### Exercise 12

### 1. Conditional Rendering in React

Conditional rendering is a core concept in React that allows you to display different elements or components based on certain conditions. It's how you make your user interface dynamic and responsive to changes in state, props, or other data.

You can implement conditional rendering in a few different ways:

* **if statements:** You can use a standard JavaScript if statement to return different JSX based on a condition. This is a common pattern when the logic is more complex or you need to return an entire component early.  
  JavaScript  
  function Greeting(props) {  
    if (props.isLoggedIn) {  
      return <UserGreeting />;  
    }  
    return <GuestGreeting />;  
  }
* **Logical && operator:** This is a concise way to conditionally render an element when a condition is true. Since true && expression evaluates to expression, and false && expression evaluates to false, React will render the element on the right-hand side if the condition on the left is true.  
  JavaScript  
  function Notifications({ unreadMessages }) {  
    return (  
      <div>  
        <h1>Hello!</h1>  
        {unreadMessages.length > 0 && <h2>You have {unreadMessages.length} unread messages.</h2>}  
      </div>  
    );  
  }
* **Ternary operator (condition ? true : false):** This is a simple and common way to choose between two different outcomes. It's often used for inline conditional rendering within JSX.  
  JavaScript  
  function LoginButton({ isLoggedIn }) {  
    return (  
      <button>  
        {isLoggedIn ? 'Logout' : 'Login'}  
      </button>  
    );  
  }
* **switch statements:** For multiple, complex conditions, you can use a switch statement outside of your JSX to determine which component or element to render.

### 2. Element Variables

Element variables are a technique used in conditional rendering where you declare a variable to hold a JSX element or component. You can then use if or else statements to assign a different element to that variable based on a condition. Finally, you render the variable itself within your component's return statement.

This approach is particularly useful for making your code more readable when you have complex conditional logic that would otherwise clutter your JSX.

**Example:**

JavaScript

function LoginControl({ isLoggedIn }) {  
  let button;  
  if (isLoggedIn) {  
    button = <LogoutButton />;  
  } else {  
    button = <LoginButton />;  
  }  
  
  return (  
    <div>  
      <p>Please log in to continue.</p>  
      {button}  
    </div>  
  );  
}

In this example, the button variable is assigned a different component depending on the isLoggedIn prop, and then the variable is rendered in the JSX.

### 3. How to Prevent Components from Rendering

In some cases, you may want a component to completely hide itself, even though its parent component is attempting to render it. The simplest way to prevent a component from rendering is to have it return null.

When a component's render method (or functional component's return) returns null, React will not render anything in its place.

**Example:**

JavaScript

function WarningBanner({ warn }) {  
  // If the 'warn' prop is false, the component will not render.  
  if (!warn) {  
    return null;  
  }  
  
  return (  
    <div className="warning">  
      Warning!  
    </div>  
  );  
}

By returning null, you effectively "hide" the component from the DOM. This is a common pattern for things like loading spinners, warning messages, or empty states that should only appear under specific conditions.

**App.js**

import React, { useState } from 'react';

import './App.css';

function GuestGreeting(props) {

  return (

    <div className="center-content">

      <h2>Please sign up.</h2>

      <LoginButton onClick={props.onLoginClick} />

    </div>

  );

}

function UserGreeting(props) {

  return (

    <div className="center-content">

      <h2>Welcome back</h2>

      <LogoutButton onClick={props.onLogoutClick} />

    </div>

  );

}

function LoginButton(props) {

  return (

    <button onClick={props.onClick}>

      Login

    </button>

  );

}

function LogoutButton(props) {

  return (

    <button onClick={props.onClick}>

      Logout

    </button>

  );

}

function App() {

  const [isLoggedIn, setIsLoggedIn] = useState(false);

  const handleLoginClick = () => {

    setIsLoggedIn(true);

  };

  const handleLogoutClick = () => {

    setIsLoggedIn(false);

  };

  let greetingComponent;

  if (isLoggedIn) {

    greetingComponent = <UserGreeting onLogoutClick={handleLogoutClick} />;

  } else {

    greetingComponent = <GuestGreeting onLoginClick={handleLoginClick} />;

  }

  return (

    <div className="App">

      <header className="App-header">

        <h1>Ticket Booking App</h1>

        <p>A simple demonstration of conditional rendering in React.</p>

      </header>

      <main>

        {greetingComponent}

      </main>

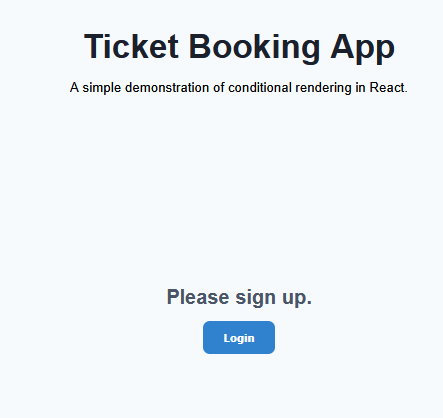
    </div>

  );

}

export default App;

**Output:**

****

