Day 8

# Conditional Rendering In React

- if
- else
- ? :
- &&
- switch

@oluwakemi Oluwadahunsi

# Introduction

Conditional rendering in React refers to the practice of **displaying different UI** elements or components based on a specific condition.

It means rendering different elements or components based on whether certain conditions are met.

In React, this is typically done using JavaScript operators like if, else, &&, ? :, or even the switch statement to decide what should be displayed on the screen.

For example, consider showing a "**Login**" button if the user is not logged in and a "**Logout**" button if the user is logged in. This is a common use case for conditional rendering.

@oluwakemi Oluwadahunsi

# Why Use Conditional Rendering?

- **Dynamic Content:** Display different content based on user interaction, application state, or data fetched from an API.
- **Enhanced User Experience:** Show or hide UI elements, such as loading spinners, error messages, or different views, based on the app's state.
- **Reusable Components:** Render components only when needed, making your app more efficient and maintainable.

@oluwakemi Oluwadahunsi

# Methods for Conditional Rendering in React

## 1. Using if-else Statements:

This is the simplest way to render components conditionally. The if-else statement checks a condition and renders a component or element accordingly.

```jsx
import React from 'react';

const UserGreeting = ({ isLoggedIn }) => {
  if (isLoggedIn) {
    return <h1>Welcome back, User!</h1>;
  } else {
    return <h1>Please sign in.</h1>;
  }
};

export default UserGreeting;
```

In this example, if the **isLoggedIn** prop is **true**, the message **"Welcome back, User!"** will be displayed. Otherwise, it will display **"Please sign in."**

@oluwakemi Oluwadahunsi

## 2. Using the Ternary Operator (? : ):

The ternary operator provides a concise way to render one of two components or elements based on a condition.

```jsx
import React from 'react';

const UserGreeting = ({ isLoggedIn }) => {
  return (
    <div>
      {isLoggedIn ? <h1>Welcome back, User!</h1> : <h1>Please sign in.</h1>}
    </div>
  );
};

export default UserGreeting;

```

The above codes works exactly like that of the **if-else** method, but the **ternary operator ( ? :)** approach is useful when the condition is simple and straightforward.

It's a compact way to handle conditional rendering without needing to use if-else statements.

@oluwakemi Oluwadahunsi

## 3. Using Logical AND (&&) Operator:

The **logical && operator** can also be used to render components conditionally. This method is particularly useful when you want to render a component **only when a condition is true**.

```jsx
function Mailbox(props) {
  const unreadMessages = props.unreadMessages;
  return (
    <div>
      <h1>Hello!</h1>
      {unreadMessages.length > 0 && (
        <h2>You have {unreadMessages.length} unread messages.</h2>
      )}
    </div>
  );
}

// Usage
<Mailbox unreadMessages={['React', 'Re: React', 'Re:Re: React']} />
```

In this example:

- If **unreadMessages.length > 0** is true, the **message count** is displayed.
- If **unreadMessages.length > 0** is false, nothing is rendered for that block.

@oluwakemi Oluwadahunsi

## 4. Using switch Statements:

For more complex conditions, you might prefer using **switch statements**. This approach is less common but can be useful when there are **multiple** conditions to check.

```jsx
import React from 'react';

const StatusMessage = ({ status }) => {
  switch (status) {
    case 'loading':
      return <p>Loading...</p>;
    case 'success':
      return <p>Data loaded successfully!</p>;
    case 'error':
      return <p>Error loading data!</p>;
    default:
      return <p>Unknown status.</p>;
  }
};

export default StatusMessage;
```

The switch statement allows for handling multiple conditions more explicitly and is useful when dealing with **several different possible states**.

@oluwakemi Oluwadahunsi

## 5. Using Immediately Invoked Function Expressions (IIFE):

In certain scenarios, you might need more control over the rendering logic. Using an **IIFE** allows you to write more complex logic directly in your JSX.

```jsx
import React from 'react';

const UserStatus = ({ isOnline }) => {
  return (
    <div>
      {(() => {
        if (isOnline) return <p>User is online</p>;
        if (!isOnline) return <p>User is offline</p>;
        return <p>Status unknown</p>;
      })()}
    </div>
  );
};

export default UserStatus;
```

This method is less common but can be useful when you want to keep the conditional logic more contained within a specific part of your component.

@oluwakemi Oluwadahunsi

## 6. Conditional Rendering with Hooks

With the advent of Hooks in React, you can manage conditional rendering effectively within functional components. You can use hooks like **useState** and **useEffect** to manage the state and side effects of your conditions.

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

function ToggleMessage() {
  const [showMessage, setShowMessage] = useState(false);

  const handleClick = () => setShowMessage(!showMessage);

  return (
    <div>
      <button onClick={handleClick}>
        {showMessage ? 'Hide Message' : 'Show Message'}
      </button>
      {showMessage && <p>This is a toggled message!</p>}
    </div>
  );
}

// Usage
<ToggleMessage />
```

In the above example:

- The **useState** hook is used to manage the **showMessage** state.
- The **button** toggles the state, and the **message** is conditionally rendered based on the **showMessage** value.

# Conditional Rendering in Class Components

While functional components with Hooks are becoming more popular, you may still need to handle conditional rendering in class components if that's what is required.

Example:

```jsx
class Greeting extends React.Component {
  constructor(props) {
    super(props);
    this.state = { isLoggedIn: false };
  }

  toggleLogin = () => {
    this.setState((prevState) => ({
      isLoggedIn: !prevState.isLoggedIn,
    }));
  };

  render() {
    return (
      <div>
        {this.state.isLoggedIn ? <h1>Welcome back!</h1> : <h1>Please sign up.</h1>}
        <button onClick={this.toggleLogin}>
          {this.state.isLoggedIn ? 'Logout' : 'Login'}
        </button>
      </div>
    );
  }
}

// Usage
<Greeting />
```

@oluwakemi Oluwadahunsi

In this example:

- The **isLoggedIn** state is managed within the class component.

- A button toggles the **isLoggedIn** state, and the component renders the appropriate message based on the current state.

# A Basic Example

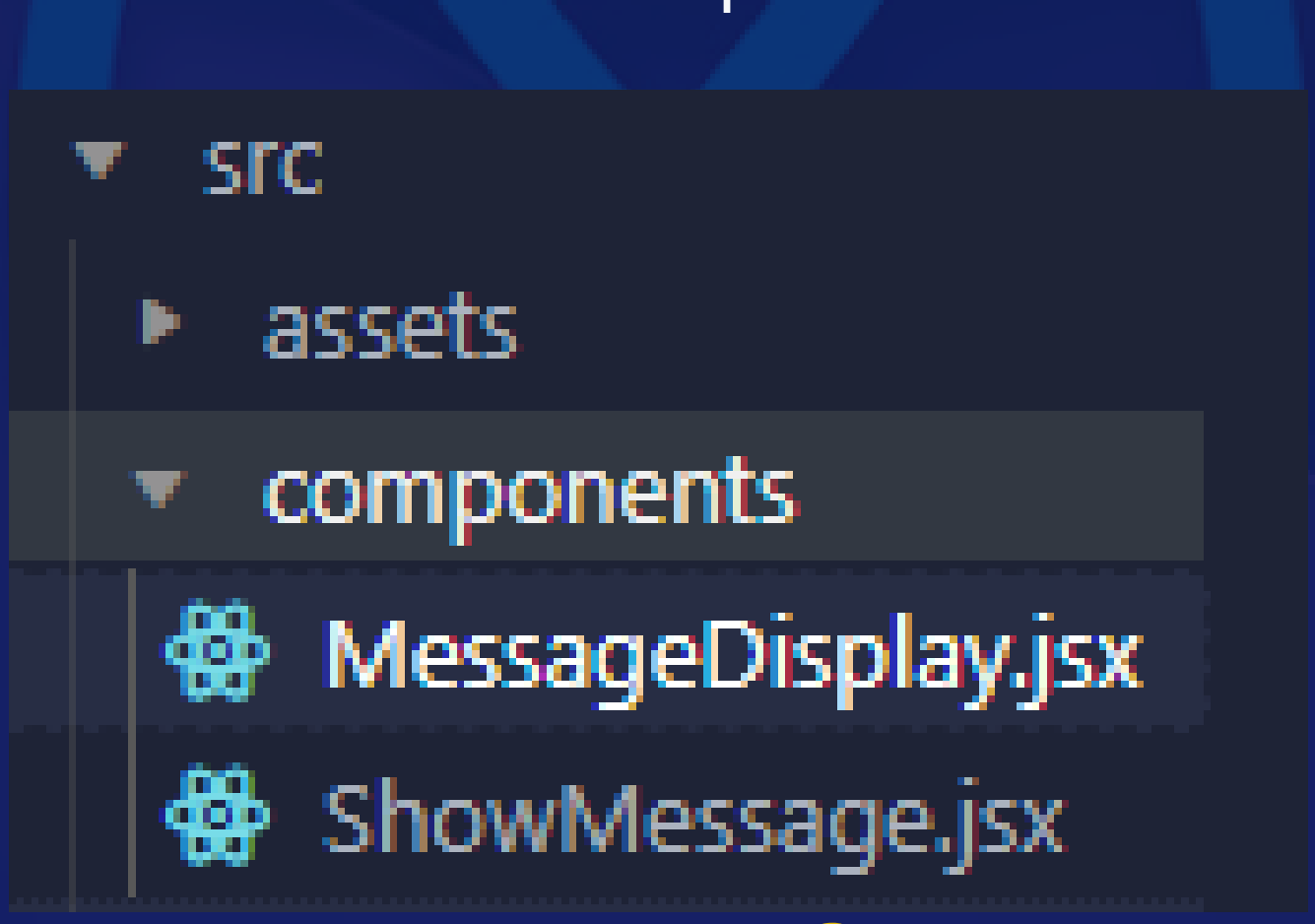
In our React application, let's create two components in the components folder: **ShowMessage** and **MessageDisplay**.

When a button in the **MessageDisplay** component is clicked, it renders the **ShowMessage** component.

The **ShowMessage** component contains a close button that, when clicked, renders the **MessageDisplay** component again.

Now let's do this step by step:

Step 1: Create the two components like this



Add the following codes to your **ShowMessage.jsx** component:

```
// src/components/ShowMessage.js

import React from 'react';

function ShowMessage({ onClose }) {
  return (
    <div>
      <p>This is the message content!</p>
      <button onClick={onClose}>Close</button>
    </div>
  );
}

export default ShowMessage;
```

**ShowMessage** Component:

- Displays a message (<p> tag).
- Contains a "**Close**" button that triggers the **onClose** function passed as a prop from the parent (**MessageDisplay**) component.

@oluwakemi Oluwadahunsi

Add the following codes to your **MessageDisplay.jsx** component:

```jsx
// src/components/MessageDisplay.js
import ShowMessage from './ShowMessage';

function MessageDisplay() {
  const [isMessageShown, setIsMessageShown] = useState(false); // State to control message visibility

  // Function to handle showing the message
  const handleShowMessage = () => {
    setIsMessageShown(true); // Set state to show message
  };

  // Function to handle closing the message
  const handleCloseMessage = () => {
    setIsMessageShown(false); // Set state to hide message
  };

  return (
    <div>
      {/* Conditional Rendering: ShowMessage or Button */}
      {isMessageShown ? (
        <ShowMessage onClose={handleCloseMessage} />
      ) : (

       <div>
          <p>Click to show message!</p>
          <button onClick={handleShowMessage}>Show Message</button>
        </div>
      )}
    </div>
  );
}

export default MessageDisplay;
```

@oluwakemi Oluwadahunsi

What the **MessageDisplay** componenet does:

- Maintains state (**isMessageShown**) to determine whether to show the message or the button.

- **handleShowMessage:** Sets **isMessageShown** to true to display the **ShowMessage** component.

- **handleCloseMessage:** Sets **isMessageShown** to **false** to hide the **ShowMessage** component and display the "**Show Message**" button again.

- Uses a **ternary operator** for conditional rendering: If **isMessageShown** is true, renders **ShowMessage** component; otherwise, renders the **div** with "**Show Message**" button.

@oluwakemi Oluwadahunsi

Call the **MessageDisplay** component in your root (**App.jsx**) like this:

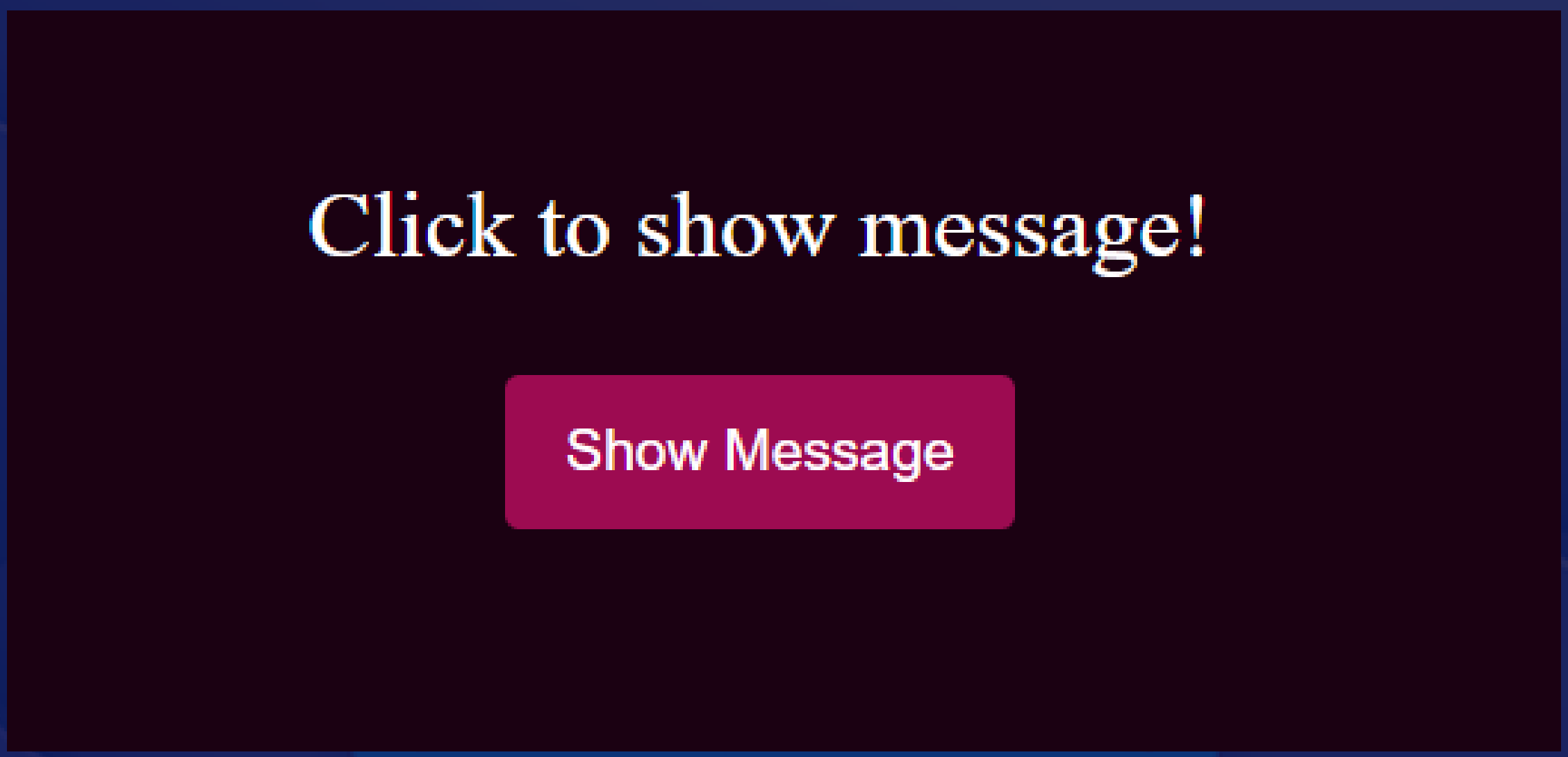
```jsx
import MessageDisplay from './components/MessageDisplay'


function App() {
  return (
    <>
      <MessageDisplay />
    </>
  )
}
export default App
```

Now run “**npm run dev**” if you are using the **Vite** method or “**npm start**” if using **create react app (CRA)** method to view how this works in your browser.

Click on the provided URL and view in the browser

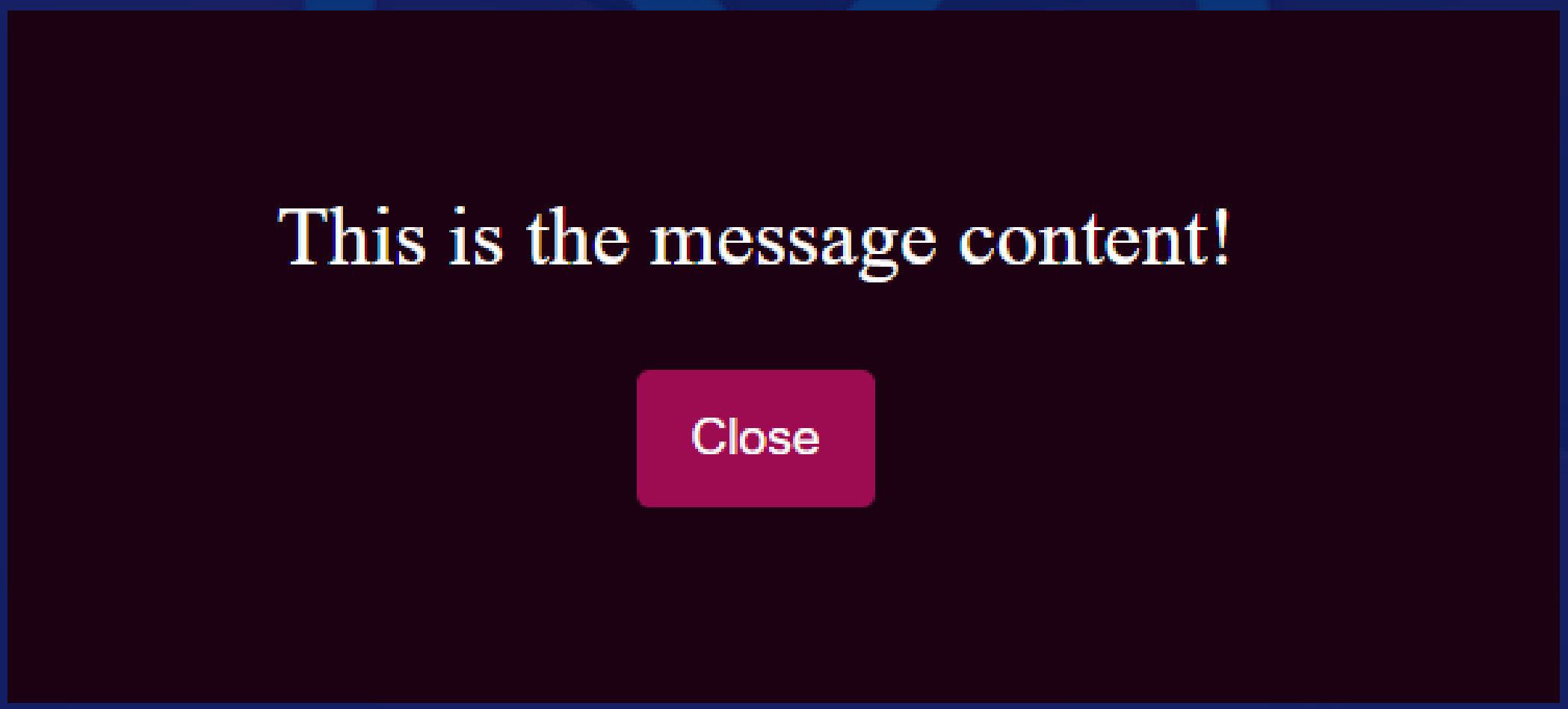
You should have it work like this in your UI:

@oluwakemi Oluwadahunsi




**PS: I added a little styling like colors and sizes for more visibility.**

When the "**Show Message**" button is clicked, the **handleShowMessage** function is called, setting **isMessageShown** to **true**. This triggers the rendering of the **ShowMessage** component.



The **ShowMessage** component receives the **onClose** prop, which is called when the "**Close**" button is clicked.

This function sets **isMessageShown** back to **false**, causing the **MessageDisplay** component to render the div with the "**Show Message**" button again.

# Best Practices for Conditional Rendering In React

- **Keep It Simple:** Use simple conditions wherever possible to keep your code readable.

- **Avoid Deep Nesting:** Deeply nested conditional logic can make your code harder to understand and maintain.

- **Use Short-Circuit Evaluation:** For simple conditions, use the && operator instead of a ternary.

- **Extract to Functions or Components:** If your conditional logic is complex, consider extracting it into a helper function or a separate component to keep your code clean.

- **Leverage Hooks and Functional Components:** Take advantage of hooks like useState and useEffect for managing state and side effects.

@oluwakemi Oluwadahunsi

I hope you found this material useful and helpful.

Remember to:

Like

Save for future reference

&

Share with your network, be helpful to someone 👌

@oluwakemi Oluwadahunsi

# Hi There!

## Thank you for reading through

Did you enjoy this knowledge?

💼 Follow my LinkedIn page for more work-life balancing and Coding tips.

🌐 LinkedIn: Oluwakemi Oluwadahunsi



kodemaven-portfolio.vercel.app