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:

* Showing or hiding elements.
* Toggling application functionality.

We will examine different ways to implement conditional rendering in React applications.

. 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.

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:

**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.

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.

Condition ? if cond is true : if condition is false

Using Ternary Operators

Expression ? true : false

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.

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.

**Using conditions with logical && operator**

We can use the logical && operator along with some condition to decide what will appear in output based on whether the condition evaluates to true or false. Below is the syntax of using the logical && operator with conditions: 

{

*condition* &&

// This section will contain

// elements you want to return

// that will be a part of output

}

If the *condition* provided in the above syntax evaluates to True then the elements right after the && operator will be a part of the output and if the condition evaluates to false then the code within the curly braces will not appear in the output.

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**// Example Component**

**function Example()**

**{**

**const counter = 5;**

**return(<div>**

**{**

**(counter==5) &&**

**<h1>Hello World!</h1>**

**}**

**</div>**

**);**

**}**

**ReactDOM.render(**

**<Example />,**

**document.getElementById('root')**

**);**

**Preventing Component from Rendering**

It might happen sometimes that we may not want some components to render. To prevent a component from rendering we will have to return *null* as its rendering output. Consider the below example:   
Open your react project directory and edit the**Index.js**file from src folder:

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**// Example Component**

**function Example(props)**

**{**

**if(!props.toDisplay)**

**return null;**

**else**

**return <h1>Component is rendered</h1>;**

**}**

**ReactDOM.render(**

**<div>**

**<Example toDisplay = {true} />**

**<Example toDisplay = {false} />**

**</div>,**

**document.getElementById('root')**

**);**

There may arise a situation when we want to render something based on some condition. For example, consider the situation of handling a login/logout button. Both the buttons have different functions so they will be separate components. Now, the task is if a user is logged in then we will have to render the Logout component to display the logout button and if the user is not logged in then we will have to render the Login component to display the login button. This is what we call **Conditional Rendering** in ReactJS. That is to create multiple components and render them based on some conditions. This is also a kind of encapsulation supported by React.   
Let us now create a page in React which will have a Message and a Button. The button will read “Login” if the user is not logged in and “Logout” if the user is logged in. We will also add some functionality to this button as upon clicking “Login” the message will read “Welcome User” and the button will change to “Logout” and upon clicking “Logout” the message will change to “Please Login” and the button will change to “Login”.  
To do this, we will create a parent component named “Homepage”, two components named “Login” and “Logout” and one more component named “Message”. We will use a state variable named “isLoggedIn” to store the information about whether the user is logged in or not. The value of this variable will change according to the click of the button by the user. The *Homepage* component will render the *Message* component to display the message and it will also render one of the components among *Login* and *Logout* based on the value stored in *isLoggedIn*. The *Message* component will also return different messages based on the value of state *isLoggedIn*.

See Second Demo for this