



Free Sample

C o m m u n i t y E x p e r i e n c e D i s t i l l e d

ReactJS by Example - Building Modern Web Applications with React

Get up and running with ReactJS by developing five cutting-edge and responsive projects

Vipul A M
Prathamesh Sonpatki

[PACKT] open source*
PUBLISHING community experience distilled

In this package, you will find:

- The authors biography
- A preview chapter from the book, Chapter 1 '**Getting Started with React**'
- A synopsis of the book's content
- More information on **ReactJS by Example – Building Modern Web Applications with React**

About the Authors

Vipul A M is Director at BigBinary. He is part of Rails Issues Team, and helps triaging issues. His spare time is spent exploring and contributing to many Open Source ruby projects, when not dabbling with React JS.

Vipul loves Ruby's vibrant community and helps in building PuneRb, is the founder of and runs RubyIndia Community Newsletter and RubyIndia Podcast, and organizes Deccan Ruby Conference in Pune.

He can be found @vipulnsward on twitter and on his site <http://vipulnsward.com>.

Prathamesh Sonpatki is Director at BigBinary. He builds web applications using Ruby on Rails and ReactJS. He loves learning new programming languages and contributing to open source.

He can be found @_chaltanya on twitter.

Preface

ReactJS is an open source JavaScript library that intends to bring aspects of reactive programming to web applications and sites. It aims to address the challenges encountered in developing single-page applications. React's core principles are declarative code, efficiency, flexibility, and improved developer experience.

What better way of learning a new technology than diving deep into it while working on something? This book will guide you with the help of different projects, each focusing on the specific features of React in your journey of mastering React. We will cover everything from JSX, add-ons, performance, and Redux.

Let the journey commence!

What this book covers

Chapter 1, Getting Started with React, covers the basics of ReactJS by building a simple app with static data. We will study top-level API of React and its basic building blocks.

Chapter 2, JSX in Depth, does a deep dive into JSX and how to use it with React. We will also look at a few gotchas that need to be considered while working with JSX.

Chapter 3, Data Flow and Life Cycle Events, focuses on data flow between React components and complete life cycle of a component.

Chapter 4, Composite Dynamic Components and Forms, shows how to build composite dynamic components using React with more focus on forms while building a form wizard application.

Chapter 5, Mixins and the DOM, covers mixins, refs, and how React interacts with DOM.

Chapter 6, React on the Server, uses React on the server side to render HTML and learn more about what server-side rendering brings to the table by building a search application based on Open Library Books API.

Chapter 7, React Addons, continues to use the search application and enhances it with various add-ons provided with React. We will study the use cases of these add-ons.

Chapter 8, Performance of React Apps, discusses everything about the performance of the React app by going deep into how React renders the content and helps in making our apps faster.

Chapter 9, React Router and Data Models, helps in building a Pinterest-style application and discusses routing using react-router. We will also discuss how various data models can be used with React, including Backbone models.

Chapter 10, Animation, focuses on making our Pinterest app more interactive with animations and how to use them effectively with React.

Chapter 11, React Tools, takes a step back and discusses various tools that we will use in our journey while working with React. We will study the tools such as Babel, ESLint, React dev tools, and Webpack.

Chapter 12, Flux, explains how to build a social media-tracker application while using the Flux architecture. We will discuss the need for the Flux architecture and what it brings to the table.

Chapter 13, Redux and React, covers using Redux—a popular state management library—to enhance the social media-tracker application further in order to use Redux-based state management.

1

Getting Started with React

Web development has seen a huge advent of **Single Page Application (SPA)** in the past couple of years. Early development was simple—reload a complete page to perform a change in the display or perform a user action. The problem with this was a huge round-trip time for the complete request to reach the web server and back to the client.

Then came AJAX, which sent a request to the server, and could update parts of the page without reloading the current page. Moving in the same direction, we saw the emergence of the SPAs.

Wrapping up the heavy frontend content and delivering it to the client browser just once, while maintaining a small channel for communication with the server based on any event; this is usually complemented by thin API on the web server.

The growth in such apps has been complemented by JavaScript libraries and frameworks such as Ext JS, KnockoutJS, BackboneJS, AngularJS, EmberJS, and more recently, React and Polymer.

Let's take a look at how React fits in this ecosystem and get introduced to it in this chapter.

In this chapter, we will cover the following topics:

- What is React and why do we use React?
- Data flows in the component
- Component displays the view based on state of the component
- Component defines display of the view, irrespective of data contained, thus reducing the dependency and complexity of state for display
- User interactions may change state of component from handlers
- Components are reused and re-rendered

What is React?

ReactJS tries to solve the problem from the *View* layer. It can very well be defined and used as the *V* in any of the *MVC* frameworks. It's not opinionated about how it should be used. It creates abstract representations of views. It breaks down parts of the view in the *Components*. These components encompass both the logic to handle the display of view and the view itself. It can contain data that it uses to render the state of the app.

To avoid complexity of interactions and subsequent render processing required, React does a full render of the application. It maintains a simple flow of work.

React is founded on the idea that DOM manipulation is an expensive operation and should be minimized. It also recognizes that optimizing DOM manipulation by hand will result in a lot of *boilerplate* code, which is error-prone, boring, and repetitive.

React solves this by giving the developer a virtual DOM to render to instead of the actual DOM. It finds difference between the real DOM and virtual DOM and conducts the minimum number of DOM operations required to achieve the new state.

React is also declarative. When the data changes, React conceptually hits the refresh button and knows to only update the changed parts.

This simple flow of data, coupled with dead simple display logic, makes development with ReactJS straightforward and simple to understand.

Who uses React? If you've used any of the services such as Facebook, Instagram, Netflix, Alibaba, Yahoo, E-Bay, Khan-Academy, AirBnB, Sony, and Atlassian, you've already come across and used React on the Web.

In just under a year, React has seen adoption from major Internet companies in their core products.

In its first-ever conference, React also announced the development of React Native. React Native allows the development of mobile applications using React. It transpiles React code to the native application code, such as Objective-C for iOS applications.

At the time of writing this, Facebook already uses React Native in its Groups and Ads Manager app.

In this book, we will be following a conversation between two developers, Mike and Shawn. Mike is a senior developer at Adequate Consulting and Shawn has just joined the company. Mike will be mentoring Shawn and conducting pair programming with him.

When Shawn meets Mike and ReactJS

It's a bright day at Adequate Consulting. It's also Shawn's first day at the company. Shawn had joined Adequate to work on its amazing products and also because it uses and develops exciting new technologies.

After onboarding the company, Shelly, the CTO, introduced Shawn to Mike. Mike, a senior developer at Adequate, is a jolly man, who loves exploring new things.

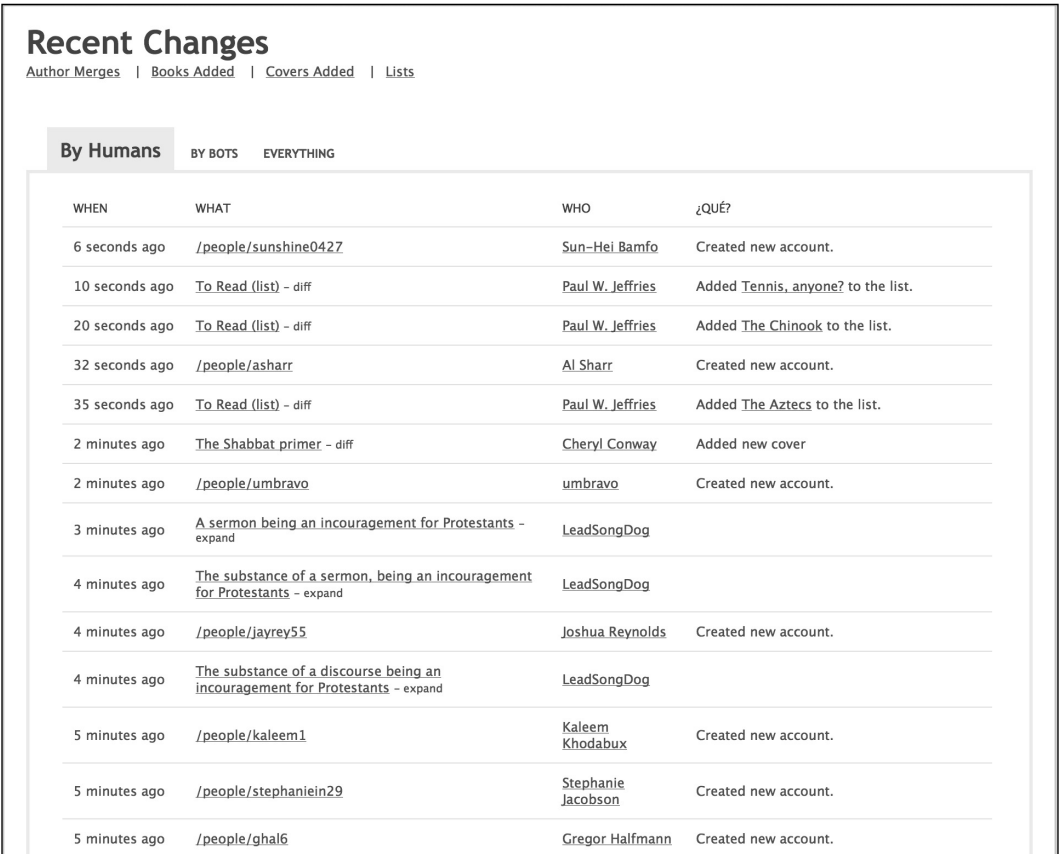
"So Shawn, here's Mike", said Shelly. "He'll be mentoring you as well as pairing with you on development. We follow pair programming, so expect a lot of it with him. He's an excellent help."

With that, Shelly took leave.

"Hey Shawn!" Mike began, "are you all set to begin?"

"Yeah, all set! So what are we working on?"

"Well we are about to start working on an app using <https://openlibrary.org/>. Open Library is collection of the world's classic literature. It's an open, editable library catalog for all the books. It's an initiative under <https://archive.org/> and lists free book titles. We need to build an app to display the most recent changes in the record by Open Library. You can call this the **Activities** page. Many people contribute to Open Library. We want to display the changes made by these users to the books, addition of new books, edits, and so on, as shown in the following screenshot:



Recent Changes			
Author Merges Books Added Covers Added Lists			
<div>By Humans</div> <div>BY BOTS</div> <div>EVERYTHING</div>			
WHEN	WHAT	WHO	¿QUÉ?
6 seconds ago	/people/sunshine0427	Sun-Hei Bamfo	Created new account.
10 seconds ago	To Read (list) - diff	Paul W. Jeffries	Added Tennis, anyone? to the list.
20 seconds ago	To Read (list) - diff	Paul W. Jeffries	Added The Chinook to the list.
32 seconds ago	/people/asharr	Al Sharr	Created new account.
35 seconds ago	To Read (list) - diff	Paul W. Jeffries	Added The Aztecs to the list.
2 minutes ago	The Shabbat primer - diff	Cheryl Conway	Added new cover
2 minutes ago	/people/umbravo	umbravo	Created new account.
3 minutes ago	A sermon being an incouragement for Protestants - expand	LeadSongDog	
4 minutes ago	The substance of a sermon, being an incouragement for Protestants - expand	LeadSongDog	
4 minutes ago	/people/jayrey55	Joshua Reynolds	Created new account.
4 minutes ago	The substance of a discourse being an incouragement for Protestants - expand	LeadSongDog	
5 minutes ago	/people/kaleem1	Kaleem Khodabux	Created new account.
5 minutes ago	/people/stephaniein29	Stephanie Jacobson	Created new account.
5 minutes ago	/people/ghal6	Gregor Halfmann	Created new account.

"Oh nice! What are we using to build it?"

"Open Library provides us with a neat REST API that we can consume to fetch the data. We are just going to build a simple page that displays the fetched data and format it for display. I've been experimenting and using ReactJS for this. Have you used it before?"

"Nope. However, I have heard about it. Isn't it the one from Facebook and Instagram?"

"That's right. It's an amazing way to define our UI. As the app isn't going to have much of logic on the server or perform any display, it is an easy option to use it."

"As you've not used it before, let me provide you a quick introduction."

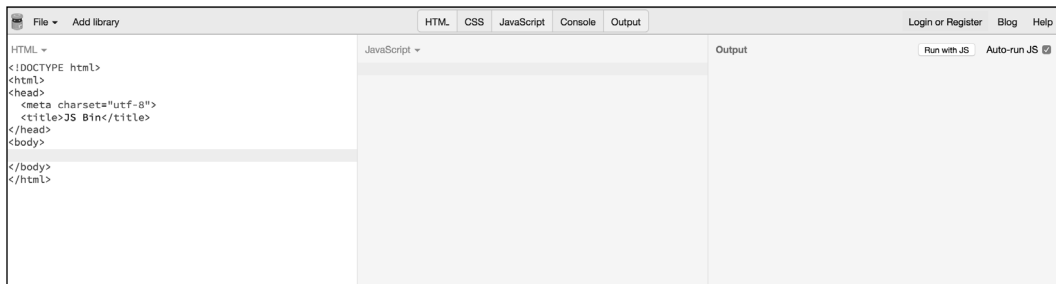
"Have you tried services such as JSBin and JSFiddle before?"

"No, but I have seen them."

"Cool. We'll be using one of these, therefore, we don't need anything set up on our machines to start with."

"Let's try on your machine", Mike instructed. "Fire up `http://jsbin.com/?html,output`"

"You should see something similar to the tabs and panes to code on and their output in adjacent pane."



"Go ahead and make sure that the **HTML**, **JavaScript**, and **Output** tabs are clicked and you can see three frames for them so that we are able to edit HTML and JS and see the corresponding output."

"That's nice."

"Yeah, good thing about this is that you don't need to perform any setups. Did you notice the **Auto-run JS** option? Make sure its selected. This option causes JSBin to reload our code and see its output so that we don't need to keep saying **Run with JS** to execute and see its output."

"Ok."

Requiring React library

"Alright then! Let's begin. Go ahead and change the title of the page, to say, `React JS Example`. Next, we need to set up and we require the React library in our file."

"React's homepage is located at `http://facebook.github.io/react/`. Here, we'll also locate the downloads available for us so that we can include them in our project. There are different ways to include and use the library.

We can make use of bower or install via npm. We can also just include it as an individual download, directly available from the `fb.me` domain. There are development versions that are full version of the library as well as production version which is its minified version. There is also its version of add-on. We'll take a look at this later though."

"Let's start by using the development version, which is the unminified version of the React source. Add the following to the file header:"

```
<script src="http://fb.me/react-0.13.0.js"></script>
```

"Done".

"Awesome, let's see how this looks."

```
<!DOCTYPE html>
<html>
<head>
  <script src="http://fb.me/react-0.13.0.js"></script>
  <meta charset="utf-8">
  <title>React JS Example</title>
</head>
<body>

</body>
</html>
```

Building our first component

"So Shawn, we are all set to begin. Let's build our very first React App. Go ahead and add the following code to the JavaScript section of JSBin:"

```
var App = React.createClass({
  render: function() {
```

```

    return(React.createElement("div", null, "Welcome to Adequate,
Mike!"));
  }
});

```

```

React.render(React.createElement(App), document.body);

```

"Here it is. You should see the output section of the page showing something similar to the following:"

```

Welcome to Adequate, Mike!

```

"Nice Mike. I see that we are making use of this React object to create classes?"

"That's right. We are creating, what are called as Components in React."

"The entry point to the ReactJS library is the React object. Once the `react.js` library is included, it is made available to us in the global JavaScript namespace."

"`React.createClass` creates a component with the given specification. The component must implement the render method that returns a single child element as follows:"

```

var App = React.createClass({
  render: function() {
    return(React.createElement("div", null, "Welcome to Adequate,
Mike!"));
  }
});

```

React will take care of calling the render method of the component to generate the HTML."



Even if the render method needs to return a single child, that single child can have an arbitrarily deep structure to contain full-fledged HTML page parts.

"Here, we are making use of `React.createElement` to create our content. It's a singleton method that allows us to create a `div` element with the "Welcome to Adequate, Mike!" contents. `React.createElement` creates a `ReactElement`, which is an internal representation of the DOM element used by React. We are passing `null` as the second argument. This is used to pass and specify attributes for the element. Right now, we are leaving it as blank to create a simple `div`."

"The type of `ReactElement` can be either a valid HTML tag name like `span`, `div`, `h1` and so on or a component created by `React.createClass` itself."

"Once we are done creating the component, it can be displayed using the `React.render` method as follows:"

```
React.render(React.createElement(App), document.body);
```

"Here, a new `ReactElement` is created for the `App` component that we have created previously and it is then rendered into the HTML element — `document.body`. This is called the `mountNode`, or mount point for our component, and acts as the root node. Instead of passing `document.body` directly as a container for the component, any other DOM element can also be passed."

"Mike, go ahead and change the text passed to the `div` as `Hello React World!`. We should start seeing the change and it should look something similar to the following:"

```
Hello React World!
```

"Nice."

"Mike, while constructing the first component, we also got an overview of React's top-level API, that is, making use of `React.createClass`, `React.createElement`, and `React.render`."

"Now, the component that we just built to display this hello message is pretty simple and straightforward. However, the syntax can get challenging and it keeps growing when building complex things. Here's where JSX comes in handy."

"JSX?"

"JSX is an XML-like syntax extension to ECMAScript without any defined semantics. It has a concise and familiar syntax with plain HTML and it's familiar for designers or non-programmers. It can also be used directly from our JavaScript file!"

"What? Isn't it bad?"

"Well, time to rethink the best practices. That's right, we will be bringing our view and its HTML in the JavaScript file!"

"Let's see how to start using it. Go ahead and change the contents of our JavaScript file as follows:"

```
var App = React.createClass({
  render: function() {
    return <div>
      Hello, from Shawn!
    </div>;
  }
});
```

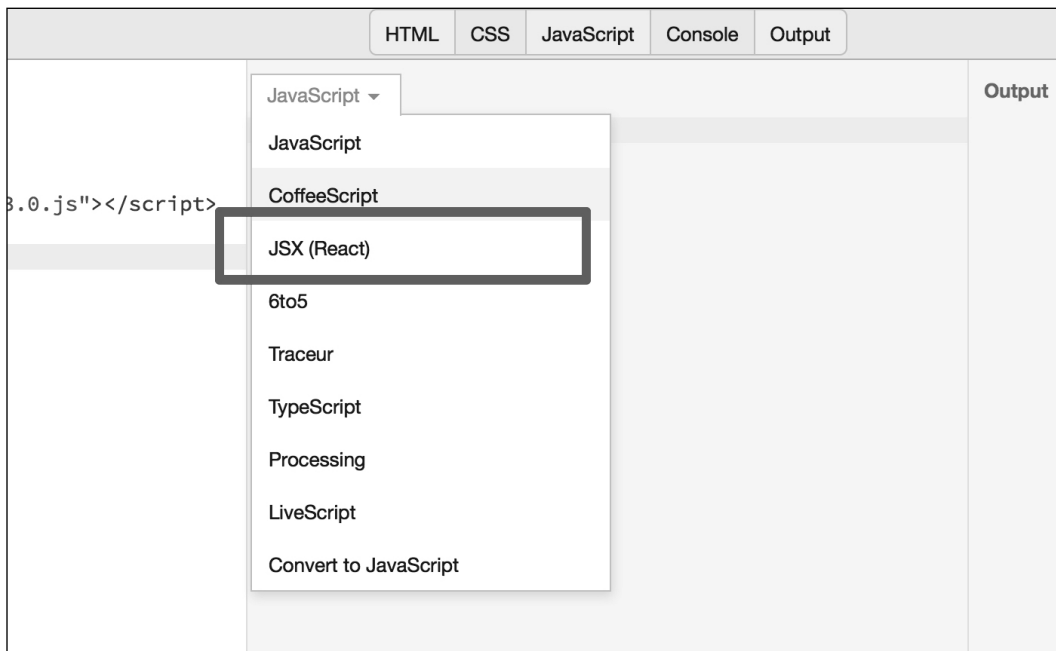
```
});
```

```
React.render(React.createElement(App), document.body);
```

"As you can see, what we did here was that instead of using `createElement`, we directly wrote the `div` tag. This is very similar to writing HTML markup directly. It also works right out of the JavaScript file."

"Mike, the code is throwing some errors on JSBin."

"Oh, right. We need to make use of the JSX transformer library so that React and the browser can understand the syntax. In our case, we need to change the type of JavaScript, which we are using, to be used to interpret this code. What we need to do is change from **JavaScript** to **JSX (React)**, from the dropdown on the JavaScript frame header, as follows:"



"That should do it."

"Looks good, Mike. It's working."

"Now you will see something similar to the following:"

```
Hello, from Shawn!
```

Back to work

"That's good to start, Shawn. Let's move back to the task of building our app using Open Library's Recent changes API now. We already have a basic prototype ready without using ReactJS."

"We will be slowly replacing parts of it using ReactJS."

"This is how the information is displayed right now, using server-side logic, as follows:"



The screenshot shows a web interface with a title 'Recent Changes' and a table of recent changes. The table has three columns: 'Last updated at', 'By Author', and 'Summary'. The data rows are: '2 minutes ago' by 'Jill Dupre' with 'Created new account', '1 hour ago' by 'Lose White' with 'Added fist chapter', and '2 hours ago' by 'Jordan Whash' with 'Created new account'. The interface also includes a 'Run with JS' button and an 'Auto-run JS' checkbox.

Last updated at	By Author	Summary
2 minutes ago	Jill Dupre	Created new account
1 hour ago	Lose White	Added fist chapter
2 hours ago	Jordan Whash	Created new account

"First task that we have is to display the information retrieved from the Open Library Recent Changes API in a table using ReactJS similar to how it's displayed right now using server-side."

"We will be fetching the data from the Open Library API similar to the following:"

```
var data = [{ "when": "2 minutes ago",
              "who": "Jill Dupre",
              "description": "Created new account"
            },
            {
              "when": "1 hour ago",
              "who": "Lose White",
              "description": "Added fist chapter"
            },
            {
              "when": "2 hours ago",
              "who": "Jordan Whash",
              "description": "Created new account"
            }
          ];
```

"Let's use this to prototype our app for now. Before that, let's take a look at the simple HTML version of this app. In our `React.render` method, we start returning a table element, as follows:"

```
var App = React.createClass({

  render: function() {
    return <table>
    <thead>
      <th>When</th>
      <th>Who</th>
      <th>Description</th>
    </thead>
    <tr>
      <td>2 minutes ago</td>
      <td>Jill Dupre</td>
      <td>Created new account</td>
    </tr>
    <tr>
      <td>1 hour ago</td>
      <td>Lose White</td>
      <td>Added fist chapter</td>
    </tr>
    <tr>
      <td>2 hours ago</td>
      <td>Jordan Whash</td>
      <td>Created new account</td>
    </tr>
  </table>
  }
});
```

"This should start displaying our table with three rows. Now, go ahead and add a heading at top of this table from the `React App`, as follows:"

```
...
return <h1>Recent Changes</h1>
      <table>
        ...
      </table>
...
```

"There, something like that?" asked Shawn. "Oh, that didn't work."

"That's because React expects our render method to always return a single HTML element. In this case, after you added the `h1` heading, our app started returning two elements, which is wrong. There'll be many cases when you will come across this. To avoid this, just wrap the elements in a `div` or `span` tag. The main idea is that we just want to return a single element from the render method."

"Got it. Something like this?"

```
...
return <div>
  <h1>Recent Changes</h1>
  <table>
    ...
  </table>
</div>
...
```

Displaying static data

"Awesome! Looks good. Now, let's change our table that is displaying static information, to start fetching and displaying this information in the rows from the JSON data that we had before."

"We'll define this data in the render method itself and see how we would be using it to create our table. We'll basically just be looping over the data and creating elements, that is, table rows in our case, for the individual data set of events. Something like this:"

```
...
var data = [{ "when": "2 minutes ago",
              "who": "Jill Dupre",
              "description": "Created new account"
            },
            {
              "when": "1 hour ago",
              "who": "Lose White",
              "description": "Added fist chapter"
            },
            {
              "when": "2 hours ago",
              "who": "Jordan Whash",
              "description": "Created new account"
            }
          ];

var rows = data.map(function(row) {
```

```
    return <tr>
      <td>{row.when}</td>
      <td>{row.who}</td>
      <td>{row.description}</td>
    </tr>
  });
  ...
```

"Notice how we are using `{ }` here. `{ }` is used in JSX to embed dynamic information in our view template. We can use it to embed the JavaScript objects in our views, for example, the name of a person or heading of this table. As you can see, what we are doing here is using the `map` function to loop over the dataset that we have. Then, we are returning a table row, constructed from the information available from the row object – the details about when the event was created, who created it and event description."

"We are using JSX syntax here to construct the rows of table. However, it is not used as the final return value from render function."

"That's correct, Shawn. React with JSX allows us to arbitrarily create elements to be used in our views, in our case, creating it dynamically from the dataset that we have. The `rows` variable now contains a part of view that we had used at a different place. We can also build another component of the view on top of it."

"That's the beauty of it. React allows us to dynamically create, use, and reuse the parts of views. This is helpful to build our views, part by part, in a systematic way."

"Now, after we are done with building our rows, we can use them in our final render call."


"So now, the return statement will look something similar to the following:"

```
...
return <table>
  <thead>
    <th>When</th>
    <th>Who</th>
    <th>Description</th>
  </thead>
  {rows}
</table>
...
```

"Here's how the complete render method now looks after building up rows with static data:"

```
render: function() {
  var data = [{ "when": "2 minutes ago",
                "who": "Jill Dupre",
                "description": "Created new account"
              },
              {
                "when": "1 hour ago",
                "who": "Lose White",
                "description": "Added fist chapter"
              },
              {
                "when": "2 hours ago",
                "who": "Jordan Whash",
                "description": "Created new account"
              }
            ];

  var rows = data.map(function(row) {
    return <tr>
      <td>{row.when}</td>
      <td>{row.who}</td>
      <td>{row.description}</td>
    </tr>
  })
  return <table>
    <thead>
      <th>When</th>
      <th>Who</th>
      <th>Description</th>
    </thead>
    {rows}
  </table>}
```

Output			Run with JS	Auto-run JS <input checked="" type="checkbox"/>	
When	Who	Description			
2 minutes ago	Jill Dupre	Created new account			
1 hour ago	Lose White	Added fist chapter			
2 hours ago	Jordan Whash	Created new account			

"That's starting to look like where we want to reach."

Passing data to components

"Do we define our data and everything else in the render method?"

"I was just getting to that. Our component should not contain this information. The information should be passed as a parameter to it."

"React allows us to pass the JavaScript objects to components. These objects would be passed when we call the `React.render` method and create an instance of the `<App>` component. The following is how we can pass objects to it:"

```
React.render(<App title='Recent Changes' />, document.body);
```

"Notice how are using the `<App/>` syntax here, instead of `createElement`. As I mentioned previously, we can create elements from our components and represent them using JSX as done earlier."

```
React.render(React.createElement(App), document.body)
```

"The preceding code becomes the following:"

```
React.render(<App />, document.body)
```

"That looks even more cleaner", said Shawn.

"As you can see, we are passing the title for our table as the `title` parameter, followed by the contents of the title. React makes this data passed to the component as something called `props`. The `props`, short for properties, are a component's configuration options that are passed to the component when initializing it."

"These `props` are just plain JavaScript objects. They are made accessible to us within our component via the `this.props` method. Let's try accessing this from the `render` method, as follows:"

```
...
render: function() {
  console.log(this.props.title);
}
...
```

"That should start logging the title that we passed to the component to the console."

"Now, let's try to abstract the headings as well as the JSON data out of the render method and start passing them to the component, as follows:"

```
var data = [{ "when": "2 minutes ago",
              "who": "Jill Dupre",
              "description": "Created new account"
            },
            ...
          ]];
var headings = ['When', 'Who', 'Description']
<App headings = {headings} data = {data} />
```

"There. We pulled the data out of the render method and are now passing it to our component."

"We defined the dynamic headers for our table that we will start using in the component."

"Here the curly braces, used to pass the parameters to our component, are used to specify the JavaScript expressions that will be evaluated and then used as attribute values."

"For example, the preceding JSX code will get translated into JavaScript by React, as follows:"

```
React.createElement(App, { headings: headings, data: data });
```

"We will revisit props later. However, right now, let's move on to complete our component."

"Now, using the passed data and headings via props, we need to generate the table structure in the app's render method."

"Let's generate the headings first, as follows:"

```
var App = React.createClass({

  render: function() {
    var headings = this.props.headings.map(function(heading) {
      return(<th>
        {heading}
      </th>);
    });
  }
});
```

"Notice, how we are using `this.props.headings` to access the passed information about headings. Now let's create rows of the table similar to what we were doing earlier:"

```
var App = React.createClass({

  render: function() {
    var headings = this.props.headings.map(function(heading) {
      return(<th>
        {heading}
      </th>);
    });

    var rows = this.props.data.map(function(change) {
      return(<tr>
        <td> { change.when } </td>
        <td> { change.who } </td>
        <td> { change.description } </td>
      </tr>);
    });
  }
});
```

"Finally, let's put the headings and rows together in our table."

```
var App = React.createClass({

  render: function() {
    var headings = this.props.headings.map(function(heading) {
      return(<th>
        {heading}
      </th>);
    });

    var rows = this.props.data.map(function(change) {
      return(<tr>
        <td> {change.when} </td>
        <td> {change.who} </td>
        <td> {change.description} </td>
      </tr>);
    });

    return(<table>
      {headings}
    </table>);
  }
});
```

```
        {rows}
      </table>);

    }
  });

  React.render(<App headings = {headings} data = {data} />,
    document.body);
```

"The table is now displayed with the passed dynamic headers and JSON data."

"The headings can be changed to ["Last change at", "By Author", "Summary"] and the table in our view will get updated automatically."

"Alright, Shawn, go ahead and add a title to our table. Make sure to pass it from the props."

"Ok," said Shawn.

"Now, the render method will be changed to the following:"

```
...
  return <div>
    <h1>
      {this.props.title}
    </h1>
    <table>
      <thead>
        {headings}
      </thead>
      {rows}
    </table>
  </div>
...
```

"While the call to `React.render` will change to the following:"

```
var title = 'Recent Changes';
React.render(<App headings={headings} data={data} title={title}/>,
  document.body);
```

"Awesome. You are starting to get a hang of it. Let's see how this looks in completion shall we?"

```
var App = React.createClass({
  render: function() {
    var headings = this.props.headings.map(function(heading) {
```

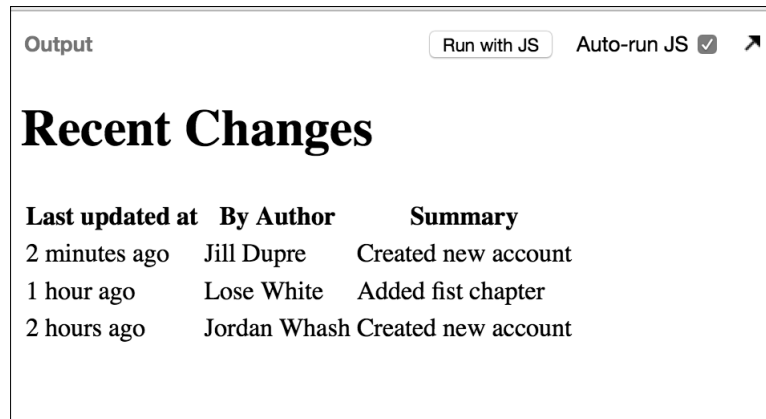
```
        return(<th>
                {heading}
            </th>);
    });

    var rows = this.props.data.map(function(row) {
    return <tr>
        <td>{row.when}</td>
        <td>{row.who}</td>
        <td>{row.description}</td>
    </tr>

    })
    return <div><h1>{this.props.title}</h1><table>
    <thead>
    {headings}
    </thead>
    {rows}
    </table></div>
    }
    });
    var data = [{ "when": "2 minutes ago",
                  "who": "Jill Dupre",
                  "description": "Created new account"
                },
                {
                  "when": "1 hour ago",
                  "who": "Lose White",
                  "description": "Added fist chapter"
                },
                {
                  "when": "2 hours ago",
                  "who": "Jordan Whash",
                  "description": "Created new account"
                }
    ]};

    var headings = ["Last updated at", "By Author", "Summary"]
    var title = "Recent Changes";
    React.render(<App headings={headings} data={data} title={title}/>,
    document.body);
```


"We should again start seeing something as follows:"



"Here we have it, Shawn. Our very first component using React!", said Mike.

"This looks amazing. I can't wait to try out more things in React!", exclaimed Shawn.

Summary

In this chapter, we started with React and built our first component. In the process, we studied the top-level API of React to construct components and elements. We used JSX to construct the components. We saw how to display static information using React and then gradually replaced all the static information with dynamic information using props. In the end, we were able to tie all ends together and display mock data in the format that is returned from Open Library's Recent Changes API using React.

In the next chapter, we will dive deep into JSX internals and continue building our application for Recent Changes API.

[Get more information ReactJS by Example – Building Modern Web Applications with React](#)

Where to buy this book

You can buy ReactJS by Example – Building Modern Web Applications with React from the [Packt Publishing website](#).

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

[Click here](#) for ordering and shipping details.



www.PacktPub.com

Stay Connected:

