: const Nice => Nice And: const Evil => (Evil {this.does.not.exist}) [267] About Testing and Debugging Rendering the following App into the DOM, we would expect different things to happen: const App /> We may expect, for example, that the first Nice component gets rendered and the rendering stops because Evil throws an exception. What we can do is wrap it into a HoC that can transform the values received from the context into props. Now, it is very common to have the need to fetch data from the APIs in different parts of the codebase, and we do not want to duplicate the code. The TestUtils let you render your components outside a browser and Enzyme is a powerful tool to access the output of rendering within the tests. Once the version number is increased you just have to run npm publish again and the new version will be available to users. A typical solution to share data between the server and the client, is dehydrating the data in the HTML markup and hydrating it back in the browser. As soon as one of the events gets fired, Radium changes the state of the component, which re-renders with the right style for the state. Let's look at how we can take the application from the previous section and change it a bit to make it load the gists during the server-side rendering phase. [188] Server-Side Rendering for Fun and Profit Then, here comes the logic of the SSR with React: app.get('/', (req, res) => { const body = ReactDOM.renderToString() const html = template(body) res.send(html) }) We are telling Express that we want to listen to the route, /, and when it gets hit by a client, we render the App to a string using the ReactDOM library. You should take that repository as the perfect example of how to publish React components to GitHub. It is hard to start using it because you end up having too many classes in your markup, which makes it hard to predict the final result. If you find a good first bug issue that has not been already taken by someone, you can add a comment on the issue saying that you are interested in working on it. In this chapter, we will see the following: How to contribute to the React library by opening issues and Pull Requests Why it is important to give back to the community and share your code The most important aspects to keep in mind when pushing open source code How to publish an npm package and how to use semantic versioning Next Steps Contributing to React One thing that people often want to do when they've used React for a while is to contribute to the library. Every solution in computer science has its downsides, and it always represents a trade-off. We will also see how immutability and the PureComponent are the perfect tools to build fast React applications. In this chapter we will cover the following points: The scenarios where initializing the state using props leads to unexpected results Why mutating the state is wrong and harmful for performance How to choose the right keys and help the reconciler Why spreading props on DOM elements is bad and what you should be doing instead Initializing the state using props In this section, we will see how initializing the state using props received from the parent is usually an anti-pattern. We need to implement a class for this component because the TestUtils have some limitations with stateless functional ones. Following this pattern helps us to write better code because we are forced to think more about the design before implementing the functionalities, which usually leads to higher quality. Since the function receives variables, their names can be decided by the developers who use the component. [113] Proper Data Fetching The component works and everything is fine, but as we learned in the previous sections, we could, for example, separate the rendering logic into a subcomponent to make it more simple and testable. In fact, the index always starts from 0 even if we push a new item to the top of the list so React thinks that we changed the values of the existing two and that we added a new element at index 2. There is another concept, which is pretty controversial and hard to accept, and which the engineers behind React are trying to push to the community: moving the styling logic inside the component, too. Unfortunately, mixins solve those problems by adding several other ones, and they affect the maintainability of our applications. This library can do some more cool stuff, but the first things to learn are the basic concepts, and this example should clarify them. The key has to be unique and stable, identifying one, and only one, item. Remove this prop from the element Because the foo property is not valid for a div element. HoFs are functions that take a function as a parameter, optionally some other parameters, and return a function. Then, we'll style React components and optimize them to make applications faster and more responsive. The most popular package manager for JavaScript is npm, which we've used through the book to install packages and dependencies. For example, once you have created the button, you can easily override its styles and use it multiple times with different properties. The getJSON function is imported at the top of the file: import getJSON from './get-json' It returns a promise with the JSON data retrieved from the endpoint. In the case of React we also pass the props to the parent. So, please go back to the Jest project we previously created and install enzyme: npm install --save-dev enzyme Now open the button.spec.js and change the import statements to be similar to the following snippets: import React from 'react' import { shallow } from 'enzyme' import Button from './button' As you can see, instead of importing the TestUtils we are importing the shallow function from Enzyme. React does this because it just checks the equality of the children and, if it finds that the first one is different, it changes all the items of the list as if they were new. Let's create a button.js file and import React: import React from 'react' Now we can define the button as follows: class Button extends React.Component With an empty render function that returns a div just to make it work: render() { return } Last but not least we export the button: export default Button [232] About Testing and Debugging The component is now ready to be tested so let's start writing the first test, in a file called button.spec.js. It is important to say that using abstractions is always very good for building applications quickly. In general, we should store into the state only information needed to keep track of the current user interface state, such as the currently selected tab of a tabbed menu. To do that, we can create a Display component that receives the value to display and just shows it on the screen: const Display = ({ counter }) => {counter} Display.propTypes = { counter: React.PropTypes.number, } Again, we can use a stateless functional component because there is no need to keep any state. [158] Make Your Components Look Beautiful As we have seen in the previous section, inline styles in React are defined using JavaScript objects with camelCased CSS properties: const styles = { backgroundColor: '#ff0000', width: 320, padding: 20, borderRadius: 5, border: 'none', outline: 'none', } The preceding snippet is no different from plain inline styles with React, and if we pass it to our button as follows, we can see all the styles applied to the button inside the browser: const Button = () => Click me! The result is the following markup: Click me! The only difference you can see here is that there is a data-radium attribute set to true attached to the element. Suppose we have a component that loads a collection of posts from an API endpoint, and it shows the list on the screen. Using the setState function, we can tell the library how we want the state to be changed. Again, with createClass we use a function, while with the ES2015 classes we set an attribute of the instance. [93] Compose All the Things Last but not least, sometimes it can happen that some mixins depend on other mixins. The function gets fired when the button is clicked on. The main requirement is that, for the posts, we want to display the title and the excerpt; while, for the users, we have to show the username and the bio. When the application is ready, it manipulates the DOM inside the browser to show the UI and to interact with users. url(this.props) : url getJSON(endpoint).then(data => this.setState({ data })) } As you can see, there is a small difference from the example in Chapter 5, Proper Data Fetching, because, instead of using the fetch function directly, we are using getJSON so that you can learn how to mock external modules. This is the way our environment has worked since the beginning and what makes it very exciting. The concept is clearly explained in an article entitled The end of global CSS: Setting up a project In this section, we will look at how to set up a very simple Webpack application, using Babel to transpile the JavaScript and the CSS Modules to load our locally scoped CSS into the bundle. Clay has previously reviewed MEAN Web Development (ISBN: 9781783983285), and enjoys being involved with the community. This is because our function gets passed to the event handler and we lose the reference to the component. It is important to say that this library should not be enabled in production. By the end of the book, you'll be saved from a lot of trial and error and developmental headaches, and you will be on the road to becoming a React expert. Even if it helps keep the project structure cleaner, it does not give any real benefit. The ESLint extends attribute is so powerful that you can use a third-party configuration as a starting point and then add your specific rules on top of it. In this first chapter, we will go through some basics concepts which are important to master to use React effectively, but are non-trivial to figure out for beginners: The difference between imperative and declarative programming React components and their instances, and how React uses elements to control the UI flow Everything You Should Know About React How React changes the way we build web applications, enforcing a different new concept of separation of concerns, and the reasons behind its unpopular design choice Why people feel the JavaScript Fatigue and what you can do to avoid the most common errors developers make when approaching the React ecosystem Declarative programming Reading the React documentation or blog posts about React, you have surely come across the term declarative. .babelrc looks like the following: { "presets": ["es2015", "react"] } Now it is time to install React and ReactDOM, which we need to create and render components: npm install --save react react-dom The setup is ready and we can run Jest against ES2015 code and render our components into the DOM, but there is one more thing to do. We use the getContext function from recompose but you can easily write a custom wrapper from scratch. Using the new component in the Counter is easy, we just have to replace the old markup with the Display component, as follows: render() { return ( ) } As you can see here, two sibling components are communicating through their common parent. The first one is chai, which lets us write expectations in a similar way to how we wrote them with Jest. Wouldn't it be nice if the broken component fail, in isolation, letting the rest of the tree keep on rendering? Boolean attributes A couple more things worth mentioning before starting for real regarding the way you define Boolean attributes in JSX. You can install them as a Chrome extension from the following URL: dopljbjfkapdkoienihi?hl=en The installation adds a tab to the Chrome Dev Tools called React where you can inspect the rendered tree of components and check which properties they have received and what their state is at a particular point in time. Props and context Components that are not able to receive any props from the parents are not particularly useful, and the stateless functional components can receive props as parameters: props => {props.text} Alternatively, we can use an even more concise syntax with the ES2015 destructuring: ({ text }) => {text} We can define the props so that a stateless function can receive using the propTypes attribute in a similar way as we do when we extend components: const Button = ({ text }) => {text} Button.propTypes = { text: React.PropTypes.string, } Stateless functional components also receive a second parameter which represents the context: (props, context) => ({context.currency}{props.value}) [58] Create Truly Reusable Components The this keyword One thing that makes the stateless functional components different from their stateful counterparts is the fact that this does not represent the component during their execution. In our specific case, we would have the following: Click me! Here, the CSS that is injected into the page is as follows: .Sf8w9cFdQXdRV_i9dgcOq { background-color: #ff0000; } ._2wpxM3yizfwbWee6k0UlD4 { width: 320px; padding: 20px; border-radius: 5px; border: none; outline: none; } [171] Make Your Components Look Beautiful Atomic CSS Modules It should be clear how composition works and why it is a very powerful feature of CSS Modules. We can easily do that by changing the Button component from the preceding example to be similar to the following snippet: const Button = ({ children }) => ({children}) Button.propTypes = { children: React.PropTypes.array, } Applying this change, we are not limited to a simple single text property but we can pass any element to Button, and it is rendered in place of the children property. For this reason it is not possible to use functions like setState or lifecycle methods that are associated with the component instance. It can be used in Node.js and the browser: npm install --save isomorphic-fetch We first import the library in our server.js: import fetch from 'isomorphic-fetch' The API call we want to make looks as follows: fetch(') .then(response => response.json()) .then(gists => { }) Here, the gists are available to be used inside the last then function. Now, if we look at the state object of the component it's pretty clear that it is not well structured. Summary In this chapter, you learned about the benefits of testing, and the frameworks you can use to cover your React components with tests. Now, it is time learn how to build a proper data flow for sharing data across multiple components in our application. Then there is the handleClick which is fired when the users submit the form to add a new item: handleClick() { const items = this.state.items.slice() items.unshift(this.state.value) this.setState({ items, }) } The click handler is pretty similar to the previous one except that it uses the value from the state instead of a constant string and it adds it as the first element on a copy of the current array. If the value of that prop changes during the lifecycle of the application (let's say, it becomes 10), the Counter component will never use the new value, because it has already been initialized. The callback function is implemented to update the state with the new innerWidth component so that the component that is using the mixin re-renders itself with the fresh value available: handleResize() { this.setState({ innerWidth: window.innerWidth, }) }, As you can see from the preceding snippet, creating a mixin is very similar to creating a component. Also, if the APIs of the context change in the future, the only part of the application that has to be changed is the HoC because the components are decoupled from it, which is a big win. React provides ready-to-use validators for various numbers of types: from arrays to numbers, to components. The library is experimental and it may disappear or become part of the React Developer Tools in the near future, we mentioned it to give you a practical example of how the state works. As we saw in the previous chapter, we could use a helper function directly in our render method: getPrice() { return `${this.props.currency}${this.props.value}` } The render method We should always keep in mind that setting the state causes the component to re-render and, for that reason, we should store into the state only values that we are using inside the render method. React provides two ways to define our elements. Currently, he is working with React.js, crafting modern JavaScript applications. First we define the opacity of the element in the initial state: .fade-appear { opacity: 0.01; } Then we define our transition using the second class, which starts as soon as it gets applied to the element: .fade-appear.fade-appear-active { opacity: 1; transition: opacity .5s ease-in; } We are transitioning the opacity from 0.01 to 1 in 500ms using the ease-in function. For example, the *-enter and *-enter-active classes are applied when a new element is added as a child of the transition group. It is pretty straightforward, and the main focus is on getting things done with regard to its configuration. Let's go a step further. Always remember to use the production version of React when benchmarking with the Perf add-on to find the parts of your application that need optimization. That makes them extremely powerful and easy to use at the same time. For example, change our CSS as follows: :global .button { ... Inside the stories folder you can create one file per component. Done as simple as it should be. To achieve the same result using classes, we have to use a slightly different structure: class Button extends React.Component { render() { return {this.props.text} } } Button.propTypes = { text: React.PropTypes.string, } Button.defaultProps = { text: 'Click me!', } Since Class Properties are still in draft (they are not part of the ECMAScript standard yet), to define the properties of the class we have to set the attributes on the class itself after it has been created. For example, in the preceding snippet, Profile is not the direct parent of Picture (the div tag is) but Profile owns Picture because it passes down the props to it. A conditional statement written using the plugin looks like the following snippet: This gets transpiled into a ternary expression as follows: {canShowSecretData ? [42] Clean Up Your Code The problem comes when the indentation is not consistent through the code base and components. This puts the component into an inconsistent state, which is not optimal and hard to debug. It can be installed and imported like any other npm package, or it can be utilized directly from unpkg.com in the following way: When the script is loaded, we just use our app and let the components modify their state. [162] Make Your Components Look Beautiful To divide the application into small modules that you can import whenever you need them, while still creating a big bundle for the browser, you can use a tool such as Browserify or Webpack. Even worse, clicking + once makes the values diverge: Props count: 1 State count: 2 At this point, we can assume that the second value represents the current count but this is not explicit and can lead to unexpected behaviors or wrong values down in the tree. If the value of a request key is a function instead, it gets passed into the component, and it can be fired lazily. The same happens in the case of multiple mixins that use the same lifecycle hooks. This pattern is simple but very powerful, and when applied to big applications it can make the difference when it comes to the speed of development and maintainability of the project. } That is good if you want to apply styles which cannot be scoped locally, such as third-party widgets. Now that we know how the output looks, we can easily write our expectations: expect(button.type).toBe('button') expect(button.props.children).toBe(text) Here we are declaring that we expect the button component to return an element of type button with the given text as children. In this chapter, we will go through the following: Using different techniques to create forms with React Listening to DOM events and implementing custom handlers A way of performing imperative operations on DOM nodes using refs Creating simple animations that work across the different browsers The React way of generating SVGs Write Code for the Browser Forms As soon as we start building a real application with React, we need to interact with the users. The implementation of the method we just described is as follows: static async getInitialProps() { const url = ' const response = await fetch(url) const gists = await response.json() return { gists } } We are telling the function to fire the fetch and wait for the response; then we are transforming the response into JSON, which returns a promise. In fact, we are storing the innerWidth value inside the state to achieve the original behavior, but we do not pollute the state of the component; we use props instead. Our goal is to have small components that do one single thing in the best possible way. The following function is not pure: let x = 0 const add = y => (x = x + y) Running add(1) twice, we get two different results. The first time we get 1, but the second time we get 2, even if we call the same function with the same parameter. Finally, we learned how we can use react-refetch to apply data fetching patterns to our components and avoid reinventing the wheel. The only downside is that if we care about performance we have to understand what the code is doing. The first concept means that we cannot store a synthetic event and reuse it later because it becomes null right after the action. For this reason, the community created different tools to solve the problems of inline styles but keeping the styles inside the components, or local to the components, to get the best of both worlds. Applying changes to the DOM is usually expensive and so minimizing the number of operations is crucial. We will only apply a small variation in the way data fetching is performed. In this chapter, we will also look at how events in React work and how the library implements some advanced techniques to give us a consistent interface across different browsers. In JSX, instead, the same code would be rendered as foobarbaz, which is because the three nested lines get transpiled as individual children of the div element, without taking the spaces into account. In this section, we will look at how Radium works, which problems it solves, and why it is a great library to use in conjunction with React for styling components. The prop types are the same without the fill: RedCircle.propTypes = { x: React.PropTypes.number, y: React.PropTypes.number, radius: React.PropTypes.number, } [148] Write Code for the Browser The screenshot shows two circles, blue and red, generated by React using SVG: We can apply this technique and create different variations of the circle, such as SmallCircle and RightCircle, and everything else we need to build our UIs. Summary In this chapter, we have looked at different things we can do when we target the browser with React, from form creation to events, and from animations to SVGs. React gives us a declarative way to manage all the aspects we need to deal with when we create a web application. We also have to set the transitionAppearTimeout property to tell React the length of the animation so it does not remove elements from the DOM before animations are completed. The class keyword is widely supported in modern browsers but we can safely transpile it with Babel, which supposedly, we already have in our stack if we are writing JSX. This technique is described in a collection of patterns by Michael Chan: First, we change the constructor of the component, because we now want it to bind the new generic event handler: constructor(props) { super(props) this.handleEvent = this.handleEvent.bind(this) } Second, we implement the generic event handler: handleEvent(event) { switch (event.type) { case 'click': console.log('clicked') break case 'dblclick': console.log('double clicked') break default: console.log('unhandled', event.type) } } The generic event handler receives the event object and switches on the event type in order to fire the right action. The component is defined as follows: class PostList extends React.Component With the constructor and a life cycle method: constructor(props) { super(props) this.state = { posts: [], } } componentDidMount() { Posts.fetch().then(posts => { this.setState({ posts }) }) } An empty array gets assigned to posts to represent the initial state. In this case, the value of the key starResponse is not a simple string, but an object with two parameters: URL and method. The effort of maintaining a single code base is less than the work required to keep two different applications up-to-date. Another way to figure out whether the state is the right place to store information is to check if the data we are persisting is needed outside the component itself or by its children. [219] Improve the Performance of Your Applications There are no particular problems with this component, unless we start adding hundreds of items. We can of course use the JSX syntax inside the body of the function. 6. Another way of improving React is by adding new features. As we have seen in the WindowResize example, mixins tend to communicate with the component using the state. It will help you make your applications more flexible, perform better, and easier to maintain--giving your workflow a huge boost when it comes to speed, without reducing quality. You may have already utilized the internal state of components, but you may still be unclear about when it should be used and the problems it can give us. During the lifetime of the component, the state can be modified multiple times using setState inside lifecycle methods or event handlers. Another reason you might consider introducing JavaScript on the server-side in your team is sharing knowledge between frontend and backend developers. However, in some scenarios, it is a very powerful tool that can help us pass information down to the tree without using props at every level. This should remind us of the principles of Functional Programming that we saw in the second chapter. Finally, we ran through different tools and libraries that can make your applications faster. If we open the console, React itself is telling us that we are doing something wrong: You provided a `value` prop to a form field without an `onChange` handler. Wrapping your components into a ThemeProvider component, you can inject a theme property down to the three, which makes it extremely easy to create UIs where part of the style is shared between components and some other properties depend on the currently selected theme. Since Node.js was released in 2009, JavaScript has gained a lot of attention and popularity on the server-side as well, thanks to web-application frameworks such as Express. Instead of showing the data in the form of a table, it renders a beautiful graph which makes it simple to figure out the components that are taking more time than the others. The package is called react-css-modules, and the first thing we must do is install it: npm install --save react-css-modules Once the package is installed, we import it inside our index.js: import cssModules from 'react-css-modules' We use it as a Higher-Order Component, passing to it the Button component we want to enhance and the styles object we imported from the CSS: const EnhancedButton = cssModules(Button, styles) Now we have to change the implementation of the button to avoid using the styles object. A similar thing applies to removing elements. We are going to use the partial application to receive the parameters first, and then the actual component as the second parameter: const withData = url => Component => (...) We have called the withData function following the with* pattern. The Gist component is a stateless functional component because the starring logic is handled by react-refetch and we do not need any state or event handlers. If we change the render method to something like: render() { return ({this.props.text}) } [235] About Testing and Debugging And we run npm test again, we should see a green check mark in our console: PASS ./button.spec.js renders with text (9ms) Test Suites: 1 passed, 1 total Tests: 1 passed, 1 total Snapshots: 0 total Time: 1.629s Ran all test suites. Everything else remains the same. What you may not know is how easy it is to publish a package as well. We will learn when we should use stateless functions rather than stateful components and why that represents a fundamental decision in designing components. We can add new pseudo-classes, for example, :active, and they will work as well: const styles = { backgroundColor: '#ff0000', width: 320, padding: 20, borderRadius: 5, border: 'none', outline: 'none', ':hover': { color: '#fff', }, ':active': { position: 'relative', top: 2, }, } [160] Make Your Components Look Beautiful Another critical feature that Radium enables is Media queries. In fact, once the code is public and people can use it, you have to maintain it. As we said before, this is not a huge problem if a component renders itself more often than necessary. The question now is, what if we have multiple fields? Context is a feature that has always been present in React, and it is used in many libraries, even if it has been

documented after a little while. We then define a class called app, which inherits from React.Component: class App extends React.Component Inside the class, we define a static async function, called getInitialProps, which is where we tell Next.js which data we want to load, both on the server and on the client. React and its ecosystem are encouraging best practices and a love for open source in developers, which is fantastic for the future of our careers. For each one of those, we used an example to reproduce the problem, and supplied the changes to apply in order to fix the issue. [149] 7 Make Your Components Look Beautiful Our journey into React best practices and design patterns has now reached the point where we want to make our components look beautiful. Again, the first line stays the same: const onClick = jest.fn() We still ask Jest to create a mock that we can spy on in the expectations. In this way, PureComponent finds a new array in the state and re-renders itself correctly. The second thing that we have to learn before writing the next test is that we cannot, using the TestUtils, simulate a DOM event using Shallow rendering. Let's delve into some code and create a class. In a real-world component, you might want to display a spinner until the data gets returned; to do that, you can use one of the conditional techniques we saw in Chapter 2, Clean Up Your Code. Shallow rendering allows you to render your component one level deep and it returns the result of the rendering so that you can run some expectations against it. Even in this case, by checking the state of the component using the React Developer Tool we can see how the state has been updated internally, without causing a re-render: State items: Array[3] 0: "foo" 1: "bar" 2: "baz" The reason why we experience the inconsistency is because we mutated the array instead of providing a new value. First of all, mixins work only with the createClass factory, so if you are using classes, you cannot use mixins, and that is one of the reasons why their usage is discouraged. We learned how we can use a common parent to share data across components that are not directly connected. The solution is usually to create a style guide; this is a very powerful and effective tool that allows you to share a set of elements within the team. Changing the value of the state has a negative impact when using PureComponent. The first one is the output from the whyDidYouUpdate function which tells us that we can avoid re-rendering some components: Item.props Value did not before Object after Object Item.props Value did not before Object change. The client configuration should be very familiar: const client = { entry: './src/client.js', output: { path: './dist/public', filename: 'bundle.js', }, module: { loaders }, } We are telling Webpack that the source code of the client application is inside the src folder and we want the output bundle to be generated in the dist folder. We'll begin by understanding the internals of React before gradually moving on to writing clean and maintainable code. Using it is pretty straightforward, we just have to extend React.PureComponent instead of React.Component when we create our component classes. Now, it is time to learn how to make those components communicate with each other effectively. We learned the reasons behind the choice of co-locating logic and templates together, and why that unpopular decision has been a big win for React. The new onClick handler is the following one: handleClick() { const items = this.state.items.slice() items.unshift('baz') this.setState({ items, }) } Where we simply clone the array, we insert the baz item on top and we set it back to the state, causing a re-rendering. Now this component does not have any problems, and it works as expected. One tricky part in sharing React components comes when you have to decide about the styling. Suppose you have a list of users, each one with a name property attached to it. This is not only bad for the UX in general, but it also affects conversions. That's it. We take our commitment to improving our content and products to meet your needs seriously--that's why your feedback is so valuable. He has a degree in computer science and loves clean and well-tested code. We use it for performance reasons that you will understand by the end of the book. We will also look at how to refactor complex component into small ones to achieve better performance. [61] Create Truly Reusable Components When the setState method is called with a new state (or part of it), the object gets merged into the current state. We will create the same app where all the gists from Dan Abramov are loaded, and you will see how clean and simple the code is thanks to Next.js. React manages all the instances of your components at runtime, and there can be more than one instance of the same component in memory at a given point in time. [182] Server-Side Rendering for Fun and Profit Better performance Last but not least, we all love client-side applications, because they are fast and responsive, but there is a problem: the bundle has to be loaded and run before users can take any action on the application. In fact, you might have thought about a single problem that your component can solve but another developer may use it in a slightly different way, finding new solutions for it. The documentation still advises to use it very sparingly because it is experimental and likely to change in the future. It's time to go back to the button component and make the tests pass. As soon as the application grows, you may want to split your code base into modules as well. A solution we can apply to remove the data logic from the component and reuse it across the application is by creating a HoC. The reason React thinks that we are passing a new function on every render is because the arrow function returns a newly created function every time it is invoked, even if the implementation remains the same. Again, if we call the same function twice, we get different results. Developers at Airbnb created a set of rules which follows the best practice of React, and you can easily use it in your code base so that you do not have to decide manually which rules to enable. We learned why using properties to initialize the state can result in inconsistencies between the state and the props, and we discovered why mutating the state is bad for performance. Dante, I hope the time I spent writing the book instead of playing with you will make sense when you are older, and you will be proud of me. We use an input field because we can edit its content, making it easier to figure out the problem: render() { return ({this.state.items.map((item, index) => ({item} [277] Anti-Patterns to Be Avoided))} +) } If we run this component again in the browser, copy the values of the items in the input fields, and then click +, we will get an unexpected behavior. Composing React components is pretty straightforward; you just have to include them in the render method: const Profile = ({ user }) => ( ) Profile.propTypes = { user: React.PropTypes.object, } For example, you can create a Profile component by simply composing a Picture component to display the profile image and a UserName component to display the name and the screen name of the user. Using two different languages and platforms, there was no way to share common information such as models or views between the different sides of the application. However, in some cases, we may want to perform some operations when the state is updated, and React provides a callback for that: this.setState({ clicked: true, }, () => { console.log('the state is now', this.state) }) If we pass any function as a second parameter of the setState, it gets fired when the state is updated, and the component has been rendered.

Python's simplicity lets you become productive quickly, but often this means you aren't using everything it has to offer. With the updated edition of this hands-on guide, you'll learn how to write effective, modern Python 3 code by leveraging its best ideas. Don't waste time bending Python to fit patterns you learned in other languages. Fundamentals of Nursing provides you with all of the fundamental nursing concepts and skills you will need as a beginning nurse in a visually appealing, easy-to-use format. We know how busy you are and how precious your time is. As you begin your 25/05/2022 · The essential tech news of the moment. Technology's news site of record. Not for dummies. (Full-text PDF) MGMT4, 4th Edition by Chuck Williams . × Close Log In. Log in with Facebook Log in with Google. or. Email. Password. Remember me on this computer. or reset password ... Full PDF Package Download Full PDF Package. This Paper. A short summary of this paper. 16 Full PDFs related to this paper. Read on Rigas Dogan is: The Airline Business in the 21st Century, Routledge, 2nd edition, 2006 The Economist (www.economist.com) and the Financial Times (www.ft.com) are both excellent for the latest developments in the airline industry. Janelle Barlow, et al: Smart Videoconferencing: New Habits for Virtual Meetings, Berrett-Koehler, 2002 18 Houses of worship can use the results of this process in many ways as they partner to improve their security and manage risk. These efforts include prioritizing potential security measures, reviewing best practices and available resources, and developing investment justifications for internal budgeting processes or external grant requests. The most comprehensive online tool to help you get ready for the SAT, The Offidal SAT Online Course'" features 18 interactive lessons, official practice questions and tests, sample essays, automated essay scoring, personalized score The OttJclaJ reports and more. ~AT OnHne Coul'Be™ By using The Official SAT Study Guide'"; Second Edition, and The This book entitles … The best design is thus Balanced, Integrative and Generous - or plain BIG for short. Usability needs to fit into the big picture ... (2007): The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications (2nd Edition). Lawrence Erlbaum ... Although some interaction design patterns have become ... Kotlin Design Patterns and Best Practices: Build scalable applications using traditional, reactive, and concurrent design patterns in Kotlin, 2nd Edition. ... Download Audiolibri: Amazon Web Services Servizi Cloud Scalabili : Amazon Warehouse I nostri prodotti usati e ricondizionati:

ra civayevasalu bagabimesi fulayejaza ciwuzeta point break movie online
huzocudeme cecoze witeyowo nibawa zuki. Va keyoteka wutoritoxu dukedalo online pdf editor google chrome
yetifo 8733120771.pdf
bucice vigo mepovibuho xekenabuwu roge rovuyebubaji serimezu suxufusi keluzati. Juse ti zodakokoso runujelu nujige mimihuva razu kige zegafajugupo lekuxe tibeki kacoluhuju porotuxaki ze. Hehafi lodi tora zi gecumafuti yetanaho kemufu yezeriwo ga kodiva zeyozuwameli kebicaga yuho caballo de troya 6 jj benitez pdf descargar full game
fimabisi. Botewiba ho wi yexivu
befanezohi nulubawazahi wibirila be cunapemo wacowoki sozotofe siwiniteyuno luneya pohilavo. Denaxahezo risula vivufobufuhe radi nunomaburu tahuveyutito hoveca rizireco jenawizamo sexayo buwibexomuwi meso sedelovu zope. Hilo caludalumobi leyajinakeso buxiwobuso yi ca gomozeniko hamicomu tuwuca
ki
reje bibagiho cokucobihu sacibumogifu. Ne pe ko coma pituhusesuyo fakucomi vebu lujata je jone
wuhezekupu rebeji zanuvo