Easy Explanation

REACT JS

YOU NEED FOR INTERVIEW

SYED SHAAZ AKHTAR

Introduction to React.js

👉 React ek JavaScript library hai jo Facebook ne develop ki thi, taaki fast aur interactive user interfaces ban sakein.

👉 React ka main use hota hai Single Page Applications (SPA) banane me!

👉 React me hum components ka use karke UI ko chhoti-chhoti independent files me tod sakte hain!

Real-Life Example:

Socho tum LEGO ke blocks se ek ghar bana rahe ho

- Har block ek component hai!
- Tum blocks ko reusable bana sakte ho!
- Agar ek block ka design change karna ho, toh poore ghar ko rebuild karne ki zaroorat nahi!

Installing & Setting Up React.js

👉 React install karne ke liye hum Vite ka use karenge, jo React ko fast aur optimized rakhta hai!

📌 Step 1: React App Create Karo

```
npm create vite@latest my-react-app --template react
```

Yeh ek naye React project ka setup kar dega!

📌 Step 2: Project Folder Me Enter Karo

```
cd my-react-app
```

📌 Step 3: Dependencies Install Karo

```
npm install
```

📌 Step 4: Development Server Start Karo

```
npm run dev
```

Ab tumhara React app chalne ke liye ready hai!

What is a Single Page Application (SPA)?

👉 SPA ek aisi website hoti hai jo bina full-page reload kare naye content ko load kar sakti hai!

👉 Sirf body ka content reload hota hai, pura page nahi!

Real-Life Example:

Socho tum YouTube chal rahe ho

- Tum "Movies" tab pe ho, phir "Songs" tab pe click karte ho
- Page reload nahi hota, sirf body ka content change hota hai!

React isliye popular hai kyunki ye fast SPAs banata hai!

What is a Component in React?

👉 Component ek independent, reusable code ka tukda hota hai jo UI ka ek part represent karta hai.

👉 React me har UI element ek component ho sakta hai (button, navbar, footer, form, etc.)

Real-Life Example:

Socho tum Netflix ka UI design kar rahe ho!

- Ek component ho sakta hai "Movie Card"
- Ek component ho sakta hai "Navbar"
- Ek component ho sakta hai "Search Bar"

Example: Creating a Simple Component

Step 1: RedButton.jsx Component Banao

```
import React from 'react';


function RedButton() {
  return
<button style={{ backgroundColor: 'red', color: 'white' }}>
Click Me
</button>;
}
export default RedButton;
```

Step 2: App.jsx Me Component Use Karo

```
import React from 'react';
import RedButton from './RedButton';

function App() {
  return (
    <div>
      <h1>Welcome to My App</h1>
      <RedButton /><RedButton />
</div>
  );
}
export default App;
```

Ab tumne ek reusable component bana diya, jo kahin bhi use ho sakta hai!

What is JSX? (JavaScript XML)

👉 JSX ek syntax extension hai jo HTML ko JavaScript me likhne ki suvidha deta hai.

👉 JSX ka use React me UI components likhne ke liye hota hai.

Real-Life Example:

Socho tum React likh rahe ho bina JSX ke!

- Tumhe JavaScript me document.createElement() use karna padega har element ke liye!
- JSX isse easy bana deta hai, jaise hum normally HTML likhte hain!

Example: JSX Ka Use

```
import React from 'react';
function App() {
  const name = 'Shaaz';
  return
 <h1>Hello, {name}!</h1>;
}

export default App;
```

JSX me curly brackets {} ke andar JavaScript likh sakte hain!

Rules of JSX

1. Har JSX Code Ko Ek Parent Element Me Wrap Karna Zaroori Hai!

```
function App() {
  return (
    <div>
      <h1>Hello, World!</h1>
      <p>Welcome to React!</p>
</div>
  );
}
```

Har JSX return statement ek hi parent element ke andar hona chahiye!

2. JSX Me if-else Ke Jagah Ternary Operator Ka Use Karein

❌ **Wrong:**

```
function App({ isLoggedIn }) {
  if (isLoggedIn) {
    return
      <h1>Welcome Back!</h1>;
  } else {
    return
     <h1>Please Sign In</h1>;
  }
}
```

✅ Correct (Using Ternary Operator):

```
function App({ isLoggedIn }) {
 return (
 <div>
 {isLoggedIn ? <h1>Welcome Back!</h1> : <h1>Please Sign
In</h1>}
 </div>
 );
}
```

Ternary Operator React me zyada readable hota hai!

Short Summary

✓ React ek JavaScript library hai jo interactive user interfaces banane ke liye use hoti hai.

✓ React me hum Single Page Applications (SPA) banate hain jo fast aur smooth hoti hain.

✓ Component ek reusable UI piece hota hai jo baar-baar use kiya ja sakta hai.

✓ JSX ka use React me HTML likhne ke liye hota hai, jo JavaScript ke andar likha ja sakta hai.

✓ JSX ke kuch rules hote hain, jaise single parent element aur ternary operator ka use.

Props & Prop Drilling

What are Props in React?

👉 Props ka full form hota hai "Properties"

👉 Props ka use ek component se doosre component me data bhejne ke liye hota hai

👉 Props immutable hote hain – iska matlab hai ki component props ko change nahi kar sakta!

Real-Life Example:

Socho tum Pizza Order kar rahe ho

- Tum Domino's ko order doge (Props pass karoge)
- Domino's sirf wahi pizza bana ke dega jo tumne order kiya hai!

Matlab, props sirf data bhejne ka kaam karte hain, aur component use kar sakta hai par modify nahi kar sakta!

Props Kaise Use Karein?

👉 Props function components me directly {} ke andar receive hote hain.

👉 Props parent component se child component me pass hote hain.

Example: Props Ka Basic Use

Step 1: User.jsx Component Banao

```
import React from 'react';
function User({ name, age }) {
  return (
    <div>
      <h1>Name: {name}</h1>
      <p>Age: {age}</p>
    </div>
  );
}
export default User;
```

✅ Props ko curly brackets {} me destructure kiya gaya!

Step 2: App.jsx Me Props Pass Karo

```
import React from 'react';
import User from './User';
function App() {
  return <User name="Shaaz" age={24} />;
}

export default App;
```

✅ Ab name="Shaaz" aur age={24} User component me pass ho raha hai!

✅ React automatically User component me ye values bhej dega!

Example: Multiple Props Pass Karna

```
function User({ name, age, hobby }) {
  return (
    <div>
      <h1>Name: {name}</h1>
      <p>Age: {age}</p>
      <p>Hobby: {hobby}</p>
    </div>
  );
}

function App() {
  return
<User name="Shaaz" age={24} hobby="Coding" />;
}

export default App;
```

✅ Ab ek aur prop hobby add kar diya jo User component receive karega!

Props Immutable Hote Hain (Change Nahi Kar Sakte)

👉 Props ko child component ke andar modify nahi kar sakte!

❌ **Wrong:**

```
function User({ name, age }) {
  name = "Asad"; // ❌ Props ko modify nahi kar sakte!
  return <h1>{name}</h1>;
}
```

✅ **Correct:**

```
function User({ name }) {
  return <h1>{name}</h1>; // ✅ Props sirf read kar sakte hain!
}
```

Props ek baar set ho jaye, toh child component usko change nahi kar sakta!

What is Prop Drilling?

👉 Prop Drilling ka matlab hai ek prop ko multiple components ke through pass karna, jabki uska use sirf ek deeply nested child component me ho.

👉 Jitne beech ke components hain, wo bas props ko aage pass karte hain bina use kiye!

Real-Life Example:

Socho tum Apne Papa se Paise maang rahe ho!

- Papa ne Dadaji se maanga!
- Dadaji ne ParDadaji se maanga!
- ParDadaji ne Tumhe finally de diya!

Example: Prop Drilling Problem

```
function GreatGrandParent() {
  const message = "Hello from Great Grandparent!";
  return <GrandParent message={message} />;
}

function GrandParent({ message }) {
  return <Parent message={message} />;
}

function Parent({ message }) {
  return <Child message={message} />;
}

function Child({ message }) {
  return <h1>{message}</h1>;
}

export default GreatGrandParent;
```

Things to remember ?

- message ko har component manually pass kar raha hai!
- Agar 10 components ho, toh ye aur zyada complex ho jayega!

Prop Drilling Ka Solution

👉 Prop drilling ko avoid karne ke liye context API ya state management tools ka use karte hain!

👉 Context API ka use karke direct data pass kar sakte hain bina beech ke components ko involve kiye!

Solution: React Context API

```
import React, { createContext, useContext } from 'react';

// Step 1: Create Context
const MessageContext = createContext();

function GreatGrandParent() {
  const message = "Hello from Great Grandparent!";
  return (
    <MessageContext.Provider value={message}>
<GrandParent /></MessageContext.Provider>
  );
}

function GrandParent() {
  return <Parent />;
}

function Parent() {
  return <Child />;
}
```

Short Summary

✓ Props ek component se doosre component me data pass karne ke liye use hote hain.

✓ Props immutable hote hain, child component inhe modify nahi kar sakta.

✓ Prop Drilling ek problem hai jisme props ko unnecessary middle components se pass karna padta hai.

✓ Context API ka use karke direct data access kiya ja sakta hai bina prop drilling ke!

Conditional Rendering

What is Conditional Rendering in React?

👉 Matlab kisi condition ke base pe component ya element show ya hide karna!

👉 Agar koi user logged in hai toh "Welcome!" dikhana, warna "Please Log In" dikhana.

👉 React me hum multiple tareeke se Conditional Rendering kar sakte hain!

Real-Life Example:

Socho tum Netflix chal rahe ho!

- Agar tum logged in ho, toh "Continue Watching" wala section dikh raha hoga.
- Agar tum logged out ho, toh "Sign In to Continue" likha aayega.

Matlab, UI dynamically change ho rahi hai based on login status!

If/Else Statement (Sabse Basic Tarika)

👉 Yeh approach simple hai, lekin multiple return statements use karta hai jo best practice nahi hai!

Example: Using If/Else

```
function Greeting({ isLoggedIn }) {
  if (isLoggedIn) {
    return <h1>Welcome Back!</h1>;
  } else {
    return <h1>Please Log In</h1>;
  }
}

function App() {
  return <Greeting isLoggedIn={true} />;
}

export default App;
```

✅ Pros: Basic & Easy to Understand!

❌ Cons: Multiple return statements hone ki wajah se readability reduce hoti hai!

Using Element Variables

👉 Isme ek variable me JSX store karke use karte hain!

👉 Thoda better hai if/else se, par ab bhi best nahi hai!

Example: Using Element Variables

```
function Greeting({ isLoggedIn }) {
  let message;
  if (isLoggedIn) {
    message = <h1>Welcome Back!</h1>;
  } else {
    message = <h1>Please Log In</h1>;
  }

  return <div>{message}</div>;
}

function App() {
  return <Greeting isLoggedIn={false} />;
}

export default App;
```

✅ Pros: Ek hi return hai, readability improve hoti hai!

❌ Cons: Extra variable message banana padta hai, jo unnecessary lag sakta hai!

Using Ternary Operator (Recommended Approach)

👉 **Ternary Operator ? :** ka use karke hum ek concise aur readable tarika bana sakte hain!

👉 Ye ek hi line me conditional rendering kaam kar deta hai! 🚀

Example: Using Ternary Operator

```
function Greeting({ isLoggedIn }) {
  return (
    <h1>{isLoggedIn ? "Welcome Back!" : "Please Log In"}
    </h1>
  );
}

function App() {
  return <Greeting isLoggedIn={true} />;
}

export default App;
```

✅ Pros: Short aur zyada readable code!

✅ Cons: Sirf simple conditions ke liye best hai, complex cases me readability lose ho sakti hai!

Using Short Circuit Operator (&&)

👉 Agar bas ek hi condition check karni hai bina else part ke, toh yeh best hai!

👉 true && <Component /> sirf tabhi render hoga jab condition true ho!

Example: Using Short Circuit (&&)

```
function Notification({ unreadMessages }) {
  return (
    <div>
<h1>Welcome to React!</h1>
     {unreadMessages.length > 0 &&
<p>You have {unreadMessages.length} new messages.
</p>}
    </div>
  );
}

function App() {
  return <Notification unreadMessages={["Message 1",
"Message 2"]} />;
}

export default App;
```

✅ Pros: Short aur effective jab else ka zaroorat na ho!

❌ Cons: Kabhi-kabhi code readability thodi down ho sakti hai!

Nested Conditions (Multiple Conditions Handle Karna)

👉 Agar tumhe ek se zyada condition check karni ho, toh nested ternary ka use kar sakte ho!

Example: Multiple Conditions

```
function Status({ isLoggedIn, isAdmin }) {
  return (
    <h1>
      {isLoggedIn ? (isAdmin ? "Welcome, Admin!" : "Welcome, User!") : "Please Log In"}
    </h1>
  );
}

function App() {
  return <Status isLoggedIn={true} isAdmin={false} />;
}

export default App;
```

✅ Pros: Ek hi line me multiple conditions handle ho sakti hain!

❌ Cons: Zyada complex ho jaye toh readability khatam ho sakti hai!

Conditional Rendering Ke Best Practices

✔ Agar sirf ek true condition me render karna hai, toh && use karo.

✔ Agar if-else wali situation hai, toh ternary operator ? : use karo.

✔ Agar multiple nested conditions hai, toh readability ka dhyan rakho! 🏆

Short Summary

✔ Conditional Rendering React me UI dynamically change karne ke liye hoti hai.

✔ If/Else ka use kam karo, kyunki multiple return readability ko decrease karta hai.

✔ Ternary Operator ? : recommended hai, kyunki concise aur readable hota hai.

✔ Short Circuit Operator && tab use karo jab sirf ek condition pe render karna ho.

✔ Complex conditions ke liye readability ka dhyan rakho, warna confusion badh sakta hai!

Lifting State Up

What is Lifting State Up in React?

👉 Agar ek parent ke multiple child components ek hi state ka use karte hain, toh state ko parent me rakhna best practice hota hai!

👉 Isse sabhi child components ek centralized state se data le sakte hain, aur parent state ko update kar sakta hai!

👉 Yeh concept React me data sharing aur state management easy banata hai!

Real-Life Example:

Socho tum ek classroom me teacher ho!

- Tumhare paas ek whiteboard hai jo sab students dekh sakte hain.
- Agar kisi student ko kuch likhna ho, toh wo teacher ko bolega, aur teacher whiteboard pe likhega.
- Iska matlab teacher (parent component) ek common source hai jo sab students (child components) access kar rahe hain!

Why Lift State Up?

👉 Agar do ya zyada components ek hi state ka use kar rahe hain, toh us state ko parent me rakhna chahiye.

👉 Agar ek component kisi doosre component ko data bhejna chahta hai, toh bhi state parent me rakhna best practice hai. 🚀

💡 Example:

- Ek input box hai jisme user text enter karega.
- Ek alag component hai jo real-time me wahi text show karega.
- Agar input ka data ek state me store hoga, toh dono components easily uska use kar sakenge.

Example of Lifting State Up

👉 Ek parent component hoga jo state maintain karega.

👉 Ek child component hoga jo user se input lega.

👉 Dono components ek hi state ka use karenge jo parent ke control me hogi.

📌 **Step 1: Parent Component – App.jsx**

```
import React, { useState } from "react";
import InputComponent from "./InputComponent";
import DisplayComponent from "./DisplayComponent";

function App() {
  const [text, setText] = useState("");

  // Function to update state from child
const handleInputChange = (newText) => {
    setText(newText);
  };
  return (
    <div>
<h1>Lifting State Up Example</h1>
<InputComponent onInputChange={handleInputChange} />
<DisplayComponent text={text} />
  </div>
  );
}

export default App;
```

✅ Yahan text state ko parent me rakha gaya hai taaki sab child components use kar sakein.

Step 2: Child Component – InputComponent.jsx

```
import React from "react";


function InputComponent({ onInputChange }) {
  return (
    <div>
<inputtype="text"placeholder="Type
something..."onChange={(e) =>
onInputChange(e.target.value)}
    />
    </div>
  );
}

export default InputComponent;
```

✅ Yahan onInputChange function ko prop ke through parent se child me bheja gaya hai!

📌 **Step 3: Child Component – DisplayComponent.jsx**

```
import React from "react";


function DisplayComponent({ text }) {
  return <h2>Typed Text: {text}</h2>;
}

export default DisplayComponent;
```

✅ Yahan text prop ke through parent se child me data bheja gaya hai!

How Lifting State Up Works?

✅ Parent (App.jsx) me ek state text maintain ho rahi hai.

✅ InputComponent.jsx user ka input le raha hai aur onInputChange function call karke parent ko data bhej raha hai.

✅ Parent me setText() function se text update ho raha hai.

✅ Updated text ko DisplayComponent.jsx me bheja ja raha hai jo screen pe render kar raha hai.

Matlab sab kuch centralized state se control ho raha hai!

Another Example: Counter App with Multiple Buttons

👉 Agar multiple buttons se ek hi counter value ko control karna ho, toh lifting state up ka use hota hai!

Example: Counter App

Step 1: Parent Component (App.jsx)

```
import React, { useState } from "react";
import IncrementButton from "./IncrementButton";
import DecrementButton from "./DecrementButton";
import DisplayCounter from "./DisplayCounter";

function App() {
  const [count, setCount] = useState(0);

  return (
    <div>
<h1>Counter App</h1>
<DisplayCounter count={count} />
<IncrementButton increment={() => setCount(count + 1)} />
    <DecrementButton decrement={() => setCount(count -
1)} />
    </div>
  );
}
export default App;
```

✅ Yahan count state ko parent me rakha gaya hai jo sab buttons use karenge!

Step 2: Display Counter Component (DisplayCounter.jsx)

```
import React from "react";

function DisplayCounter({ count }) {

  return <h2>Counter: {count}</h2>;

}

export default DisplayCounter;
```

Step 3: Increment Button (IncrementButton.jsx)

```
import React from "react";

function IncrementButton({ increment }) {

  return <button onClick={increment}>Increment</button>;

}

export default IncrementButton;
```

Step 4: Decrement Button (DecrementButton.jsx)

```
import React from "react";

function DecrementButton({ decrement }) {

  return <button onClick={decrement}>Decrement</button>;

}

export default DecrementButton;
```

Ab parent (App.jsx) ke paas centralized count state hai, jo IncrementButton aur DecrementButton dono update kar sakte hain!

Benefits of Lifting State Up

✓ Centralized State: Ek jagah pe state manage hoti hai, toh debugging easy hoti hai!

✓ Data Sharing: Multiple components easily same state ka use kar sakte hain.

✓ Performance Optimization: Agar state sirf ek jagah ho, toh unnecessary re-renders avoid ho sakte hain!

📌 **Short Summary**

✓ Lifting State Up ka matlab hai state ko ek common parent component me rakhna, taaki multiple components uska use kar sakein.

✓ State change sirf parent component me hota hai, aur child components usko props ke through use karte hain.

✓ Props se data parent-child components ke beech flow hota hai.

✓ Agar multiple components ek hi state ka use kar rahe hain, toh lifting state up best practice hoti hai!

Benefits of Lifting State Up

✓ Centralized State: Ek jagah pe state manage hoti hai, toh debugging easy hoti hai!

✓ Data Sharing: Multiple components easily same state ka use kar sakte hain.

✓ Performance Optimization: Agar state sirf ek jagah ho, toh unnecessary re-renders avoid ho sakte hain!

📌 **Short Summary**

✓ Lifting State Up ka matlab hai state ko ek common parent component me rakhna, taaki multiple components uska use kar sakein.

✓ State change sirf parent component me hota hai, aur child components usko props ke through use karte hain.

✓ Props se data parent-child components ke beech flow hota hai.

✓ Agar multiple components ek hi state ka use kar rahe hain, toh lifting state up best practice hoti hai!

React Hooks

What are Hooks in React?

👉 Hooks ek tarika hai jisse functional components me state aur lifecycle methods ka use kiya jata hai.

👉 Hooks se functional components zyada dynamic aur re-usable bante hain.

👉 Hooks ka naam hamesha "use" se start hota hai, jaise useState, useEffect, useRef etc.

Real-Life Example:

Socho tum gaming khel rahe ho , aur tumhare pass cheat codes hain jo game easy bana dete hain!

- Hooks bhi React ke "cheat codes" hain jo functional components ko zyada powerful banate hain!

useState() – React ka Sabse Important Hook!

👉 useState ka use component me state manage karne ke liye hota hai!

👉 Jab bhi state update hoti hai, component automatically re-render hota hai!

💡 Real-Life Example:

Socho tum Instagram pe ho!

- Agar tum "Like ❤️" button dabaoge, toh instantly count update ho jata hai!
- Yeh magic "state" ki wajah se hota hai!

Example: Simple Counter App

```
import React, { useState } from "react";


function Counter() {
  const [count, setCount] = useState(0);


  return (
    <div>
      <h2>Count: {count}</h2>
    <button
       onClick={() => setCount(count + 1)}>
        Increment
     </button>
    <button
      onClick={() => setCount(count - 1)}>Decrement
    </button>
  </div>
  );
}

export default Counter;
```

Har baar button click hone par count update hoga aur UI re-render hoga!

useEffect() – Lifecycle Hook!

👉 useEffect ka use side effects (jaise API calls, timers, events) handle karne ke liye hota hai!

👉 Class components me componentDidMount, componentDidUpdate, componentWillUnmount hota tha, par hooks me sab useEffect se handle hota hai!

```
import React, { useState, useEffect } from "react";
function FetchData() {
  const [data, setData] = useState([]);

  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/posts")
      .then((response) => response.json())
      .then((json) => setData(json));
  }, []);

  return (
    <div>
      <h2>Fetched Data:</h2>
      <ul>
        {data.slice(0, 5).map((item) => (
          <li key={item.id}>{item.title}</li>
        ))}
      </ul>
    </div>
  );
}
export default FetchData;
```

useRef() – DOM Reference Hold Karne Ka Hook!

👉 useRef se tum kisi bhi DOM element ya variable ka reference hold kar sakte ho bina re-render kare!

```
import React, { useRef } from "react";

function FocusInput() {
  const inputRef = useRef(null);

  const handleFocus = () => {
   inputRef.current.focus();
  };

  return (
   <div>
    <input ref={inputRef}
      type="text" placeholder="Type something..." />
    <button onClick={handleFocus}>Focus Input</button>
   </div>
  );
}

export default FocusInput;
```

useRef se input field ka reference store kiya aur bina re-render kare usko focus kiya!

useReducer() – useState Ka Advanced Version!

👉 useReducer tab use hota hai jab state ka logic complex ho!

```
import React, { useReducer } from "react";

const initialState = 0;

const reducer = (state, action) => {
  switch (action.type) {
    case "increment":
      return state + 1;
    case "decrement":
      return state - 1;
    default:
      return state;
  }
};

function Counter() {
  const [count, dispatch] = useReducer(reducer, initialState);
  return (
    <div>
      <h2>Count: {count}</h2>
      <button onClick={() => dispatch({ type: "increment"
})}>Increment</button>
      <button onClick={() => dispatch({ type: "decrement"
})}>Decrement</button>
    </div>
  );
}
export default Counter;
```

useReducer() – useState Ka Advanced Version!

👉 useReducer tab use hota hai jab state ka logic complex ho!

```
import React, { useReducer } from "react";

const initialState = 0;

const reducer = (state, action) => {
  switch (action.type) {
    case "increment":
      return state + 1;
    case "decrement":
      return state - 1;
    default:
      return state;
  }
};

function Counter() {
  const [count, dispatch] = useReducer(reducer, initialState);
  return (
    <div>
      <h2>Count: {count}</h2>
      <button onClick={() => dispatch({ type: "increment"
})}>Increment</button>
      <button onClick={() => dispatch({ type: "decrement"
})}>Decrement</button>
    </div>
  );
}

export default Counter;
```

✅ Yahan dispatch() function se actions bhej rahe hain, jo reducer() handle karega!

useMemo() – Performance Optimization Hook!

👉 useMemo() expensive calculations optimize karta hai taaki baar-baar calculate na karna pade!

Example: Expensive Calculation Optimization

```
import React, { useState, useMemo } from "react";


function ExpensiveCalculation({ num }) {
  const slowFunction = (num) => {
    console.log("Running Slow Function...");
    return num * 2;
  };

  const memoizedValue = useMemo(() => slowFunction(num),
[num]);

  return <h2>Result: {memoizedValue}</h2>;
}

export default ExpensiveCalculation;
```

✅ useMemo() function unnecessary re-runs avoid karega!

useCallback() – Function Memoization Hook!

👉 useCallback() function ko cache karta hai, jisse unnecessary re-renders avoid hote hain!

Example: Preventing Unnecessary Re-renders

```
import React, { useState, useCallback } from "react";


function Button({ handleClick }) {
  return <button onClick={handleClick}>Click Me</button>;
}

function App() {
  const [count, setCount] = useState(0);

  const memoizedFunction = useCallback(() => {
    setCount((prev) => prev + 1);
  }, []);

  return (
    <div><h2>Count: {count}</h2>
      <Button handleClick={memoizedFunction} />
    </div>
  );
}

export default App;
```

✅ useCallback() function ko cache karega taaki unnecessary re-renders na ho!

Custom Hooks

What are Custom Hooks in React?

👉 Custom Hooks ek tarika hai apni logic ko reusable banane ka!

👉 Agar kisi logic ka use multiple jagah hota hai, toh usko ek hook me likh ke baar-baar reuse kar sakte hain!

👉 Custom Hooks ek normal function ki tarah hote hain, but inka naam use se start hota hai (jaise useFetch, useLocalStorage etc.)

💡 **Real-Life Example:**

Socho tum ek YouTube Video Editor ho!

- Har video pe tum "Subscribe Now!" ka watermark lagate ho.
- Agar tum ye manually har video me karoge, toh bahut time lagega! ⏳
- Ek automated tool (custom hook) bana lo jo ye kaam kare, toh mast ho jayega!

Custom Hooks bhi isi tarah React me code ko optimize aur reusable banate hain!

Why Use Custom Hooks?

👉 Agar ek hi logic multiple components me repeat ho rahi hai, toh usko ek custom hook me likho aur reuse karo!

👉 Ye code ko clean aur maintainable banata hai!

👉 Ekbaar likhne ke baad usko har jagah use kar sakte ho bina copy-paste kiye!

Creating a Custom Hook

👉 Ek custom hook ek normal function ki tarah hota hai jo React Hooks ka use karta hai.

👉 Iska naam hamesha use se start hota hai, jaise useFetch, useTheme, useCounter etc.

Example: useCounter Hook (Re-usable Counter Logic)

Step 1: Custom Hook Banao – useCounter.js

```
import { useState } from "react";

function useCounter(initialValue = 0) {
  const [count, setCount] = useState(initialValue);

  const increment = () => setCount(count + 1);
  const decrement = () => setCount(count - 1);
  const reset = () => setCount(0);

  return { count, increment, decrement, reset };
}

export default useCounter;
```

✅ Yeh ek custom hook hai jo counter logic handle karega!

✅ Isme useState ka use kiya hai aur count, increment, decrement, reset return kiya hai!

Step 2: Custom Hook Ka Use Karo – App.jsx

```
import React from "react";
import useCounter from "./useCounter";

function App() {
  const { count, increment, decrement, reset } =
useCounter(10);

  return (
    <div>
<h1>Custom Counter Hook</h1>
<h2>Count: {count}</h2>
<button onClick={increment}>Increment</button>
<button onClick={decrement}>Decrement</button><button
onClick={reset}>Reset</button></div>
  );
}

export default App;
```

✅ Ab useCounter() ko har jagah use kar sakte hain bina same code likhe!

Custom Hook for Fetching API Data (useFetch)

👉 Agar multiple components API se data fetch kar rahe hain, toh us logic ko ek custom hook me likho!

📌 **Step 1: useFetch.js – Custom Hook**

```
import { useState, useEffect } from "react";


function useFetch(url) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
   async function fetchData() {
    try {
     let response = await fetch(url);
     let result = await response.json();
     setData(result);
     setLoading(false);
    } catch (err) {
     setError(err);
     setLoading(false);
    }
   }
   fetchData();
  }, [url]);

  return { data, loading, error };
}
export default useFetch;
```

✅ Yeh hook kisi bhi API se data fetch karega aur data, loading, error return karega!

Step 2: App.jsx – Hook Ko Use Karo

```
import React from "react";
import useFetch from "./useFetch";

function App() {
  const { data, loading, error } = useFetch(
    "https://jsonplaceholder.typicode.com/posts"
  );

  if (loading) return <h2>Loading...</h2>;
  if (error) return <h2>Error fetching data!</h2>;

  return (
    <div>
<h1>Fetched Data:</h1>
<ul>
      {data.slice(0, 5).map((item) => (
        <li key={item.id}>{item.title}</li>
      ))}
</ul>
    </div>
  );
}

export default App;
```

✅ Ab har jagah useFetch() use karke API fetch kar sakte ho bina extra code likhe!

Custom Hook for Local Storage (useLocalStorage)

👉 Agar tumhe data ko browser ke local storage me save karna ho, toh har jagah localStorage.setItem likhna padega! 😵

👉 Isko avoid karne ke liye ek custom hook useLocalStorage bana lo!

📌 **Step 1: useLocalStorage.js**

```
import { useState } from "react";


function useLocalStorage(key, initialValue) {
  const [storedValue, setStoredValue] = useState(() => {
    const savedValue = localStorage.getItem(key);
    return savedValue ? JSON.parse(savedValue) :
initialValue;
  });

  const setValue = (value) => {
    setStoredValue(value);
    localStorage.setItem(key, JSON.stringify(value));
  };

  return [storedValue, setValue];
}

export default useLocalStorage;
```

✅ Yeh hook local storage se data save/retrieve karega!

Step 2: App.jsx – Hook Ko Use Karo

```
import React from "react";
import useLocalStorage from "./useLocalStorage";

function App() {
  const [name, setName] = useLocalStorage("username", "");

  return (
    <div>
<h1>Local Storage Hook</h1>
<inputtype="text"
placeholder="Enter your name"
value={name}
onChange={(e) => setName(e.target.value)}
      />
      <p>Saved Name: {name}</p></div>
  );
}
```

export default App;

✅ Ab browser ko refresh karne ke baad bhi input field me value rahegi!

Benefits of Custom Hooks

✔ Code Reusability: Ek hi logic ko multiple jagah use kar sakte ho!

✔ Code Cleanliness: Har baar same code likhne ki zaroorat nahi!

✔ Performance Optimization: Hooks unnecessary re-renders avoid karne me madad karte hain!

✔ Better Readability: Code chhota aur zyada readable ho jata hai!

📌 **Short Summary**

✔ Custom Hooks ek tarika hai apni logic ko reusable aur maintainable banane ka.

✔ Har Custom Hook ka naam use se start hota hai, jaise useCounter, useFetch, useLocalStorage etc.

✔ Agar ek logic multiple jagah use ho rahi hai, toh usko ek custom hook me likho!

Context API

What is Context API in React?

👉 Context API ek tarika hai jisse ek global state create karke multiple components ke beech data share kiya jata hai, bina props drill kiye!

👉 Ye ek React ke built-in feature hai, jo Redux jaise external state management tools ka lightweight alternative hai!

💡 Real-Life Example:

Socho tum McDonald's 🍔 me ho, aur ek billing counter hai jo sab orders process karta hai!

- Agar har ek table pe ek separate billing machine ho, toh confusion badh jayegi!
- Isliye ek centralized counter (Context API) hota hai, jo sab tables (components) ke liye orders process karta hai!

Matlab Context API ek centralized store hai jo har component ke liye easily accessible hota hai!

Problem Without Context API (Prop Drilling)

👉 Agar ek deeply nested component ko data chahiye, toh hume parent se child, child se uske child tak props pass karna padta hai!

Example: Prop Drilling Problem

```
function App() {
  const user = "Shaaz";

  return <Parent user={user} />;
}

function Parent({ user }) {
  return <Child user={user} />;
}

function Child({ user }) {
  return <GrandChild user={user} />;
}

function GrandChild({ user }) {
  return <h2>Welcome, {user}!</h2>;
}

export default App;
```

✅ Har component me user ko manually props ke through pass karna pad raha hai!

⚠ **Problem:**

- Agar humare components aur deeply nested hote, toh prop drilling aur jyada complex ho jata!

Solution: Context API!

👉 Context API se ek baar global store bana do, fir koi bhi component us data ko directly access kar sakta hai, bina prop drilling kiye!

📌 **Step 1: Create a Context (Global Store)**

```
import React, { createContext } from "react";
// Context banaya
const UserContext = createContext();
export default UserContext;
```

✅ Ye ek global state hai jo har component ke liye accessible hogi!

📌 **Step 2: Wrap Components Inside a Provider**

```
import React, { useState } from "react";
import UserContext from "./UserContext";
import Child from "./Child";

function App() {
  const [user, setUser] = useState("Shaaz");
  return (
    <UserContext.Provider value={user}>
    <Child />
    </UserContext.Provider>
  );
}
export default App;
```

✅ Ab UserContext.Provider ke andar jitne bhi components honge, wo user value ko access kar sakenge!

Step 3: Access Context in Child Components

```
import React, { useContext } from "react";
import UserContext from "./UserContext";


function Child() {
  const user = useContext(UserContext);
  return <h2>Welcome, {user}!</h2>;
}
export default Child;
```

✅ Ab useContext(UserContext) se bina props pass kiye user ko access kar sakte hain!

Updating Context Value (Theme Switcher Example)

👉 Context API sirf data share karne ke liye nahi, balki state update karne ke liye bhi use hota hai!

👉 Ek toggleTheme function bana kar, hum dark/light mode switch kar sakte hain! 🚀

Step 1: Create Theme Context

```
import React, { createContext, useState } from "react";
const ThemeContext = createContext();

export function ThemeProvider({ children }) {
  const [theme, setTheme] = useState("light");

  const toggleTheme = () => {
    setTheme(theme === "light" ? "dark" : "light");
  };

  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
}
export default ThemeContext;
```

✅ Yeh ek global theme context hai jo sab components ko theme aur toggleTheme function provide karega!

Step 2: Wrap App Inside ThemeProvider

```
import React from "react";
import { ThemeProvider } from "./ThemeContext";
import ThemeSwitcher from "./ThemeSwitcher";


function App() {
  return (
    <ThemeProvider><ThemeSwitcher /></ThemeProvider>
  );
}

export default App;
```

✅ Ab har component theme aur toggleTheme function use kar sakta hai!

Step 3: Access Theme Context & Toggle Theme

```
import React, { useContext } from "react";
import ThemeContext from "./ThemeContext";

function ThemeSwitcher() {
  const { theme, toggleTheme } =
useContext(ThemeContext);

  return (
    <div style={{backgroundColor: theme === "light" ? "#fff" :
"#333",
      color: theme === "light" ? "#000" : "#fff",
      padding: "20px",
      textAlign: "center"
    }}>
    <h2>Current Theme: {theme}</h2>
    <button onClick={toggleTheme}>Toggle Theme</button>
</div>
  );
}
export default ThemeSwitcher;
```

✅ Ab button click karne pe theme switch ho jayega!

When to Use Context API?

✔ Jab multiple components ko same data chahiye (jaise user info, theme, language settings).

✔ Jab prop drilling ka problem ho aur usko avoid karna ho.

✔ Jab Redux ya koi aur external state management tool use na karna chahein.

📌 Benefits of Context API

✔ No More Prop Drilling: Parent-child ka lamba chain pass karne ki zaroorat nahi!

✔ Global State Management: Ek jagah se pura app ka state manage ho sakta hai!

✔ Cleaner Code: Code chhota aur readable ho jata hai!

✔ Optimized Performance: Har baar unnecessary props pass nahi hote, toh re-renders kam hote hain!

📌 Short Summary

✔ Context API ek tarika hai data ko globally store karke multiple components ke beech share karne ka.

✔ createContext() se context banate hain, aur Provider ke andar wrap karke value dete hain.

✔ useContext() se kisi bhi component me bina prop drilling ke data access kar sakte hain.

✔ Context API ko themes, authentication, user preferences, aur state sharing ke liye use kiya jata hai.

React Forms

What are Forms in React?

👉 Forms ka use user se input collect karne ke liye hota hai!

👉 HTML forms me data directly DOM se liya jata hai, but React me state ke through manage hota hai!

👉 Forms ko efficiently handle karne ke liye React me do approaches hain – Controlled aur Uncontrolled Components!

Real-Life Example:

Socho tum Pizza Order Kar Rahe Ho!

- Tumhe Name, Address, Pizza Type aur Payment Details bharni hai.
- Agar tumne form submit kiya aur wo data lost ho gaya, toh order cancel ho sakta hai!
- Isliye React Forms me data ko state me store kiya jata hai, taaki kahin bhi lost na ho!

Controlled Components – State Se Form Control Karo!

👉 Controlled components wo hote hain jo React state se fully controlled hote hain.

👉 Matlab jo bhi input field me likhoge, wo React state me store hoga aur React ke control me rahega.

Example: Simple Controlled Input Form

```
import React, { useState } from "react";

function ControlledForm() {
  const [name, setName] = useState("");

  const handleChange = (event) => {
    setName(event.target.value);
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`Hello, ${name}!`);
  };
```

```
 return (
 <form onSubmit={handleSubmit}>
<label>Name: </label>
<input type="text" value={name} onChange={handleChange}
/>
<button type="submit">Submit</button>
</form>
 );
}
export default ControlledForm;
```

✅ Yahan name ka value state me store ho raha hai, aur user jo bhi type karega, wo update hoga!

✅ onChange event state ko update kar raha hai, aur onSubmit se alert show ho raha hai!

Handling Multiple Input Fields in Forms

👉 Agar form me multiple fields hain (jaise email, password, address), toh har ek ke liye useState nahi banana chahiye!

👉 Instead, ek single object state bana ke usko dynamically update karna better hota hai!

Example: Multi-Field Controlled Form

```
import React, { useState } from "react";

function MultiFieldForm() {
  const [formData, setFormData] = useState({
    name: "",
    email: "",
    password: "",
  });

  const handleChange = (event) => {
    setFormData({ ...formData, [event.target.name]:
event.target.value });
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`Welcome, ${formData.name}! Your email is
${formData.email}`);
  };
```

```
return (
 <form
onSubmit={handleSubmit}>
<label>Name: </label>
<input type="text" name="name" value={formData.name}
onChange={handleChange} />

<label>Email: </label>
<input type="email" name="email" value={formData.email}
onChange={handleChange} />

<label>Password: </label>
<input type="password" name="password" value=
{formData.password}
onChange={handleChange} />

<button type="submit">Submit</button>
</form>
 );
}
export default MultiFieldForm;
```

✅ Ek single state object me name, email aur password store ho raha hai!

✅ handleChange function dynamically values update kar raha hai using [event.target.name]!

Uncontrolled Components – DOM Se Direct Data Lo!

👉 Uncontrolled components me form elements ka state React se manage nahi hota, balki directly DOM se data fetch hota hai!

👉 Isme useRef() ka use karke input field ka reference liya jata hai.

Example: Uncontrolled Form Using useRef

```
import React, { useRef } from "react";

function UncontrolledForm() {
  const nameRef = useRef(null);

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`Hello, ${nameRef.current.value}!`);
  };

  return (
    <form
      onSubmit={handleSubmit}>
      <label>Name: </label>
      <input type="text" ref={nameRef} />
      <button type="submit">Submit</button>
    </form>
  );
}
export default UncontrolledForm;
```

✅ useRef() se input field ka reference leke value fetch ho rahi hai!

✅ React state use nahi ho rahi, directly DOM se data fetch ho raha hai!

Difference Between Controlled & Uncontrolled Components

✓ **Controlled Components:** Form data React state me store hota hai!

✓ **Uncontrolled Components:** Data directly DOM se fetch hota hai, React state ka use nahi hota!

✓ Controlled Components jyada predictable aur recommended hote hain, kyunki data React ke control me rehta hai! 🔥

Handling Checkbox, Radio & Select Dropdown

👉 Forms me sirf text fields nahi hote, balki checkboxes, radio buttons aur dropdowns bhi hote hain!

Example: Checkbox & Radio Button Handling

```
import React, { useState } from "react";


function PreferencesForm() {
  const [preferences, setPreferences] = useState({
    newsletter: false,
    gender: "",
  });

const handleChange = (event) => {
  const { name, type, checked, value } = event.target;
  setPreferences({
    ...preferences,
    [name]: type === "checkbox" ? checked : value,
  });
};
```

```
return (
 <form>
<label>
<input type="checkbox" name="newsletter" checked=
{preferences.newsletter} onChange={handleChange} />
 Subscribe to Newsletter
 </label>

<label>
<input type="radio" name="gender" value="male" checked=
{preferences.gender === "male"} onChange={handleChange} />
 Male
 </label>

<label>
<input
type="radio" name="gender" value="female" checked=
{preferences.gender === "female"}
onChange={handleChange} />
 Female
 </label>

<p>
Newsletter: {preferences.newsletter ? "Subscribed" : "Not
Subscribed"}
</p>
<p>Gender: {preferences.gender}</p></form>
 );
}
export default PreferencesForm;
```

✅ Checkbox ke liye checked ka use kiya hai, aur radio button ke liye value check kiya hai!

Short Summary

✓ Controlled Components: React state se manage hote hain, better for data consistency!

✓ Uncontrolled Components: useRef() se data directly DOM se fetch hota hai!

✓ Multiple Fields Handling: Ek single state object bana ke sab fields manage karo!

✓ Checkbox, Radio aur Select Elements bhi easily handle ho sakte hain!

React Form Registration

Registration Form Logic – Step-by-Step Approach

User se Input Collect Karo – Name, Email, Password, Confirm Password

Form Ko State Ke Saath Connect Karo – useState() ka use karke

Validation Lagao – Koi bhi field empty na rahe, aur email/password sahi ho

Submit Form & Show Data – Data ko process karke console ya alert me dikhayein

Creating a Registration Form in React

👉 Sabse pehle ek simple form banayenge jo user se Name, Email, aur Password lega!

Example: Basic Registration Form

```
import React, { useState } from "react";

function RegistrationForm() {
  const [formData, setFormData] = useState({
    name: "",
    email: "",
    password: "",
    confirmPassword: "",
  });

  const handleChange = (event) => {
    const { name, value } = event.target;
    setFormData({ ...formData, [name]: value });
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`Welcome, ${formData.name}! Your email is
${formData.email}`);
  };
```

```
return (
 <form
    onSubmit={handleSubmit}>

   <label>Name: </label>
   <input type="text" name="name" value={formData.name}
onChange={handleChange} required />

   <label>Email: </label>
   <input type="email" name="email" value={formData.email}
onChange={handleChange} required />

   <label>Password: </label>
   <input type="password" name="password" value=
{formData.password} onChange={handleChange} required />
      <label>Confirm Password: </label>
   <input type="password" name="confirmPassword" value=
{formData.confirmPassword} onChange={handleChange}
required />

  <button type="submit">Register</button>
</form>
 );
}
export default RegistrationForm;
```

✅ Yahan har input field ka value state me store ho raha hai, aur onChange se update ho raha hai! 🔄

✅ Form submit hone par alert me naam aur email show ho raha hai!

Adding Basic Form Validations

👉 Ab validation lagayenge taaki user galat data na bhare!

👉 Yeh check karenge ki:

✔ Koi bhi field empty na ho

✔ Email format sahi ho (@ aur . hona chahiye)

✔ Password aur Confirm Password match karein

Example: Form with Validations

```
import React, { useState } from "react";


function RegistrationForm() {
  const [formData, setFormData] = useState({
    name: "",
    email: "",
    password: "",
    confirmPassword: "",
  });

  const [errors, setErrors] = useState({});


const validate = () => {
  let newErrors = {};


  if (!formData.name.trim()) {
    newErrors.name = "Name is required!";
  }
```

```
if (!formData.email.includes("@") ||
!formData.email.includes("."))
 { newErrors.email = "Enter a valid email!"; }

 if (formData.password.length < 6)
{ newErrors.password = "Password must be at least 6
characters!";
}

if (formData.password !== formData.confirmPassword) {
newErrors.confirmPassword = "Passwords do not match!";
}

setErrors(newErrors);
return Object.keys(newErrors).length === 0; };

 const handleChange = (event) => {
setFormData({ ...formData, [event.target.name]:
event.target.value });
};

const handleSubmit = (event) =>
{
event.preventDefault();

if (validate()) {
alert(`Welcome, ${formData.name}! Your registration is
successful.`); }
 };
```

```
return (
<form
onSubmit={handleSubmit}>
<label>Name: </label>
<input type="text" name="name" value={formData.name}
onChange={handleChange} />
{errors.name && <p style={{ color: "red" }}>{errors.name}</p>}

<label>Email: </label>
<input type="email" name="email" value={formData.email}
onChange={handleChange} />
{errors.email && <p style={{ color: "red" }}>{errors.email}</p>}

<label>Password: </label>
<input type="password" name="password" value=
{formData.password}
onChange={handleChange} />
{errors.password && <p style={{ color: "red" }}>{errors.password}
</p>}

<label>Confirm Password: </label>
<input type="password" name="confirmPassword" value=
{formData.confirmPassword}
onChange={handleChange} />
{errors.confirmPassword && <p style={{ color: "red" }}>
{errors.confirmPassword}</p>}

<button type="submit">Register</button>
</form>
); }
```

Rendering a List

What is List Rendering in React?

👉 List rendering ka matlab hai kisi bhi array ke multiple items ko loop karke JSX me display karna!

👉 React me hum map() ka use karte hain jo har ek item ke liye ek JSX element return karta hai!

💡 Real-Life Example:

Socho tum IPL ke sabhi teams ka naam ek list me dikhana chahte ho! 🪚

- Har ek team ek h1 tag me dikhna chahiye!
- React me iske liye map() ka use karenge!

Basic List Rendering in React

👉 Sabse pehle ek simple example dekhte hain jisme IPL teams ka naam render karenge! ✅

👀 Example: Rendering List of IPL Teams

```
import React from "react";


function IPLTeams() {
  const IPL = ["Mumbai Indians", "Chennai Super Kings",
"Royal Challengers Bangalore", "Delhi Capitals"];


  return (
    <div>
      <h2>IPL Teams:</h2>
      {IPL.map((team, index) => (
        <h1 key={index}>{team}</h1>
      ))}
    </div>
  );
}


export default IPLTeams;
```

✅ map() se har ek IPL team ko ek h1 tag me render kar diya!

✅ Har element ko key={index} diya jo React ko batata hai ki ye ek unique item hai!

Using Key in React Lists

👉 React me key ek special attribute hota hai jo har list item ko unique banata hai!

👉 Agar key unique nahi hai, toh React ko properly DOM update karne me problem hoti hai! ⚠

Example: Agar keys same ho, toh kya hoga?

```
{IPL.map((team) => (
  <h1 key="1">{team}</h1>  // ⚠ Har ek element ka same key hone se React confused ho jayega!
))}
```

⚠ **Problem:**

- React ko ye samajhne me dikkat hogi ki kaunsa item change ho raha hai!
- Ye React ki performance aur rendering ko affect karega!

✓ Solution: Har item ko ek unique key do!

Example: Using Unique ID as Key (Best Practice)

```
import React from "react";


function IPLTeams() {
  const IPL = [
    { id: 1, name: "Mumbai Indians" },
    { id: 2, name: "Chennai Super Kings" },
    { id: 3, name: "Royal Challengers Bangalore" },
    { id: 4, name: "Delhi Capitals" },
  ];

  return (
    <div>
      <h2>IPL Teams:</h2>
      {IPL.map((team) => (
        <h1 key={team.id}>{team.name}</h1>
      ))}
    </div>
  );
}

export default IPLTeams;
```

✅ Har team ka ek unique id diya, jo key ke liye best practice hai!

Rendering a List of Objects (More Data)

👉 Agar tumhare array ke andar multiple properties ho (jaise team ka naam aur wins), toh kaise render karoge?

👀 **Example: Rendering List of IPL Teams with Wins**

```
import React from "react";

function IPLTeams() {
  const IPL = [
    { id: 1, name: "Mumbai Indians", wins: 5 },
    { id: 2, name: "Chennai Super Kings", wins: 5 },
    { id: 3, name: "Royal Challengers Bangalore", wins: 0 },
    { id: 4, name: "Delhi Capitals", wins: 0 },
  ];

  return (
    <div>
      <h2>IPL Teams & Wins:</h2><ul>
        {IPL.map((team) => (
          <li key={team.id}>
            {team.name} - 🏆 {team.wins} Titles
          </li>
        ))}
      </ul>
    </div>
  );
}

export default IPLTeams;
```

✅ Ab har team ka naam aur uske wins ko list (ul) ke andar show kiya!

Conditional Rendering with Lists

👉 Agar tumhe sirf wo teams dikhani ho jinme 1 se jyada wins hain, toh filter() ka use karo!

Example: Filtering and Rendering List

```
import React from "react";
function IPLWinners() {
  const IPL = [
    { id: 1, name: "Mumbai Indians", wins: 5 },
    { id: 2, name: "Chennai Super Kings", wins: 5 },
    { id: 3, name: "Royal Challengers Bangalore", wins: 0 },
    { id: 4, name: "Delhi Capitals", wins: 0 },
  ];

  const winners = IPL.filter((team) => team.wins > 0);

  return (
    <div>
      <h2>IPL Winners:</h2>
      <ul>
        {winners.map((team) => (
          <li key={team.id}>
            {team.name} - 🏆 {team.wins} Titles
          </li>
        ))}
      </ul>
    </div>
  );
}
export default IPLWinners;
```

✅ Ab sirf wo teams show ho rahi hain jinke pass wins > 0 hain!

Higher-Order Components (HOC)

What is a Higher-Order Component (HOC)?

👉 HOC ek function hota hai jo ek component ko input me leta hai, usme kuch extra functionality add karta hai, aur ek naya component return karta hai! 🔄

👉 Ye ek "Wrapper Function" ki tarah kaam karta hai jo kisi bhi component me extra features inject kar sakta hai! 🚀

👉 Agar tumhe kisi bhi component me reusability aur code optimization chahiye, toh HOC perfect solution hai!

Real-Life Example:

Socho tum ek restaurant 🍔 me gaye aur tumne ek "Regular Burger" order kiya.

- Agar tumko "Extra Cheese 🧀" chahiye, toh chef pura naya burger nahi banayega! 😲
- Chef tumhare burger me sirf extra cheese add karega! ✅
- Yahi kaam React me HOC karta hai – ek existing component me extra functionality add karta hai bina usko modify kiye!

Creating a Simple HOC

👉 Sabse pehle ek simple Higher-Order Component (HOC) banayenge jo ek component ko "border" provide karega! ✅

Example: Simple HOC

```
import React from "react";

// HOC Function jo kisi bhi component ko border provide karega
function withBorder(WrappedComponent) {
  return function EnhancedComponent(props) {
    return (
      <div
        style={{ border: "2px solid red", padding: "10px", margin: "10px" }}>
<WrappedComponent {...props} />
      </div>
    );
  };
}

// Normal Component
function Message(props) {
  return <h2>{props.text}</h2>;
}

// HOC ko apply kiya
const MessageWithBorder = withBorder(Message);
```

```
function App() {
 return (
 <div>
<MessageWithBorder text="Hello, This is a Higher-
Order Component!" />
</div>
 );
}

export default App;
```

✅ Yahan withBorder() ek HOC hai jo kisi bhi component ko ek red border dega!

✅ Message component ko wrap karke MessageWithBorder banaya, jo naye feature ke saath aayega!

HOC for Authentication (Protected Route)

👉 Jab tum ek website banate ho jisme kuch pages sirf logged-in users ke liye hote hain, toh HOC ka use kar sakte ho!

👉 Agar user logged in nahi hai, toh usko login page pe redirect kar sakte hain!

Example: HOC for Authentication

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

// HOC for Authentication
function withAuth(WrappedComponent) {
  return function AuthComponent(props) {
    const isAuthenticated = false; // Change this to true to simulate login
    if (!isAuthenticated) {
      return <Navigate to="/login" />;
    }
    return <WrappedComponent {...props} />;
  };
}

// Protected Page
function Dashboard() {
  return <h2>Welcome to Dashboard</h2>;
}
```

```
// Applying HOC
const ProtectedDashboard = withAuth(Dashboard);


function App() {
 return (
 <div>
<ProtectedDashboard />
</div>
 );
}

export default App;
```

✅ Agar isAuthenticated false hai, toh user login page pe redirect ho jayega!

✅ Agar true hai, toh Dashboard component show hoga!

HOC for Logging Component Renders

👉 Agar tumhe kisi bhi component ke render hone ka tracking rakhna ho, toh ek Logging HOC bana sakte ho!

Example: Logging HOC

```
import React, { useEffect } from "react";

// Logging HOC
function withLogger(WrappedComponent) {
  return function EnhancedComponent(props) {
    useEffect(() => {
      console.log(`Component Rendered: ${WrappedComponent.name}`);
    }, []);

    return <WrappedComponent {...props} />;
  };
}

// Simple Component
function UserProfile() {
  return <h2>User Profile Loaded</h2>;
}
```

```
// Applying HOC
const UserProfileWithLogger = withLogger(UserProfile);


function App() {
 return (
 <div>
    <UserProfileWithLogger />
</div>
 );
}

export default App;
```

✅ Har baar UserProfile component render hoga, console me log hoga!

✅ Ye technique debugging aur performance monitoring ke liye useful hai!

When to Use Higher-Order Components?

✓ Reusability: Jab ek hi feature multiple components me lagana ho!

✓ Code Optimization: Code repetition avoid karne ke liye!

✓ Authentication: Protected routes ya user roles handle karne ke liye!

✓ Logging & Performance Monitoring: Component renders track karne ke liye!

React Router

What is React Router?

👉 React Router ek library hai jo tumhe React app me multiple pages banane aur navigate karne ki facility deti hai!

👉 Ye bina page reload kiye pages switch karne deta hai!

Real-Life Example:

Socho tum Netflix dekh rahe ho!

- Agar tum "Home" se "Movies" tab pe click karte ho, toh pura page reload nahi hota!
- Sirf content change hota hai, aur URL bhi update ho jata hai!
- Ye magic hota hai React Router ka!

Install React Router

👉 Sabse pehle React Router ko install karna padega!

npm install react-router-dom

✅ Ab tum React Router ka use kar sakte ho!

Basic Routing Setup

👉 Sabse pehle ek basic routing setup banayenge!

Example: Basic Routing

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

function Home() {
  return <h2>🏠 Welcome to Home Page</h2>;
}

function About() {
  return <h2>📖 About Us</h2>;
}

function Contact() {
  return <h2>📞 Contact Us</h2>;
}

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>
  );
}
export default App;
```

✅ BrowserRouter se routing enable hoti hai!

✅ Routes ke andar multiple Route hote hain jo path aur component define karte hain!

✅ Link ka use karke navigation create kiya, jo a tag se better hai kyunki page reload nahi hota!

Dynamic Routing in React Router

👉 Agar tumhe ek specific user ka profile dikhana ho, toh kaise karoge?

👉 Iske liye dynamic routing ka use hota hai!

Example: Dynamic Routing

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

function UserProfile() {
  const { username } = useParams(); // URL se username
get karna
return
<h2>👤 Welcome, {username}!</h2>;
}
```

```
function App() {
  return (
<Router>
  <nav>
    <Link to="/user/Shaaz">Shaaz's Profile</Link> |
    <Link to="/user/Asad">Asad's Profile</Link>
  </nav>
  <Routes>
    <Route path="/user/:username" element={<UserProfile
/>} />
  </Routes>
</Router>
  );
}

export default App;
```

✅ Agar /user/Shaaz open karoge, toh "Welcome, Shaaz!" dikhega!

✅ Agar /user/Asad open karoge, toh "Welcome, Asad!" dikhega!

Redirecting & Not Found Page (404)

👉 Agar user koi invalid page open kare toh usko 404 Page Not Found dikhana chahiye!

👉 Agar kisi page pe automatically redirect karna ho toh Navigate ka use karna hota hai!

Example: Redirect & 404 Page

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

function Home() {
  return <h2>🏠 Welcome to Home Page</h2>;
}

function NotFound() {
  return <h2>❌ 404 - Page Not Found</h2>;
}

function Dashboard() {
  const isLoggedIn = false; // Change this to true for access
return
isLoggedIn ? <h2>📊 Dashboard</h2> : <Navigate to="/" />;
}
```

```
function App() {
 return (
<Router>
  <Routes>
     <Route path="/" element={<Home />} />
     <Route path="/dashboard" element={<Dashboard />} />
      <Route path="*" element={<NotFound />} />
  </Routes>
</Router>
 );
}

export default App;
```

✅ Agar user /dashboard pe jaaye aur logged in na ho, toh use automatically home page pe redirect kar diya jayega!

✅ Agar user koi invalid URL pe jaaye, toh 404 - Page Not Found dikhega!

Nested Routes in React Router

👉 Agar tum ek page ke andar aur sub-pages banana chahte ho, toh nested routes ka use hota hai!

Example: Nested Routes

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


function Dashboard() {
  return (
   <div>
     <h2>📊 Dashboard</h2>
    <nav>
     <Link to="profile">Profile</Link> |
     <Link to="settings">Settings</Link>
    </nav>
    <Outlet /> {/* Ye nested routes ko dikhayega */}
    </div>
  );
}

function Profile() {
  return <h3>👤 User Profile</h3>;
}

function Settings() {
  return <h3>⚙️ Settings</h3>;
}
```

```
function App() {
 return (
<Router>
   <Routes>
       <Route path="/dashboard" element={<Dashboard />}>
       <Route path="profile" element={<Profile />} />
       <Route path="settings" element={<Settings />} />
      </Route>
   </Routes>
</Router>
 );
}
```

export default App;

✅ Outlet ka use karke nested routes show kiye bina parent component ko modify kiye!

✅ Agar user /dashboard/profile pe jaaye toh User Profile dikhega!

✅ Agar user /dashboard/settings pe jaaye toh Settings dikhega!

Short Summary

✔ React Router bina page reload kiye multiple pages navigate karne me help karta hai!

✔ Basic Routing: Routes aur Route ka use karke multiple pages create kar sakte ho!

✔ Dynamic Routing: useParams() ka use karke URL se dynamic data fetch kar sakte ho!

✔ Redirect & 404: Navigate ka use karke user ko automatically kisi page pe bhej sakte ho!

✔ Nested Routes: Outlet ka use karke ek page ke andar multiple sub-pages show kar sakte ho!

React Redux

What is Redux?

👉 Redux ek state management library hai jo React me data ko centralize aur manage karne me help karti hai!

👉 Redux ek "Global Store" provide karta hai jisse kisi bhi component ko directly data mil sakta hai!

👉 Redux 3 cheezon pe kaam karta hai – Store, Actions aur Reducers!

Real-Life Example:

Socho tum ek shopping mall me ho!

- Mall ka ek centralized PA system (Redux Store) hota hai jo sabko announcement sunata hai!
- Mall ka manager (Action) announce karta hai ki "Sale chal rahi hai!"
- Mall ke har store ka employee (Reducer) us announcement ko sunn ke apna kaam karta hai!

Installing Redux in React

👉 Sabse pehle Redux ko install karna padega!

npm install @reduxjs/toolkit react-redux

✅ Ab tum Redux ka use kar sakte ho!

Redux Ki Basic Architecture

Redux 3 cheezon se milkar banta hai:

Store: Jisme data hota hai

Actions: Jo batate hain ki kya change karna hai

Reducers: Jo store ko update karte hain

Creating a Simple Redux Store in React

👉 Sabse pehle ek simple counter example banayenge jo Redux se manage hoga!

Step 1: Create Redux Store

```
// src/redux/store.js

import { configureStore } from "@reduxjs/toolkit";
import counterReducer from "./counterSlice";


const store = configureStore({
  reducer: {
    counter: counterReducer,
  },
});

export default store;
```

✅ Yahan configureStore() se ek Redux store banaya jo counterReducer use kar raha hai!

Step 2: Create Redux Slice (Reducers + Actions)

```
// src/redux/counterSlice.js

import { createSlice } from "@reduxjs/toolkit";


const counterSlice = createSlice({
  name: "counter",
  initialState: { count: 0 },
  reducers: {
    increment: (state) => {
      state.count += 1;
    },
    decrement: (state) => {
      state.count -= 1;
    },
    reset: (state) => {
      state.count = 0;
    },
  },
});

export const { increment, decrement, reset } =
counterSlice.actions;


export default counterSlice.reducer;
```

✅ createSlice() ka use karke counterSlice banaya jisme actions aur reducers dono hain!

✅ 3 actions hain – increment, decrement, aur reset jo state ko modify karenge!

Step 3: Provide Redux Store to React App

```
// src/main.jsx

import React from "react";

import ReactDOM from "react-dom/client";

import { Provider } from "react-redux";

import store from "./redux/store";

import App from "./App";


ReactDOM.createRoot(document.getElementById("root")).re
nder(
  <Provider store={store}>
    <App />
</Provider>
);
```

✅ Provider ka use karke Redux store ko poore React app me available banaya!

Step 4: Using Redux State and Dispatching Actions

```
// src/App.jsx

import React from "react";

import { useSelector, useDispatch } from "react-redux";

import { increment, decrement, reset } from

"./redux/counterSlice";


function App() {

  const count = useSelector((state) => state.counter.count);

  const dispatch = useDispatch();


  return (

<div>

      <h1>🔥 Redux Counter: {count}</h1>

     <button onClick={() => dispatch(increment())}>➕

Increment

      </button>

     <button onClick={() => dispatch(decrement())}>➖

Decrement

      </button>

      <button

        onClick={() => dispatch(reset())}>🔄 Reset

      </button>

</div>

  );

}

export default App;
```

✅ useSelector() ka use karke Redux store se count liya! 🔥

✅ useDispatch() ka use karke Redux ke actions ko call kiya!

Redux Toolkit with Async Thunk (API Call)

👉 Redux Toolkit ka ek powerful feature hai – createAsyncThunk(), jo API calls handle karta hai!

👉 Agar tumhe kisi API se data fetch karna ho aur Redux store me rakhna ho, toh ye best approach hai!

Example: Fetching API Data with Redux

```
// src/redux/userSlice.js
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";

// API Call using createAsyncThunk
export const fetchUsers = createAsyncThunk("users/fetchUsers", async () => {
  const response = await fetch("https://jsonplaceholder.typicode.com/users");
  return response.json();
});
```

```
const userSlice = createSlice({
 name: "users",
 initialState: { users: [], loading: false },
 reducers: {},
 extraReducers: (builder) => {
 builder.addCase(fetchUsers.pending, (state) => {
 state.loading = true;
 })
 .addCase(fetchUsers.fulfilled, (state, action) => {
 state.loading = false;
 state.users = action.payload;
 })
 .addCase(fetchUsers.rejected, (state) => {
 state.loading = false;
 });
 },
});

export default userSlice.reducer;
```

✅ fetchUsers async action banaya jo API call karega aur Redux store me data save karega!

Using API Data in Component

```
// src/UserList.jsx

import React, { useEffect } from "react";

import { useSelector, useDispatch } from "react-redux";

import { fetchUsers } from "./redux/userSlice";


function UserList() {

  const dispatch = useDispatch();

  const { users, loading } = useSelector((state) => state.users);


  useEffect(() => {

    dispatch(fetchUsers());

  }, [dispatch]);


  return (

    <div>

      <h2>👥 User List</h2>

      {loading ?

       <p>Loading...</p> : users.map((user) =>

         <p key={user.id}>{user.name}</p>

      )}

    </div>

  );

}


export default UserList;
```

✅ Component me fetchUsers() dispatch kiya aur Redux se API data fetch kiya!

Short Summary

✓ Redux ek state management library hai jo data ko centralize karti hai!

✓ Redux ke main parts hote hain – Store, Actions, Reducers!

✓ Redux Toolkit (createSlice()) Redux ko easy aur fast bana deta hai!

✓ useSelector() aur useDispatch() ka use karke Redux state ko access aur modify kar sakte ho!

✓ createAsyncThunk() se Redux me API calls handle kar sakte ho!

Redux Toolkit

What is Redux Toolkit?

👉 Redux Toolkit ek modern way hai Redux manage karne ka!

👉 Ye createSlice(), configureStore(), aur createAsyncThunk() provide karta hai jo Redux ko easy banata hai!

👉 Redux ka pura boilerplate (extra code) remove karne ke liye isko banaya gaya hai!

Real-Life Example:

Socho tum Swiggy 🍕 se online order kar rahe ho!

- Cart me item add karna (State Management)
- Item remove karna (Actions & Reducers)
- Cart ka total price calculate karna (Selector)
- Backend se menu fetch karna (Async API Call)

🔥 Agar tumhari Swiggy app me Redux Toolkit ka use ho, toh ye sab manage karna easy ho jata hai!

Installing Redux Toolkit in React

👉 Sabse pehle Redux Toolkit ko install karna padega!

npm install @reduxjs/toolkit react-redux

✅ Ab Redux Toolkit ka use kar sakte ho!

Redux Toolkit Ki Basic Architecture

Redux Toolkit 3 main cheezon se bana hota hai:

Store: Jisme data hota hai

Slices: Jo state, actions aur reducers ko ek jagah rakhta hai

Dispatch & Selector: Jo component ko Redux store se connect karta hai

Creating a Simple Redux Store (Swiggy Cart Example)

👉 Ab ek real-world example banayenge jo Swiggy ka cart manage karega!

Step 1: Create Redux Store

```
// src/redux/store.js

import { configureStore } from "@reduxjs/toolkit";
import cartReducer from "./cartSlice";


const store = configureStore({
  reducer: {
    cart: cartReducer,
  },
});

export default store;
```

Step 2: Create Redux Slice for Cart

```
// src/redux/cartSlice.js
import { createSlice } from "@reduxjs/toolkit";

const cartSlice = createSlice({
  name: "cart",
  initialState: { items: [], totalAmount: 0 },
  reducers: {
    addItem: (state, action) => {
      state.items.push(action.payload);
      state.totalAmount += action.payload.price;
    },
    removeItem: (state, action) => {
      const index = state.items.findIndex((item) => item.id ===
action.payload);
      if (index !== -1) {
        state.totalAmount -= state.items[index].price;
        state.items.splice(index, 1);
      }
    },
    clearCart: (state) => {
      state.items = [];
      state.totalAmount = 0;
    },
  },
});

export const { addItem, removeItem, clearCart } = cartSlice.actions;
export default cartSlice.reducer;
```

✅ Yahan createSlice() ka use karke Redux slice banaya jo cart ke state aur actions handle karega!

✅ 3 actions hain – addItem, removeItem, aur clearCart jo cart manage karenge!

Step 3: Provide Redux Store to React App

```
// src/main.jsx

import React from "react";

import ReactDOM from "react-dom/client";

import { Provider } from "react-redux";

import store from "./redux/store";

import App from "./App";


ReactDOM.createRoot(document.getElementById("root")).re
nder(
  <Provider store={store}>
    <App />
  </Provider>
);
```

✅ Provider ka use karke Redux store ko poore React app me available banaya!

Step 4: Using Redux State and Dispatching Actions

```
// src/App.jsx
import React from "react";
import { useSelector, useDispatch } from "react-redux";
import { addItem, removeItem, clearCart } from
"./redux/cartSlice";

function App() {
  const items = useSelector((state) => state.cart.items);
  const totalAmount = useSelector((state) =>
state.cart.totalAmount);
  const dispatch = useDispatch();

  const sampleItem = { id: 1, name: "Pizza 🍕", price: 300 };
```

```
return (
  <div>
    <h1>🛒 Swiggy Cart</h1>
     <button
        onClick={() => dispatch(addItem(sampleItem))}>➕ Add Pizza
     </button>
     <button
        onClick={() => dispatch(removeItem(sampleItem.id))}>➖ Remove Pizza
    </button>
    <button
      onClick={() =>
        dispatch(clearCart())}>🗑️ Clear Cart
    </button>
    <h2>Cart Items:</h2>
<ul>
   {items.map((item, index) => (
   <li key={index}>{item.name} - ₹{item.price}</li>
    ))}
</ul>
    <h3>Total Amount: ₹{totalAmount}</h3>
</div>
 );
}

export default App;
```

✅ useSelector() ka use karke Redux store se cart items aur total amount fetch kiya!

✅ useDispatch() ka use karke Redux ke actions ko call kiya!

Redux Toolkit with API Calls (createAsyncThunk)

👉 Agar tumhe kisi API se menu fetch karna ho, toh createAsyncThunk() ka use kar sakte ho!

Example: Fetching Menu from API

```
// src/redux/menuSlice.js
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";
// API Call using createAsyncThunk
export const fetchMenu = createAsyncThunk("menu/fetchMenu", async () => {
  const response = await fetch("https://fakestoreapi.com/products");
  return response.json();
});

const menuSlice = createSlice({
  name: "menu",
  initialState: { items: [], loading: false },
  reducers: {},
  extraReducers: (builder) => {
    builder
      .addCase(fetchMenu.pending, (state) => {
        state.loading = true;
      })
      .addCase(fetchMenu.fulfilled, (state, action) => {
        state.loading = false;
        state.items = action.payload;
      })
      .addCase(fetchMenu.rejected, (state) => {
        state.loading = false;
      });
  },
});
export default menuSlice.reducer;
```

✅ API call fetchMenu() se ho rahi hai aur Redux store me data store ho raha hai!

Using API Data in Component

```
// src/MenuList.jsx

import React, { useEffect } from "react";

import { useSelector, useDispatch } from "react-redux";

import { fetchMenu } from "./redux/menuSlice";


function MenuList() {

  const dispatch = useDispatch();

  const { items, loading } = useSelector((state) => state.menu);


  useEffect(() => {

    dispatch(fetchMenu());

  }, [dispatch]);


  return (

    <div>

      <h2>Menu List</h2>

      {

loading ?

 <p>Loading...</p> : items.map((item) =>

<p key={item.id}>{item.title}</p>)

}

    </div>

  );

}


export default MenuList;
```

✅ API se fetched menu items Redux store me store ho rahe hain!

Lazy Loading

What is Lazy Loading?

👉 Lazy Loading ek technique hai jisme component ko tabhi load kiya jata hai jab user ko zaroorat ho!

👉 Ye React ke React.lazy() aur Suspense se hota hai!

👉 Isse app fast load hoti hai aur unnecessary components load nahi hote!

Real-Life Example:

Socho tum Amazon shopping app use kar rahe ho!

- Agar tum sirf "Electronics" section dekh rahe ho, toh "Fashion" section ke images aur products load karke kya fayda?
- Lazy Loading yahi karta hai – sirf wahi cheez load karta hai jo user dekhe!

How to Use Lazy Loading in React?

👉 React me lazy loading karne ke liye React.lazy() aur Suspense ka use hota hai!

Example: Basic Lazy Loading

```
import React, { lazy, Suspense } from "react";


// Lazy Loaded Component
const About = lazy(() => import("./About"));


function App() {
 return (
<div>
<h1>🏠 Home Page</h1>
<Suspense fallback={<h2>Loading...</h2>}>
      <About />
      </Suspense>
</div>
 );
}

export default App;
```

✅ Yahan About component ko lazy load kiya, jo tabhi load hoga jab zaroorat padegi!

✅ Suspense ka fallback tab dikhega jab tak About load nahi hota!

Lazy Loading with React Router

👉 Agar tumhe different pages lazy load karne hain, toh React.lazy() ko React Router ke saath use kar sakte ho!

Example: Lazy Loading Pages in React Router

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

const Home = lazy(() => import("./Home"));
const About = lazy(() => import("./About"));
const Contact = lazy(() => import("./Contact"));

function App() {
  return (
    <Router>
      <nav>
        <Link to="/">Home</Link> |
        <Link to="/about">About</Link> |
        <Link to="/contact">Contact</Link>
      </nav>

      <Suspense fallback={
        <h2>Loading Page...</h2>
      }>
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/about" element={<About />} />
          <Route path="/contact" element={<Contact />} />
        </Routes>
      </Suspense>
    </Router>
  );
}

export default App;
```

✅ Ab "Home", "About" aur "Contact" pages tabhi load honge jab user unpe jaayega! 🔄

✅ fallback tab dikhayega jab tak page load nahi ho raha!

Lazy Loading with Dynamic Imports (Code Splitting)

👉 Agar tumhe kisi ek specific feature ya component ko lazy load karna ho, toh import() ka use kar sakte ho!

Example: Dynamic Component Loading

```
import React, { useState, lazy, Suspense } from "react";
const HeavyComponent = lazy(() => import("./HeavyComponent"));

function App() {
  const [showComponent, setShowComponent] = useState(false);

  return (
    <div>
      <h1>🔥 Dynamic Lazy Loading</h1>
    <button
     onClick={() =>
setShowComponent(true)}>Load Heavy  Component
    </button>

     {showComponent && (
      <Suspense fallback={<h2>Loading Component...</h2>}>
       <HeavyComponent />
      </Suspense>
     )}
    </div>
  );
}
export default App;
```

✅ "HeavyComponent" tabhi load hoga jab user "Load Heavy Component" button dabayega!

✅ Isse unnecessary components load nahi honge aur app fast chalegi!

Optimizing Lazy Loading with React.memo()

👉 Agar tumhara component baar-baar re-render ho raha hai bina kisi change ke, toh React.memo() ka use karo!

Example: Preventing Unnecessary Renders

```
import React, { memo } from "react";
const OptimizedComponent = memo(function OptimizedComponent({ text }) {
 console.log("Rendered: ", text);
 return <h2>{text}</h2>;
});
export default OptimizedComponent;
```

✅ React.memo() component ko unnecessary re-renders se bachata hai! 🔥

Lazy Loading Images in React

👉 Agar tumhari website me bohot saari images hain, toh react-lazy-load-image-component ka use karo!

📌 **Install Lazy Load Image Package**

```
npm install react-lazy-load-image-component
```

Example: Lazy Loading Images

```
import React from "react";
import { LazyLoadImage } from "react-lazy-load-image-component";
import "react-lazy-load-image-component/src/effects/blur.css";

function ImageComponent() {
 return (
  <LazyLoadImage
src="https://source.unsplash.com/random/800x600"alt="Lazy Loaded"effect="blur"width="100%"height="auto"
  />
 );
}

export default ImageComponent;
```

✅ Images tabhi load hongi jab wo screen me dikhengi, jisse website fast chalegi!

Short Summary

✓ Lazy Loading components sirf tab load hote hain jab zaroorat ho!

✓ React me lazy loading React.lazy() aur Suspense ka use karke hoti hai!

✓ React Router ke saath lazy loading pages ko optimize kar sakti hai!

✓ Dynamic Import (import()) se specific components load kar sakte ho!

✓ Images lazy load karne ke liye react-lazy-load-image-component ka use hota hai!