**7 Ways to Implement Conditional Rendering in React Applications**

Published on September 19, 2020 · Updated on February 5, 2021

* [JavaScript](https://www.digitalocean.com/community/tags/javascript)
* [React](https://www.digitalocean.com/community/tags/react)



By Philip Obosi



[**Introduction**](https://www.digitalocean.com/community/tutorials/7-ways-to-implement-conditional-rendering-in-react-applications#introduction)

You can build Single Page Applications (SPA) that are dynamic and highly interactive with React. One feature that allows for this is conditional rendering.

Conditional rendering is a term to describe the ability to render different user interface (UI) markup if a condition is true or false. In React, it allows us to render different elements or components based on a condition. This concept is applied often in the following scenarios:

* Rendering external data from an API.
* Showing or hiding elements.
* Toggling application functionality.
* Implementing permission levels.
* Handling authentication and authorization.

In this article, you will examine seven ways to implement conditional rendering in React applications.

[**Prerequisites**](https://www.digitalocean.com/community/tutorials/7-ways-to-implement-conditional-rendering-in-react-applications#prerequisites)

To complete this tutorial, you’ll need:

* An understanding of JavaScript variables and functions. You can consult the [How To Code in JavaScript](https://www.digitalocean.com/community/tutorial_series/how-to-code-in-javascript) series to learn more.
* An understanding of importing, exporting, and rendering React components. You can check out our [How To Code in React.js](https://www.digitalocean.com/community/tutorial_series/how-to-code-in-react-js) series.
* Node.js installed locally, which you can do by following [How to Install Node.js and Create a Local Development Environment](https://www.digitalocean.com/community/tutorial_series/how-to-install-node-js-and-create-a-local-development-environment).

This tutorial was verified with Node v15.6.0, npm v7.4.0, and react v17.0.1.

[**Setting Up the Sample Project**](https://www.digitalocean.com/community/tutorials/7-ways-to-implement-conditional-rendering-in-react-applications#setting-up-the-sample-project)

Consider an application that requires a user to log in. If the user is logged out, it will display a **Login** button. If the user is logged in, it will display a **Logout** button.

Start with using [create-react-app](https://www.digitalocean.com/community/tutorials/react-create-react-app) to generate a React App:

1. npx create-react-app react-conditional-rendering-example

Change into the new project directory:

1. cd react-conditional-rendering-example

Next, open the App.js file in your code editor. And replace the contents with the following lines of code:

src/App.js

import React, { Component } from "react";

import './App.css';

class App extends Component {

constructor(props) {

super(props);

this.state = {

isLoggedIn: true

};

}

render() {

return (

<div className="App">

<h1>

This is a Demo showing several ways to implement Conditional Rendering in React.

</h1>

<button>Login</button>

<button>Logout</button>

</div>

);

}

}

export default App;

Next, open the App.css file in your code editor. And replace the contents with the following lines of code:

src/App.css

body {

padding: 1em;

}

h1 {

font-size: 3em;

font-weight: 500;

text-align: center;

margin-bottom: 1em;

margin-right: 1em;

padding: 0;

}

button {

appearance: none;

background-color: #246bec;

border: 1px solid #246bec;

border-radius: 0;

box-sizing: border-box;

color: #ffffff;

display: block;

font-size: 2em;

font-weight: 500;

margin-bottom: 1em;

margin-top: 1em;

padding: .5em;

width: 100%;

transition: border-color, background-color 300ms ease-in-out;

}

button:focus {

border-color: #00006D;

}

button:hover {

background-color: #0B52D3;

}

button:active {

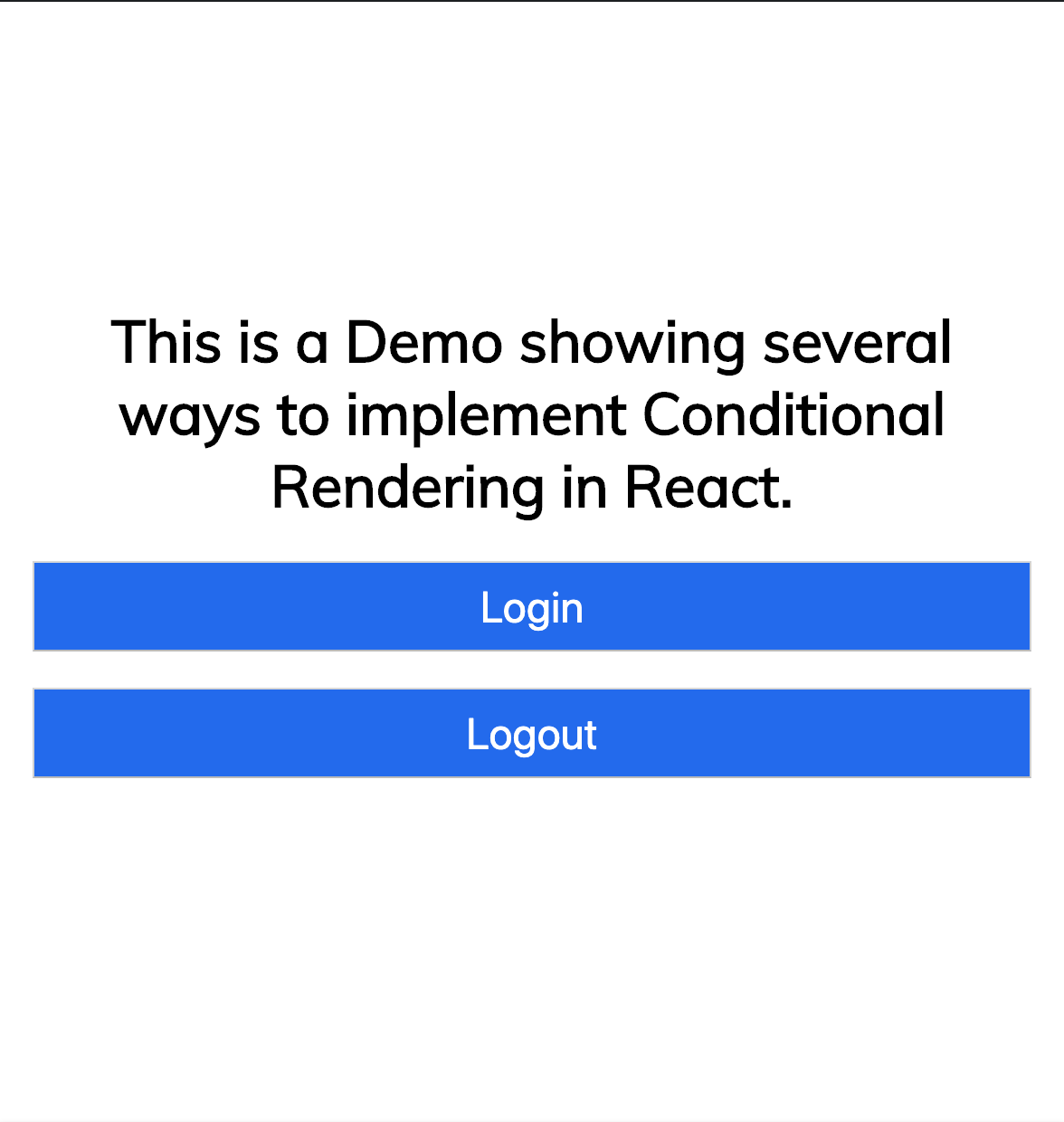
background-color: #00006D;

}

Then, run the application from your terminal window:

1. npm start

And interact with the application in your browser:



Each of the conditional rendering approaches will build upon this code. You may wish to create a git commit at this point to rollback changes as you progress through this tutorial.

At this point, you will have a React application that displays a **Login** and a **Logout** button. Your goal is to have only one of these buttons display. Let’s look at conditional rendering approaches to accomplish this.

[**1. Using an if…else Statement**](https://www.digitalocean.com/community/tutorials/7-ways-to-implement-conditional-rendering-in-react-applications#1-using-an-if-else-statement)

An if…else statement will execute the actions contained in the if block when the condition is satisfied. Otherwise, it will execute the actions contained in the else block.

In JSX, you are able to use JavaScript code with markup to render dynamic values within your application. JSX uses curly braces ({ and }) to signify expressions that need to be interpreted prior to rendering. The caveat, however, is that there is a limit to what can be done within such braces.

Let’s consider if you were to attempt to use an if…else statement in the render() method:

**Warning:** This is an example of code that will not work properly. It is presented as an example of the limitations of interpretation in the render() method.

// ...

class App extends Component {

// ...

render() {

let {isLoggedIn} = this.state;

return (

<div className="App">

<h1>

This is a Demo showing several ways to implement Conditional Rendering in React.

</h1>

{

if(isLoggedIn){

return <button>Logout</button>

} else{

return <button>Login</button>

}

}

</div>

);

}

}

// ...

This code would produce an Unexpected token error. The logic will need to be moved outside of the render() method.

Open the App.js file in your code editor, scroll down to the render() method and make the following highlighted code changes:

src/App.js

// ...

class App extends Component {

// ...

render() {

let {isLoggedIn} = this.state;

const renderAuthButton = () => {

if (isLoggedIn) {

return <button>Logout</button>;

} else {

return <button>Login</button>;

}

}

return (

<div className="App">

<h1>

This is a Demo showing several ways to implement Conditional Rendering in React.

</h1>

{renderAuthButton()}

</div>

);

}

}

// ...

This is the process of creating an extracted function. This code extracts the logic from JSX into a function renderAuthButton. And the function is executed within the JSX curly braces.

Open your application in your web browser. Previously, the **Login** and **Logout** buttons were displayed. Now, the isLoggedIn state is true and the conditional logic results in only the **Logout** button displayed.

Now, let’s consider if you were to attempt to use multiple returns in the render() method instead:

**Warning:** This is an example of poorly performant code that should be avoided.

// ...

class App extends Component {

// ...

render() {

let {isLoggedIn} = this.state;

if (isLoggedIn) {

return (

<div className="App">

<h1>

This is a Demo showing several ways to implement Conditional Rendering in React.

</h1>

<button>Logout</button>

</div>

);

} else {

return (

<div className="App">

<h1>

This is a Demo showing several ways to implement Conditional Rendering in React.

</h1>

<button>Login</button>

</div>

);

}

}

}

// ...

The snippet above would achieve the same result but bloat the component unnecessarily while introducing performance issues as a result of constantly re-rendering an unchanging component.

The best practice is to keep components as simple as possible to avoid a wasted re-render of sibling or parent components.

In your code editor, create a new AuthButton.js file:

src/AuthButton.js

import React from "react";

const AuthButton = props => {

let { isLoggedIn } = props;

if (isLoggedIn) {

return <button>Logout</button>;

} else {

return <button>Login</button>;

}

};

export default AuthButton;

AuthButton returns various elements or components depending on the value of state that is passed down via the isLoggedIn props.

Next, revisit App.js and modify it to use the new component:

src/App.js

import React, { Component } from "react";

import './App.css';

import AuthButton from "./AuthButton";

class App extends Component {

constructor(props) {

super(props);

this.state = {

isLoggedIn: true

};

}

render() {

return (

<div className="App">

<h1>

This is a Demo showing several ways to implement Conditional Rendering in React.

</h1>

<AuthButton isLoggedIn={isLoggedIn} />

</div>

);

}

}

export default App;

This is the process of creating an extracted functional component. This code produces the same result as the renderAuthButton() approach. but has the advantage of moving the changes to a separate component.

[**2. Using a switch Statement**](https://www.digitalocean.com/community/tutorials/7-ways-to-implement-conditional-rendering-in-react-applications#2-using-a-switch-statement)

As shown previously, you can conditionally return different markup from a component based on set conditions using an if…else statement. The same could be achieved with a switch statement where you can specify the markup for various conditions.

Revisi the AuthButton component and replace the if…else statement with a switch statement:

src/AuthButton.js

import React from "react";

const AuthButton = props => {

let { isLoggedIn } = props;

switch (isLoggedIn) {

case true:

return <button>Logout</button>;

break;

case false:

return <button>Login</button>;

break;

default:

return null;

}

};

export default AuthButton;

Notice how this code returns various buttons based on the value of isLoggedIn.

**Note:** It would be more practical to apply the switch statement method when there are more than two possible values or outcomes.

Furthermore, returning null from a component will cause it to hide itself (display nothing). This a good way to toggle the visibility of components.

[**3. Using Element Variables**](https://www.digitalocean.com/community/tutorials/7-ways-to-implement-conditional-rendering-in-react-applications#3-using-element-variables)

Element variables are similar to the approach to extract the conditional rendering into a function. Element variables are variables that hold JSX elements. You can conditionally assign elements or components to these variables outside the JSX and only render the variable within JSX.

The application could be rewritten like this:

src/App.js

import React, { Component } from "react";

import './App.css';

class App extends Component {

constructor(props) {

super(props);

this.state = {

isLoggedIn: true

};

}

render() {

let { isLoggedIn } = this.state;

let AuthButton;

if (isLoggedIn) {

AuthButton = <button>Logout</button>;

} else {

AuthButton = <button>Login</button>;

}

return (

<div className="App">

<h1>

This is a Demo showing several ways to implement Conditional Rendering in React.

</h1>

{AuthButton}

</div>

);

}

}

export default App;

Notice how this code conditionally assigns values - components - to AuthButton and then it can be referenced later in the JSX.

[**4. Using Ternary Operators**](https://www.digitalocean.com/community/tutorials/7-ways-to-implement-conditional-rendering-in-react-applications#4-using-ternary-operators)

The conditional (ternary) operator is the only JavaScript operator that takes three operands. This operator is frequently used as a shortcut for the if statement.

The application could be rewritten like this:

src/App.js

import React, { Component } from "react";

import './App.css';

class App extends Component {

constructor(props) {

super(props);

this.state = {

isLoggedIn: true

};

}

render() {

let { isLoggedIn } = this.state;

return (

<div className="App">

<h1>

This is a Demo showing several ways to implement Conditional Rendering in React.

</h1>

{isLoggedIn ? <button>Logout</button> : <button>Login</button>}

</div>

);

}

}

export default App;

In cases where, this approach makes the component bloated, bulky, or less readable, you may encapsulate the conditional within a functional component:

src/AuthButton.js

import React from "react";

const AuthButton = props => {

let { isLoggedIn } = props;

return isLoggedIn ? <button>Logout</button> : <button>Login</button>;

};

export default AuthButton;

The ternary approach is useful for uncomplicated if…else evaluations. For complicated comparisons and components, it may impact readability as a project grows.

[**5. Using Logical && (Short Circuit Evaluation)**](https://www.digitalocean.com/community/tutorials/7-ways-to-implement-conditional-rendering-in-react-applications#5-using-logical-short-circuit-evaluation)

Short circuit evaluation is a technique used to ensure that there are no side effects during the evaluation of operands in an expression. The logical && helps you specify that an action should be taken only on one condition, otherwise, it would be ignored entirely.

The application could be rewritten like this:

src/App.js

import React, { Component } from "react";

import './App.css';

class App extends Component {

constructor(props) {

super(props);

this.state = {

isLoggedIn: true

};

}

render() {

let { isLoggedIn } = this.state;

return (

<div className="App">

<h1>

This is a Demo showing several ways to implement Conditional Rendering in React.

</h1>

{isLoggedIn && <button>Logout</button>}

</div>

);

}

}

export default App;

This code would display the **Logout** button if isLoggedIn is true, otherwise it would display nothing.

Now, let’s consider using a second short circuit evaluation for the **Login** button:

**Warning:** This is an example of poorly performant code that should be avoided.

{isLoggedIn && <button>Logout</button>}

{!isLoggedIn && <button>Login</button>}

This code would render the right button based on the value of isLoggedIn. However, this is not recommended because there are better, cleaner ways to achieve the same effect. As your application grows, this overuse of short circuit evaluation may become cumbersome and unintuitive.

[**6. Using Immediately Invoked Function Expressions (IIFEs)**](https://www.digitalocean.com/community/tutorials/7-ways-to-implement-conditional-rendering-in-react-applications#6-using-immediately-invoked-function-expressions-iifes)

Earlier sections mentioned that JSX limitations make it unable to execute every type of JavaScript code. It is possible to bypass these limitations with Immediately Invoked Function Expressions (IFFEs). IFFEs is a JavaScript function that runs as soon as it is defined:

(function () {

// statements

})();

With this technique, you are able to write conditional logic directly within JSX but wrapped within an anonymous function that is immediately invoked on the evaluation of that code.

The application could be rewritten like this:

src/App.js

import React, { Component } from "react";

import './App.css';

class App extends Component {

constructor(props) {

super(props);

this.state = {

isLoggedIn: true

};

}

render() {

let { isLoggedIn } = this.state;

return (

<div className="App">

<h1>

This is a Demo showing several ways to implement Conditional Rendering in React.

</h1>

{(function() {

if (isLoggedIn) {

return <button>Logout</button>;

} else {

return <button>Login</button>;

}

})()}

</div>

);

}

}

export default App;

This can also be written in a slightly more concise manner using an arrow function:

{(() => {

if (isLoggedIn) {

return <button>Logout</button>;

} else {

return <button>Login</button>;

}

})()}

You may learn more about [IIFE from MDN](https://developer.mozilla.org/en-US/docs/Glossary/IIFE).

[**7. Using Enhanced JSX Libraries**](https://www.digitalocean.com/community/tutorials/7-ways-to-implement-conditional-rendering-in-react-applications#7-using-enhanced-jsx-libraries)

Certain libraries expose functionality to extend JSX, making it possible to implement conditional rendering directly with JSX. One of such libraries is [JSX Control Statements](https://github.com/AlexGilleran/jsx-control-statements). It is a [Babel](https://babeljs.io/) plugin that transforms component-like control statements into their JavaScript counterparts during transpilation.

After installing the babel-plugin-jsx-control-statements package and modifying your Babel configuration, the application could be rewritten like this:

src/App.js

import React, { Component } from "react";

import './App.css';

class App extends Component {

constructor(props) {

super(props);

this.state = {

isLoggedIn: true

};

}

render() {

let { isLoggedIn } = this.state;

return (

<div className="App">

<h1>

This is a Demo showing several ways to implement Conditional Rendering in React.

</h1>

<Choose>

<When condition={isLoggedIn}>

<button>Logout</button>;

</When>

<When condition={!isLoggedIn}>

<button>Login</button>;

</When>

</Choose>

</div>

);

}

}

export default App;

However, this approach is not recommended as the code you write is eventually transpiled to a regular JavaScript conditional. It is probably always better to just write JavaScript than add an extra dependency over something so trivial.

[**Selecting a Conditional Rendering Approach**](https://www.digitalocean.com/community/tutorials/7-ways-to-implement-conditional-rendering-in-react-applications#selecting-a-conditional-rendering-approach)

As a general rule, it is best to ensure that in implementing conditional rendering you:

* Do not change the position of components arbitrarily in order to prevent components from unmounting and remounting unnecessarily.
* Change only the markup that is concerned with the conditional rendering.
* Do not bloat your component unnecessarily within the render method as this will cause components to delay in rendering.

For more, see [this article on high performing conditionals in React](https://medium.com/@cowi4030/optimizing-conditional-rendering-in-react-3fee6b197a20) by Cole Williams.

Things to consider include:

* The size of markup to be rendered conditionally
* The number of possible outcomes
* Which would be more intuitive and readable

Generally, keep in mind the following recommendations:

* For situations where there is only one expected outcome, the “short circuit evaluation” is possibly most applicable.
* For situations where there are two expected outcomes, an if…else statement, element variable, ternary operator, or “immediately invoked function expression” is probably most applicable.
* For situations where there are more than two outcomes, a switch statement, extracted function, or extracted functional component is probably most applicable.

Bear in mind that these are recommendations and not rules. The needs and conventions of your project may require you to adopt approaches that do not follow these recommendations.

[**Conclusion**](https://www.digitalocean.com/community/tutorials/7-ways-to-implement-conditional-rendering-in-react-applications#conclusion)

In this article, you examined seven ways to implement conditional rendering in React applications. Each method has its own advantage and the choice of which to use is mostly dependent on the use case.

For more, consult the documentation for [Conditional Rendering](https://reactjs.org/docs/conditional-rendering.html).

Continue your learning with [Higher Order Components](https://www.digitalocean.com/community/tutorials/react-higher-order-components) and [Conditional Rendering with HOCs](https://www.robinwieruch.de/conditional-rendering-react/#conditional-rendering-hoc).

Want to deploy your application quickly? Try Cloudways, the #1 managed hosting provider for small-to-medium businesses, agencies, and developers - for free. DigitalOcean and Cloudways together will give you a reliable, scalable, and hassle-free managed hosting experience with anytime support that makes all your hosting worries a thing of the past. Start with $100 in free credits!

[Learn more here](https://www.cloudways.com/en/digital-ocean-hosting.php?id=1248182)

About the authors



Philip Obosi

author

**Still looking for an answer?**

[Ask a question](https://www.digitalocean.com/community/questions)[Search for more help](https://www.digitalocean.com/community)

Was this helpful?

1 Comments

Top of Form

Bottom of Form

Top of Form

﻿

Bottom of Form

This textbox defaults to using **Markdown** to format your answer.

You can type **!ref** in this text area to quickly search our full set of tutorials, documentation & marketplace offerings and insert the link!

[Sign In or Sign Up to Comment](https://www.digitalocean.com/api/dynamic-content/v1/login?success_redirect=https%3A%2F%2Fwww.digitalocean.com%2Fcommunity%2Ftutorials%2F7-ways-to-implement-conditional-rendering-in-react-applications&error_redirect=https%3A%2F%2Fwww.digitalocean.com%2Fauth-error&type=register&site=community)

[ukolsch](https://www.digitalocean.com/community/users/ukolsch) • [March 6, 2021](https://www.digitalocean.com/community/tutorials/7-ways-to-implement-conditional-rendering-in-react-applications?comment=96589)

Could you please elaborate a bit on this:

Option 5 you classify as ‘poorly performant code’, as opposed to option 6, which is not ‘poorly performant code’. The reason given why option 5 is pretty much just readability, as far as i can tell (…may become cumbersome and unintuitive). Now if you have more then 2 conditions, like in the example below, option 5 seems much more concise and easier to read then option 6 (that’s obviously a very subjective judgment). So my real question is: I there actually any real performance difference between option 5 and option 6?

return (

<div className="App">

<h1>

This is a Demo showing several ways to implement Conditional Rendering in React.

</h1>

{type === 1 && <button>Logout1</button>}

{type === 2 && <button>Logout2</button>}

{type === 3 && <button>Logout3</button>}

{type === 4 && <button>Logout4</button>}

{type === 5 && <button>Logout5</button>}

</div>

);

#################################################################

return (

<div className="App">

<h1>

This is a Demo showing several ways to implement Conditional Rendering in React.

</h1>

{(function() {

if (type === 1) {

return <button>Logout1</button>;

} else if (type === 2) {

return <button>Logout2</button>;

} else if (type === 3) {

return <button>Logout3</button>;

} else if (type === 4) {

return <button>Logout4</button>;

} else if (type === 5) {

return <button>Logout5</button>;

}

})()}