**Conditional Rendering in React**

**Introduction**

Conditional rendering in React allows developers to dynamically control what content is displayed based on the application's state or props. React efficiently updates the UI by rendering only the necessary components when a condition changes.

**Why Use Conditional Rendering?**

* Improves user experience by showing relevant content.
* Avoids unnecessary UI elements, reducing complexity.
* Efficiently manages UI state changes.

**Methods of Conditional Rendering**

**1. Using if-else Statements**

The classic way to conditionally render elements in React.

**Example:**

import React, { useState } from "react";

export default function UserGreeting() {

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

const handleLogin = () => setIsLoggedIn(true);

const handleLogout = () => setIsLoggedIn(false);

if (isLoggedIn) {

return (

<div>

<h1>Welcome, User!</h1>

<button onClick={handleLogout}>Logout</button>

</div>

);

} else {

return (

<div>

<h1>Please log in</h1>

<button onClick={handleLogin}>Login</button>

</div>

);

}

}

**2. Using Ternary Operator (? :)**

A shorter way to conditionally render components.

**Example:**

return (

<div>

<p>{isLoggedIn ? "Welcome, User!" : "Please log in"}</p>

<button onClick={isLoggedIn ? handleLogout : handleLogin}>

{isLoggedIn ? "Logout" : "Login"}

</button>

</div>

);

**3. Using Logical && Operator**

Best when rendering UI **only when a condition is true**.

**Example:**

return (

<div>

{isLoggedIn && <h1>Welcome, User!</h1>}

<button onClick={handleLogout}>Logout</button>

</div>

);

**4. Using Switch Case (For Complex Conditions)**

Useful when there are multiple rendering conditions.

**Example:**

const userRole = "admin";

switch (userRole) {

case "admin":

return <h1>Admin Dashboard</h1>;

case "user":

return <h1>User Dashboard</h1>;

default:

return <h1>Guest Page</h1>;

}

**Advanced Conditional Rendering**

**1. Rendering Components Conditionally**

Instead of inline JSX, you can return different components.

**Example:**

function Greeting({ isLoggedIn }) {

return isLoggedIn ? <UserDashboard /> : <GuestPage />;

}

**2. Conditional Styling**

You can change styles dynamically based on conditions.

**Example:**

const buttonStyle = isLoggedIn ? { backgroundColor: "green" } : { backgroundColor: "red" };

return <button style={buttonStyle}>Click Me</button>;

**Exercises**

**1. Basic Conditional Rendering**

Modify the UserGreeting component to display a different message based on user roles ("admin", "user", "guest").

**2. Show/Hide Element**

Create a button that toggles the visibility of a paragraph (<p>Hello, World!</p>).

**3. Multi-Condition Rendering**

Create a React component that renders different messages based on weather conditions ("sunny", "rainy", "cloudy").

**4. Form Validation Message**

Build a login form that shows an error message when the user tries to submit without filling both email and password fields.

**Conclusion**

Conditional rendering is a powerful technique that enables dynamic UI updates in React applications. By mastering different methods, you can create interactive and user-friendly applications efficiently.

**Introduction to React Router 🚀**

**What is React Router?**

React Router is a powerful **client-side routing library** for React applications. It enables seamless navigation between different views or pages **without refreshing the browser**, making React apps feel like **single-page applications (SPAs)**.

**🔹 Why Use React Router?**

✅ Enables **dynamic navigation** between different components.  
✅ Supports **nested routes** and **protected routes** (e.g., authentication).  
✅ Allows **URL parameter handling** (useful for dynamic content).  
✅ Provides **history management** (back, forward navigation).

**🔹 Installing React Router**

Before using React Router, install it in your project:

sh

CopyEdit

npm install react-router-dom

or

sh

CopyEdit

yarn add react-router-dom

**🔹 Basic Example: Setting Up Routes**

Create a simple app with **Home**, **About**, and **Contact** pages.

**📌 Step 1: Setup App.js**

jsx

CopyEdit

import React from "react";

import { BrowserRouter as Router, Route, Routes, Link } from "react-router-dom";

import Home from "./Home";

import About from "./About";

import Contact from "./Contact";

export default function App() {

return (

<Router>

<nav>

<Link to="/">Home</Link> |

<Link to="/about">About</Link> |

<Link to="/contact">Contact</Link>

</nav>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/about" element={<About />} />

<Route path="/contact" element={<Contact />} />

</Routes>

</Router>

);

}

**📝 Explanation**

* **<Router>**: Wraps the entire app to enable routing.
* **<Link>**: Navigation links to different routes **without refreshing** the page.
* **<Routes>**: Defines all the route paths.
* **<Route>**: Specifies which component should render at each URL.

**📌 Step 2: Create Home.js, About.js, and Contact.js**

**🏠 Home Component**

jsx

CopyEdit

export default function Home() {

return <h2>Welcome to the Home Page!</h2>;

}

**ℹ️ About Component**

jsx

CopyEdit

export default function About() {

return <h2>About Us</h2>;

}

**📞 Contact Component**

jsx

CopyEdit

export default function Contact() {

return <h2>Contact Us</h2>;

}

**🔹 Navigating with useNavigate()**

Instead of <Link>, you can navigate programmatically using **useNavigate()**.

**Example: Redirect to Home after Clicking a Button**

jsx

CopyEdit

import { useNavigate } from "react-router-dom";

export default function About() {

const navigate = useNavigate();

return (

<div>

<h2>About Page</h2>

<button onClick={() => navigate("/")}>Go to Home</button>

</div>

);

}

**🔹 Dynamic Routing with URL Parameters (useParams)**

**Example: Displaying a User Profile Page**

jsx

CopyEdit

import { useParams } from "react-router-dom";

export default function Profile() {

const { username } = useParams();

return <h2>Welcome, {username}!</h2>;

}

**Route Setup in App.js**

jsx

CopyEdit

<Route path="/profile/:username" element={<Profile />} />

**How to Access This Page?**

Go to **/profile/Yohannes**, and it will display:

CopyEdit

Welcome, Yohannes!

**🔹 Protected Routes (Authentication)**

A **protected route** restricts access based on conditions (e.g., user login status).

**Example: Restrict Access to Dashboard**

jsx

CopyEdit

import { Navigate } from "react-router-dom";

export default function Dashboard({ isLoggedIn }) {

return isLoggedIn ? <h2>Welcome to Dashboard</h2> : <Navigate to="/" />;

}

**Route Setup**

jsx

CopyEdit

<Route path="/dashboard" element={<Dashboard isLoggedIn={true} />} />

If isLoggedIn is **false**, the user is redirected to **Home**.

**🎯 Conclusion**

✅ React Router enables **smooth client-side navigation**.  
✅ Use <Link> or useNavigate() for **navigation**.  
✅ Use useParams() for **dynamic routing**.  
✅ Use Navigate for **protected routes**.

Would you like an **exercise** to practice React Router? 🚀