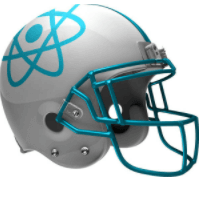
|  |  |
| --- | --- |
| **ch46.ReactJS Helmet** | **Date: 22-02-2022** |

**Topics**

React Props Validation,

# React Props Validation

# React Helmet



Every developer aspires that his website should appear at the top of the search results pages in a browser. Unfortunately, there are certain instances where the search engine crawler cannot prove to be reliable. This problem occurs because of the inability of the browser to render JavaScript. This means that SPA build on the **React** or **Angular platform** cannot favor the search engine crawlers. This is one of the most significant issues the developers face to bring their pages to the top of the webpage. So, to solve the problem, Server Side Rendering may be used, but it generally has limitations to adapt. This is where React Helmet comes to the rescue.

## **What is React Helmet?**

React Helmet can be termed as the document **head manager** for React-based applications. Using it, it becomes very easy for developers to update meta tags present on the server-side and the client-side. This library can be termed perfect for applications where SEO plays a crucial role. It is also a media-friendly library with different dependencies and features that make its usage easy in any React app. We will study all the aspects in this tutorial and see how you can add React Helmet into your project.

### **Setup**

Before moving any further, we need to install React Helmet using the traditional and most effective npm approach. We will first initialize a new React project using some commands and start the development process when the server is ready.

To install and initialize React Helmet, follow the below command.

Skip Ad

1. npx create-react-app react-helmet-tutorial
2. cd react-helmet-tutorial
3. npm start

Ensure that the system port 3000 is occupied, and then you can get straightforward head over to **http://localhost:3000** and see your app.

## **Newly Created React App**

We might have known that, like most SPAs available, our application will have a default <head> element. This can be easily verified if we move to the browser for inspection and press F12 in Chrome or use Ctrl+Shift+C in the Firefox browser. Analyzing the content on Chrome, we might see what the <head> element looks like.

1. **<head>**
2. **<meta** charset="utf-8"**>**
3. **<link** rel="icon" href="/favicon.ico"**>**
4. **<meta** name="viewport" content="width=device-width, initial-scale=1"**>**
5. **<meta** name="theme-color" content="#000000"**>**
6. **<meta** name="description" content="Web site created using create-react-app"**>**
7. **<link** rel="apple-touch-icon" href="/logo192.png"**>**
9. **<link** rel="manifest" href="/manifest.json"**>**
11. **<title>**React App**</title>**
12. **<style** type="text/css"**>**
13. <!-- CSS -->
14. **</style>**
15. **</head>**

There's another way to manage the <head>. This is done by updating the public/index.html directory but should not always be an alternative since it is a bad practice. After this step, we need to install and make use of React Helmet in our project.

## **Installation and Setup**

React Helmet can be installed into your React project through Yarn or npm using the command given below.

1. npm install react-helmet
3. # OR, using Yarn:
5. yarn add react-helmet

The next is to follow the process of importing the Helmet component. Once done, we can easily add elements to the <head> of the document. See how the following step is implemented in the below code snippet.

1. // src/App.js
3. import React from "react";
4. import "./App.css";
5. import { Helmet } from "react-helmet";
7. function App() {
8. return (
9. **<div** className="App"**>**
11. **<Helmet>**
12. **<html** lang="en" **/>**
13. **<title>**React Helmet Tutorial**</title>**
14. **<meta** name="description" content="Tutorial for React Helmet" **/>**
15. **<meta** name="theme-color" content="#E6E6FA" **/>**
16. **</Helmet>**
18. **<header** className="App-header"**>**Title will be React Helmet
19. Tutorial**</header>**
20. **</div>**
21. );
22. }
23. export default App;

As soon as the above code snippet is updated to the directory src/App.js file, we will notice the changes in our React application.

In this code snippet, to keep the understanding level easy, we have used only <title>, <html> and <meta> tag elements for description and some amount of theme color is also used for the other elements like **base, meta, link, script,** etc. These elements are generally treated as the children of Helmet. Below is the example where we can see the attributes that have been defined for the body tag and html tag.

1. **<Helmet>**
2. {/\* html attributes \*/}
3. **<html** lang="en" **/>**
5. {/\* body attributes \*/}
6. **<body** className="dark" **/>**
8. {/\* title element \*/}
9. **<title>**React Helmet Tutorial**</title>**
11. {/\* base element \*/}
12. **<base** target="\_blank" href="https://blog.logrocket.com/" **/>**
14. {/\* meta elements \*/}
15. **<meta** name="description" content="Tutorial for React Helmet" **/>**
17. **<meta** name="theme-color" content="#E6E6FA" **/>**
19. {/\* link elements \*/}
20. **<link** rel="canonical" href="https://blog.logrocket.com/" **/>**
21. **</Helmet>**
23. // src/Child.js
25. import React from "react";
26. import { Helmet } from "react-helmet";
27. function Child() {
28. return (
29. **<div>**
30. **<Helmet>**
31. **<title>**Child Component Rocks!**</title>**
32. **</Helmet>**
33. {"  "}  This time title will be  Child Component Rocks!
34. **</div>**
35. );
36. }
37. export default Child;

#### **Note: There is ambiguity in the above example because we may not know whether the preference would be given to the Child or Parent.**

To clarify this, the preference is given to the child component. Consider the below code:

1. // src/Child.js
3. import React from "react";
4. import { Helmet } from "react-helmet";
5. function Child() {
6. return (
7. **<div>**
8. **<Helmet>**
9. **<title>**Child Component Rocks!**</title>**
10. **</Helmet>**
11. {"  "}  This time title will be  Child Component Rocks!
12. **</div>**
13. );
14. }
15. export default Child;

Now, to import the child component, i.e. the Child.js file, to the parent component, i.e. App.js file, we must not forget to change the document's title according to the child component. But, it is also important to understand that the metadata and the theme color cannot be overwritten. The demonstration for the same is shown in the code snippet given below.

1. // src/App.js
3. **import** React from "react";
4. **import** "./App.css";
5. **import** { Helmet } from "react-helmet";
6. **import** Child from "./Child";
7. function App() {
8. **return** (
9. <div className="App">
10. <Helmet>
11. <title>React Helmet Tutorial</title>
12. <meta name="description" content="Tutorial for React Helmet" />
13. <meta name="theme-color" content="#E6E6FA" />
14. </Helmet>
16. <header className="App-header">
17. Title will not be React Helmet Tutorial <Child />
18. </header>
19. </div>
20. );
21. }
22. export **default** App;

Another instance to consider here is that the one occurring later will always be given the utmost preference between two child components. For instance, consider two child components <Child1> and <Child2>, then the <Child2> will be given the preference and vice-versa. The demonstration to support this statement is given below.

1. // src/App.js
2. **import** React from "react";
3. **import** "./App.css";
4. **import** { Helmet } from "react-helmet";
5. **import** Child1 from "./Child1";
6. **import** Child2 from "./Child2";
7. function App() {
8. **return** (
9. <div className="App">
10. <Helmet>
11. <title>React Helmet Tutorial</title>
12. <meta name="description" content="Tutorial for React Helmet" />
13. <meta name="theme-color" content="#E6E6FA" />
14. </Helmet>
15. <header className="App-header">
16. Title will not be React Helmet Tutorial
18. <Child1 />
19. <Child2 />
20. </header>
21. </div>
22. );
23. }
24. export **default** App;

It is evident from the above code that the title will be set according to the <Child2> component. If the order is reversed, then the <Child1> will be set to the preference. This can be shown using the code snippet shown below.

1. <header className="App-header">
2. Title will not be React Helmet Tutorial
3. <Child2 />
4. <Child1 />
5. </header>

## **React Helmet with SSR integration**

As we previously discussed, React Helmet is mostly nominated to be the best partner for app rendering residing in the server-side of the application; you can easily refer to the terms by setting up a sample React SSR application. Thus, in the application's server-side code, just after the ReactDOMServerr's renderToString or renderToStaticMarkup, we need to apply the method of render static in the Helmet. To illustrate how it is done, consider the below code snippet.

1. // server/index.js
2. **import** React from "react";
3. **import** { renderToString } from "react-dom/server";
4. **import** express from "express";
5. **import** App from "./src/App";
6. **import** { Helmet } from "react-helmet";
7. **const** app = express();
9. app.get("/\*", (req, res) => {
10. **const** app = renderToString(<App />);
11. **const** helmet = Helmet.renderStatic();
13. **const** html = `
14. <!DOCTYPE html>
15. <html ${helmet.htmlAttributes.toString()}>
16. <head>
17. ${helmet.title.toString()}
18. ${helmet.meta.toString()}
19. ${helmet.link.toString()}
20. </head>
21. <body ${helmet.bodyAttributes.toString()}>
22. <div id="root">
23. ${app}
24. </div>
25. </body>
26. </html>
27. `;
28. res.send(html);
29. });
30. app.listen(8000);

It is important to understand that in every Helmet property, there is a method of toString(). This method is mainly used inside the HTML string. It is also an instance of Helmet's render static that generally returns with all the properties covered like the **script, link, title,** etc., and all of these properties have default toString() method contained in them.

It might not always be a good option to work synchronously to proceed with the React Helmet's SSR implementation. If we are open or bound to make asynchronous requests like streaming, the React Helmet would lead to potential threats and errors. Hence, to cope up with this issue, we use the react-helmet-async method. This method solves this problem by forking React Helmet explicitly. To install it, we need to run the following terminal command in the node environment.

1. npm i react-helmet-async

After running this command, we need to be assured to keep everything except just using HelmetProvider so that the React tree gets encapsulated on both the server-side and the client-side. The simple demonstration for this is shown below.

1. // src/App.js
2. **import** React from "react";
3. **import** "./App.css";
4. **import**  {Helmet, HelmetProvider } from 'react-helmet-async';
5. function App() {
6. **return** (
7. <HelmetProvider>
8. <div className="App">
9. <Helmet>
10. <title>React Helmet Tutorial</title>
11. <meta name="description" content="Tutorial for React Helmet" />
12. <meta name="theme-color" content="#E6E6FA" />
13. </Helmet>
14. <header className="App-header">
15. Title will be React Helmet Tutorial
16. </header>
17. </div>
18. </HelmetProvider>
19. );
20. }
21. export **default** App;

## **React Helmet & React Router**

React Helmet works great if the application being build uses React Router for routing. One hook associate with it is that we need to use React Helmet with every Router, so the routes remain intact, and this can be demonstrated using the below code snippet.

1. // src/App.js
2. **import** React from "react";
3. **import** { Helmet } from "react-helmet";
4. **import** { BrowserRouter, Switch, Route, Link } from "react-router-dom";
5. **import** Home from './Home'
6. **import** About from './About'
7. export **default** function App() {
8. **return** (
9. <div>
10. <BrowserRouter>
11. <Switch>
12. <Route exact path="/">
13. <Home />
14. </Route>
15. <Route path="/about">
16. <About />
17. </Route>
18. </Switch>
19. </BrowserRouter>
20. </div>
21. );
22. }
23. Where Home.js is:
25. // src/Home.js
26. **import** React from "react";
27. **import** { Helmet } from "react-helmet";
28. **import** {  Link } from "react-router-dom";
30. **const** Home = () => (
31. <div>
32. <Helmet>
33. <title>Home</title>
34. </Helmet>
35. <h2>Home.</h2>
36. <Link to="/about">About</Link>
37. </div>
38. );
40. export **default** Home
41. And About.js is:
43. // src/About.js
44. **import** React from "react";
45. **import** { Helmet } from "react-helmet";
46. **import** {  Link } from "react-router-dom";
48. **const** About = () => (
49. <div>
50. <Helmet>
51. <title>About</title>
52. </Helmet>
53. <h2>About.</h2>
54. <Link to="/">Home</Link>
55. </div>
56. );
57. export **default** About;

This is how routing is done using React Helmet and React Router, and also, the method can be used as many routes as possible.

## **Conclusion**

In this tutorial, we learned how to install, set up and use React Helmet. React Helmet is a powerful library mainly used as the document head manager that brings our SEO to the top of the web page. It is small and convenient to use, and if the application is built with the intent to remain at the top charts, it outlasts better than any library in its class.

Next Topic[#](https://www.javatpoint.com/react-helmet)