Gaurav Singhal

How to Implement a "Read More" Link in React

Gaurav Singhal

Jun 5, 2020 • 8 Min read • 10,542 Views

Jun 5, 2020 • 8 Min read • 10,542 Views

Web Development Front End Web Dev... Client-side Framew... React

Introduction

38

Introduction

React's state triggers render every time it changes. This property allows the implementation of many different features, including tons of custom UI components such as *popovers*, *tooltips*, *accordions*, etc. Popular UI libraries for React use this concept extensively for their components.

This guide explores the simplest way to implement a "read more" link in a React app.

About "Read More" Links

As the name suggests, a "read more" link is a hyperlink or an `<a>` rendered on the DOM as collapsible content. When you click the link, you are shown more content on the same page that is otherwise hidden. They are extensively used on web apps these days to manage additional content and are also quite popular in native apps for the same use case. The underlying concept behind any collapsible or expandable UI component is the same as that of a "read more" link, even though they may have different use cases.

Using "Read More" Links

Typically, a "read more" link is used to display additional content on web pages where either the total content is not known beforehand or it occupies too much space in a viewport. Accordions and collapsible content work in the same way, the only difference being that they generally present a list of items, either a list of cards or a list of `<div>` with content inside.

Read more" links are a great way to manage content on a web app without taking up extra space. You can use them to display the full bio of a person on a profile page in your app. You can also use them to conditionally display additional content on blogs or news feed details in a social media application.

Implementing "Read More" Links

Let's jump into implementing "read more" links in React.

Approach

The concept behind implementing a "read more" link is *conditional rendering*. Using a simple `<a>` on the page coupled with an on-click event, you can fire a function to conditionally render the additional content in its parent `<div>`. Your app will have a state that will be set to false initially, since the link is collapsed. This state variable will be set to true inside the event handler, and using some condition blocks such as if-else statements or ternary operators, the component or content can be outputted. Since the state variable changes on an event, a render method will be triggered and the updated DOM will be mounted on the page displaying additional data without refreshing the page.

Setup

Make sure you have *Nodejs* and *npm* installed on your machine (at least version 8 or higher)

Create a new project using create-react-app:

shell

```shell
npx create-react-app react-read-more
```

Cleaning Up the Template

Remove the logo, `App.css`, and all their imports from `App.js`. Clean out the starter template inside the App component. Your `App.js` should look like this:

jsx

```jsx
import React from 'react';
import './App.css';

function App() {
  return(
      <div className="App">
     </div>
  )
}
```

```

export default App;
```

Note: Sometimes removing `App.css` might land you an error. In such cases, instead of deleting the `App.css`, keep it empty. Or, if you have already deleted it, create a blank CSS file under the same name in the root directory.

Adding Styles

Add the following styles inside `index.css`.

```css
.App{
  margin: 10% 40%;
}
.read-more-link{
  color: blueviolet;
  text-decoration: underline;
  letter-spacing: 1px;
  cursor: pointer;
}
.extra-content{
  color: cornflowerblue;
  font-weight: 500;
}
```

Creating State and Local Variables

Import `useState` hook from React to use state inside a functional component.

```jsx
import React,{useState} from 'react';
```

Create a state variable to store the present state that conveys information about the expanded or collapsed state of the link. Call it `readMore`.

```jsx
...
const [readMore,setReadMore]=useState(false);
...
```

Keeping a clean code saves the JSX for the link and the extra content inside JavaScript constants.

```jsx
...
const extraContent=<div>
      <p className="extra-content">
        Lorem ipsum dolor sit amet consectetur adipisicing elit. Qui, cor
        porro quasi culpa nulla rerum quis minus voluptatibus sed hic ad
        commodi officia aliquam! Maxime.
      </p>
  </div>
const linkName=readMore?'Read Less << ':'Read More >> '
...
```

The first constant stores a `<div>` with a child `<p>` and embedded dummy lorem ipsum. You can have additional content, links, cards, or any kind of HTML that you want to display here inside the parent `<div>` .

The second constant conditionally stores the name of the link. Your link should change its name dynamically depending on what it's doing at the moment. As the assignment is made on conditionally checking the state variable, any change made to the state variable will also make the necessary changes to this constant.

Conditionally Outputting the HTML

Finally, inside the return statement, render an `<a>` with an `onClick` event and fire a function on triggering this event. Inside this function or the event handler, toggle the `readMore` state and output the `linkName` inside the `<a>`. Finally, using a ternary operator, output the constant `extraContent` holding the additional content.

jsx

```jsx
    ...
    return (
       <div className="App">
         <a className="read-more-link" onClick={()=>{setReadMore(!readMore)}}
         {readMore && extraContent}
```

```
        </div>
      );
    ...
```

All in all, your `App.js` should look like this:

jsx

```
import React,{useState} from 'react';

function App() {
  const [readMore,setReadMore]=useState(false);
  const extraContent=<div>
      <p className="extra-content">
        Lorem ipsum dolor sit amet consectetur adipisicing elit. Qui, cor
        porro quasi culpa nulla rerum quis minus voluptatibus sed hic ad
        commodi officia aliquam! Maxime.
      </p>
  </div>
  const linkName=readMore?'Read Less << ':'Read More >> '
  return (
    <div className="App">
      <a className="read-more-link" onClick={()=>{setReadMore(!readMore)}
      {readMore && extraContent}
    </div>
  );
}

export default App;
```

Testing

Inside the root directory, run :

```
npm start
```

Click the link and it will show you some additional content. Click it again and the content goes away.

Improvising

Remember to break the entire code into modular and reusable components. Feel free to customize this feature as per the needs of your app. If you intend to show only text, you can render this component inside a card already holding some truncated content. In some cases, instead of rendering a link, you can render it as a **Show More** button. If you want to make a custom *accordion* or an *expanded menu* on the navbar, you can use the same concept of conditional rendering along with some CSS touch up.

Conclusion

"Read more" links are common these days and can be easily implemented using conditional rendering, as explained in this guide. You can also use several React packages for the same purpose, but using a library for such a task seems like overkill. Implementing components using some well-defined logic not only widens the learning horizons of a developer, but also mkeeps the app free from unnecessary packages and libraries. As an exercise, you can try figuring out a neat CSS hack to do something similar.

38