#. React hook

**\*\*. State and lifecycle hook start**

#. Usestate hook

#. Useeffect hook

#. useLayoutEffect hook

**\*\*. State and lifecycle hook end**

#. Props drilling

\*\*. Context and state management hook start

#. Usecontext

#. useReducer hook

#. Usememo hook important

#. Usecallback hook

#. useTransition hook

#. useRef hook

#. useId hook very easy hook

#. Custom hook using useFetch fetching data

<https://www.youtube.com/watch?v=KUyxBsop5Tc&ab_channel=WebDevMastery>

#. React hook

React hooks are functions that let functional components access the state and lifecycle features which were available only in class components

Ex:usestate, useeffect, usememo

**Hooks are predefined functions** like in c# you have string.toUpper() string.toLower() string.replace()

Arr.sort()

In react predefined functions are **hooks**

**\*\*. State and lifecycle hook start**

#. Usestate hook

In my ui I want button and counter

2 buttons increment and decrement buttons on clicking them counter changes

In react if we create normal variable inside function example

let counter = 0; this is immutable you can’t change with any other method also

function Counter(){

    let exCounter = 0;

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

 const increment = () => {

    exCounter++;

console.log(exCounter);

    setCount((c)=> c + 1);

// setCount(count+1); this also works

 }

 const decrement = () => {

    exCounter--;

    setCount((c)=> c - 1);

 }

 return (

    <div>

        <h1>count is : {count}</h1>

        <h1>exCounter is : {exCounter}</h1>

        <button onClick={increment}>increment</button>

        <button onClick={decrement}>decrement</button>

    </div>

 );

}

In above code when you run when you click button exCounter will not change as it’s immutable

It’ll change in console means while printing in browser you see its value changing but not in ui with h1 tag

To see change just after button in click UI use **useState** hook

Here useState() is predefined function in react

 const [count, setCount] = useState(0) here we have 2 things variable counter and function setCount

**count=>variable**

**setCount=> function**

**useState=> sets initial value**

setCount is function which will change value of count variable

useState(0) here 0 indicates initial state of counter variable means 0 will go to counter that is counter=0

useState(10) then count=10 count becomes 10

we can set string, decimal, array with useState

const [name, setName] = useState('ram');

 const [price, setPrice] = useState(0.1);

 const [nums, setNums] = useState([1,2,3]);

#. Useeffect hook

Very important

This is mainly used to handle side effects

Use when you need to perform side effects such as API calls, event listeners, or subscriptions.

The function that runs after render (the side effect)

Simple: **when you want to fetch data that time use useeffect**

function FetchData(){

 useEffect(()=>{

    console.log("useeffect runs")

 },[])

 return (

    <div>

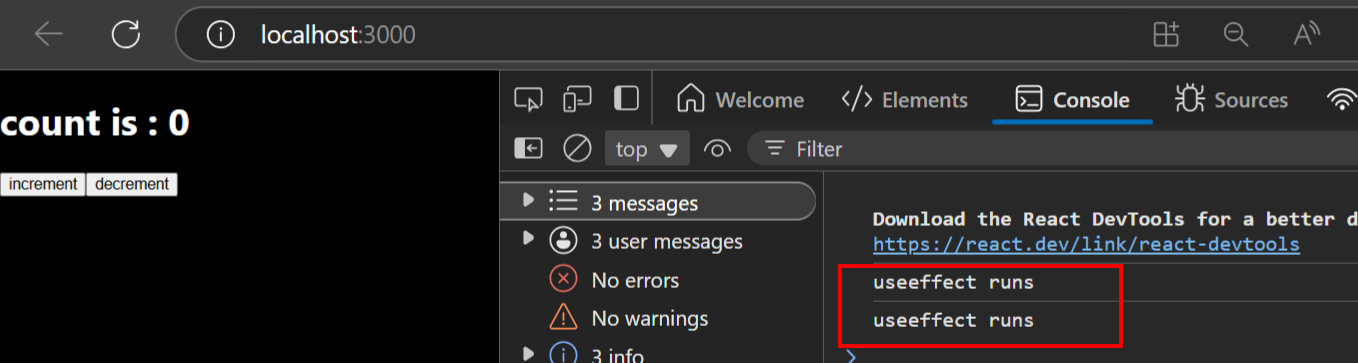
    </div>

 );

}

When component or method called that time useeffect will be called twice you can see this in console

This is due to react **life cycle mount unmount mount method**s



Even though function called once

console.log("useeffect runs")

above printed twice

In development only (not in production!), React intentionally calls some functions (including useEffect) twice:

Mount → Unmount → Mount again

To help catch side effect bugs early

✅ That’s why console.log("useeffect runs") prints twice — even though it logically runs only once.

\*. Now we want to show title as counter like 0 1 2 so for this we use useEffect

Title means browsers tab name

function FetchData(){

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

 useEffect(()=>{

    console.log("useeffect runs");

    document.title=count;

 },[count])

 const increment = () => {

    setCount(count+1);

 }

 const decrement = () => {

    setCount(count-1);

 }

 return (

    <div>

        <h1>count is : {count}</h1>

        <button onClick={increment}>increment</button>

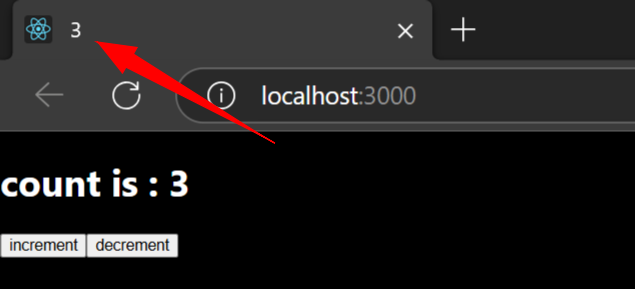
        <button onClick={decrement}>decrement</button>

    </div>

 );

}

above



As soon as you click button value of count changes then title will also shows count value

How this happens: useFetch we have passed dependency array in that we passed [count] **as soon as count changes** with button clicks like increment or decrement button useEffect will be called and it updates browser tab name or title to new count value

document.title=count; this code upates title

useEffect(()=>{

    console.log("useeffect runs");

    document.title=count;

 },[count])

[count] array is dependency array

If dependency array is empty useeffect runs only 2 times

\*\*. Fetch data from api using useeffect

We use <https://jsonplaceholder.typicode.com/> this gives free fake rest api for testing

fetch('https://jsonplaceholder.typicode.com/todos/1')

.then(response => response.json())

.then(json => console.log(json))

Above code is In that website

https://jsonplaceholder.typicode.com/todos/1

Now copy above code paste in browser url it shows data

In that jsonplaceholder also there is instructions how to use other

|  |  |
| --- | --- |
| [/posts](https://jsonplaceholder.typicode.com/posts) | 100 posts |
| [/comments](https://jsonplaceholder.typicode.com/comments) | 500 comments |

above

<https://jsonplaceholder.typicode.com/posts> will show us 100 posts

means

first write useffect

useEffect(()=>{

 },[])

Like above next write a method inside it which is async method

const fetchDataFromApi = async() => {

    }

After async() two brackets() are callback

Next we use **fetch** which is inbuilt method of javascript to fetch from api

useEffect(()=>{

    const fetchDataFromApi = async() => {

        const api = await fetch("https://jsonplaceholder.typicode.com/users")

        const result = await api.json();

        console.log(result);

    };

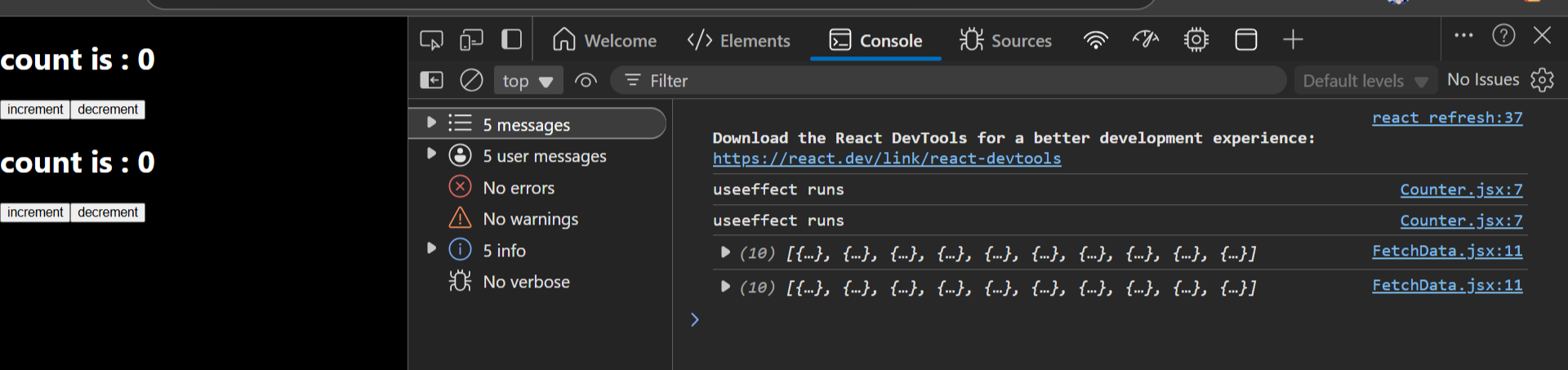
**fetchDataFromApi**()

 },[])

Above we can see await means async method will wait asynchrounously means it’ll not freeze the ui when it tries to fetch data from api control will go to cpu, next once data comes it goes to next line calls result

1. Starts the request.
2. **Pauses** the function at that point.
3. **Gives control back** to the event loop.
4. When the response arrives, the function **resumes** from where it paused.

✅ **Meanwhile, the UI stays responsive** — you can still scroll, click buttons, type input, etc.



We can see 2 times data came here

#. useLayoutEffect hook

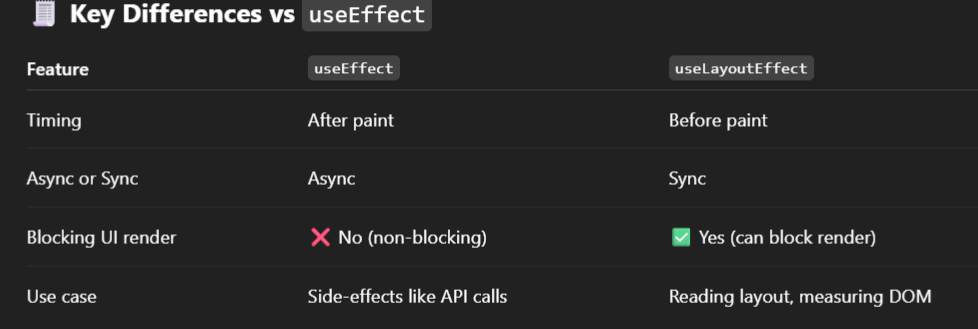
useLayoutEffect is a React Hook that is similar to useEffect, but it runs synchronously after all DOM mutations and before the browser paints the screen

Use when you need to measure DOM size or manipulate the layout before the browser paints.

✅ **Benefits:**

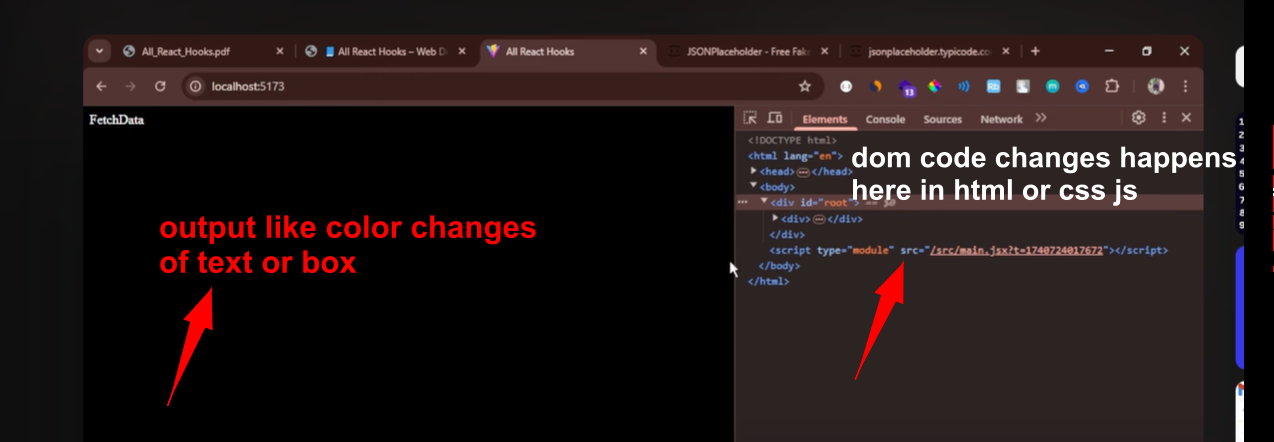
**● Runs before browser paint, ensuring smoother UI updates**

when we have to do modification in browser



Example: if you’re updating style using css based on the state you want to make sure that the style changes are applied before the browser paints we might use uselayouteffect

**First we want to update DOM later we want to update style that is paint our browser page like coloring a text**



**Above** image see: with uselayouteffect first output changes happens in left side in our screen next right side in dom actual code changes

He told it runs before uselayouteffect

Changes appear in browser first then dom means when you click or see ‘browser developer tools’ you’ll see conole, elements that is where inside elements our dom means html css js codes are present

We use this code: document.body.style.backgroundcolor to change background color

In uselayouteffect code also we have dependency array where we pass state variable on change of state variable value uselayouteffect rerenders or called again

function ChangeBackColor(){

 const [color, setColor] = useState('lightblue');

 useLayoutEffect(()=>{

    document.body.style.backgroundColor = color;

 },[color]);

 return (

    <div style={{textAlign:"center",padding:"20px"}}>

        <h2>Current background color is {color}</h2>

        <button onClick={() =>setColor("lightcoral")}>Red</button>

    </div>

 );

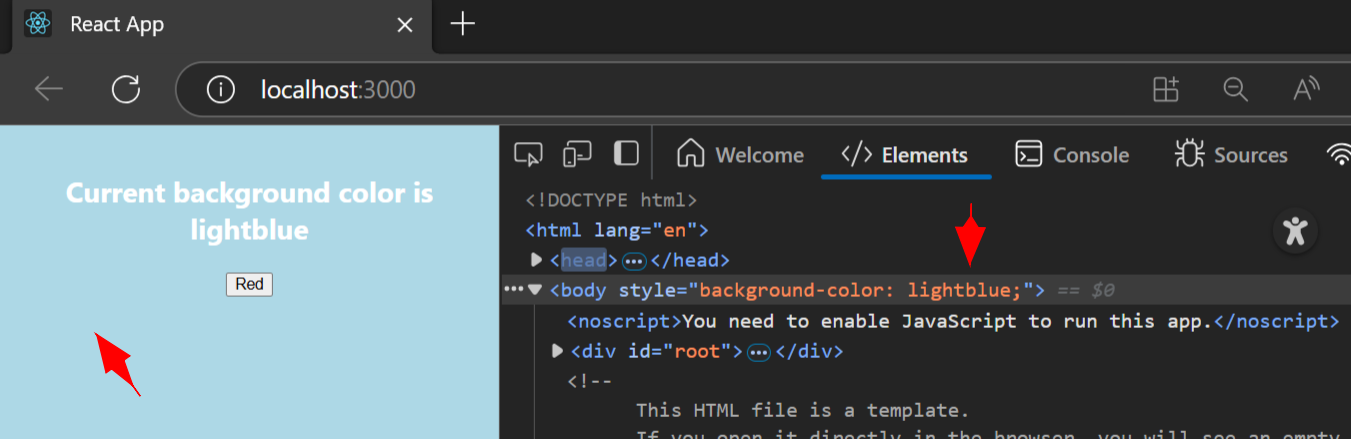
}

Because of useState(lightblue) initial when browser loads color is lightblue

Whenever [color] changes in dependency array uselayouteffect will be callled

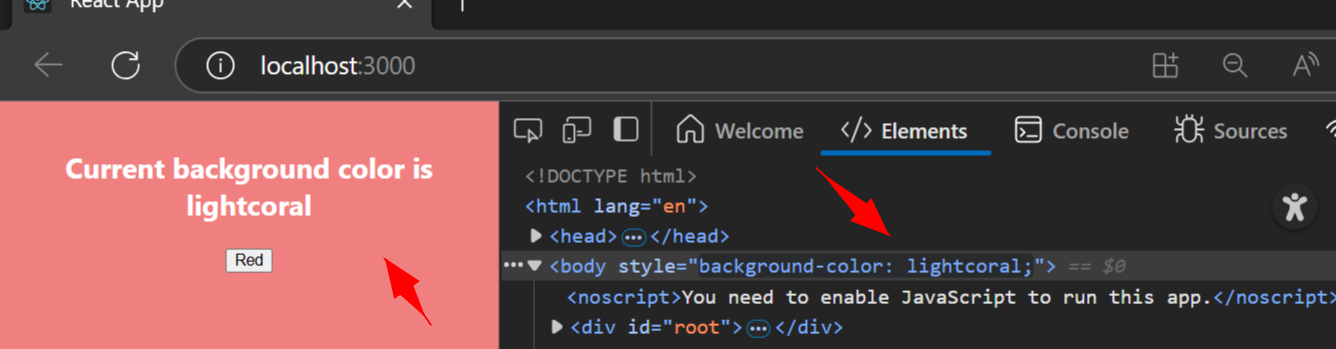
I got to know difference

Below image is before clicking button red we see left side browser back color light blue also in dom’s element color lightblue



Next when I clicked on red button I saw first color changed in browser left side after that in dom’s elements tab’s

Background color change to lightcoral



1:00:02

**\*\*. State and lifecycle hook end**

#. Props drilling:

Example: take government: Indian government -> state government -> district -> village -> people

So if Indian government sends 100rs to people while reaching to common citizen it becomes 1rs earlier

It was pass throught state,district,village middle politicians they’ll deduct take those money

Above example in earlier government

Now modi will tell if he sends 100rs he’ll send directly to common citizen this is **by useContext hook**

Nobody will cut their money or charge when modi sends money when using useContext

So **above state government -> district -> village** all these are **middleware**

useContext what it does without passing money to middleware it directly sends money from Indian government from modi to common citizen

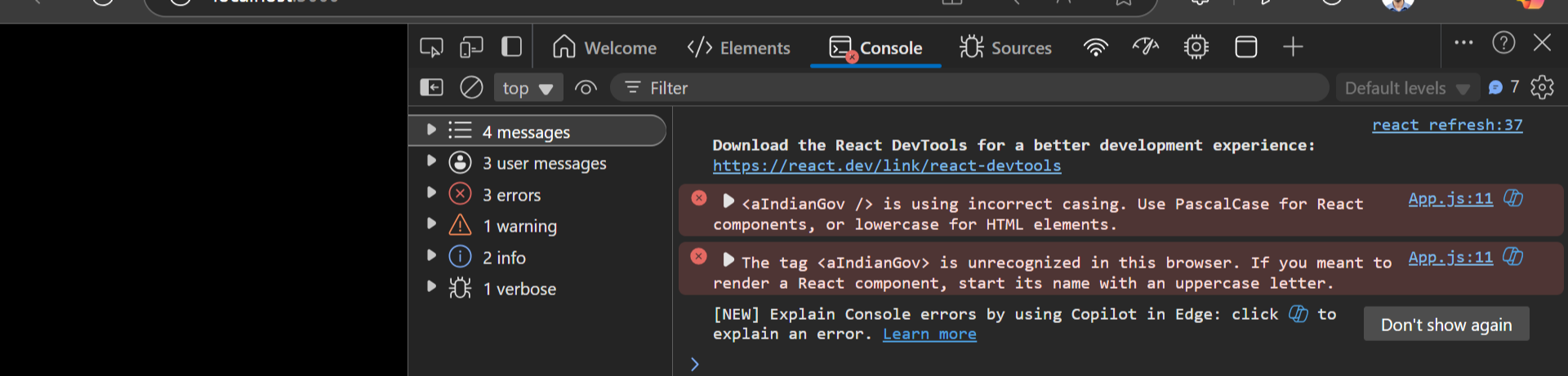
useContext: centralized system bank where Indian citizen’s account is there central government directly credits money to that account

Props drilling is a pattern in React where data (props) is **passed from** a parent component down through **multiple layers of child components**, even if some intermediate components **don’t need it** — just to get it to a deeply **nested component**.

\*\*. Context and state management hook start

#. Usecontext

Above naming **will not work above is camelCase react only allows pascalcase PascalCase**



**I got above error later** I changed aIndianGov to AIndianGov

**Prefix I named added a,b,c,d,e,f so they look in order india,state,district,block,panchayat,people**

Video youtube example:

Create: Indiangov.jsx StateGov.jsx District.jsx Block.jsx Panchayat.jsx People.jsx we create all these components

Next what we do: it’s like stategov reports to indiangovt

Inside Indiangovt we import stategovt and use stategovt why because inside country india we have state

Inside stategovt we import District and use District why because inside stategovt we have District

Inside District we import Block and use Block why because inside District we have Block

Inside Block we import and Panchayat use Panchayat why because inside Block we have Panchayat

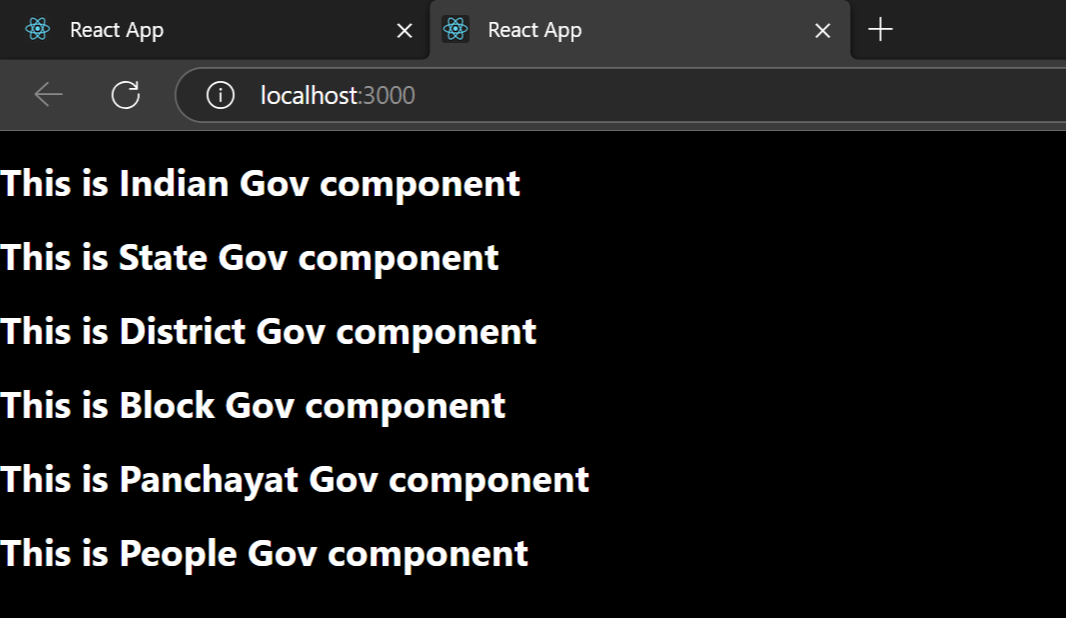
Inside Panchayat we import People and use People why because inside Panchayat we have People

A screen shot of a computer program

AI-generated content may be incorrect.

LIKE ABOVE for all others we do we import lower file and use inside div of return

aIndianGov.jsx is our main component which we import in our app.jsx file



Output like above

Now I’ve 100rs money in IndiaGovt.jsx component that I want to print or display in People.jsx component

function IndianGov(){

    const money = 100;

 return (

    <div>

        <h1>This is Indian Gov component</h1>

        <StateGov />

    </div>

 );

}

Above is in indiaGov.jsx component where we have money

Props drilling means: from indianGov.jsx pass money parameter to StateGov.js

from StateGov.jsx pass money parameter to District.jsx

from District.jsx pass money parameter to Block.jsx

from Block.jsx pass money parameter to Panchayat.jsx

from Panchayat.jsx pass money parameter to People.jsx

every component from StateGovt receive parameter money

below is full code of all component

**function IndianGov(){**

**const money = 100;**

**return (**

**<div>**

**<h1>This is Indian Gov component</h1>**

**<StateGov money={money} />**

**</div>**

**);**

**}**

function StateGov(**{money}**){

 return (

    <div>

        <h1>This is State Gov  component</h1>

        <District money={money}/>

    </div>

 );

}

function District(**{money}**){

 return (

    <div>

        <h1>This is District Gov component</h1>

        <Block money={money} />

    </div>

 );

}

function Block(**{money}**){

 return (

    <div>

        <h1>This is Block Gov component</h1>

        <Panchayat money={money} />

    </div>

 );

}

function Panchayat(**{money}**){

 return (

    <div>

        <h1>This is Panchayat Gov component</h1>

        <People money={money} />

    </div>

 );

}

**function People({money}){**

**return (**

**<div>**

**<h1>This is People Gov component and money is {money}</h1>**

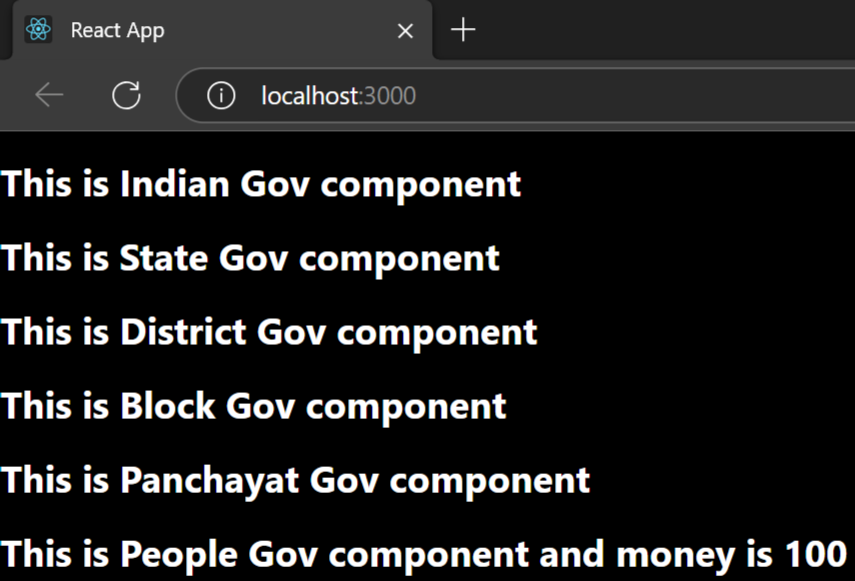
**</div>**

**);**

**}**

We see at final People.jsx component we print money

Output like below



Above is called props drilling where we pass props to each component

Props drilling problem is if I want to pass another input parameter then I have to pass through all different components

Solve this by using useContext hook

Create centralized state in that state keep all data

He told 4 step we’ve to follow for useContext

1.Create Context

2. Create Provider

3. Wrap the App with Provider

4. Use The Context in Child Component

\*1. We have file inside UseContext folder that is **Context.jsx here** we write all code

First import **createContext from react** here

const Context = createContext()

export default Context;

\*2. Next step we create another file MoneyState.jsx here we put all data which we want to send to people

It can be shoppingCart bookCart.jsx etc HotelState.jsx can contain data of hotel

Inside this file we create centralized state using usestate

Once after creating separate state file next import Context into MoneyState.jsx file and

**function MoneyState(){**

**return (**

**<** **MyContext>**

**</** **MyContext>**

**);**

**}**

Use like above is MoneyState.jsx file and Context is from Context.jsx file which we imported

Next use provider

Moneystate becomes our provider component make Context.Provider

Next for MoneyState pass argument props

Next put {props.children} so that it can be used by children

Next for <Context.Provider> pass value and value will be in {{}} double curly bracket

In that bracket we pass money value, state anything we can pass

**function MoneyState(props){**

**const money = 1000;**

**const [counter, setCounter] = useState(0);**

**return (**

**<** **MyContext.Provider value={{money,counter,setCounter}}>**

**{props.children}**

**</** **MyContext.Provider>**

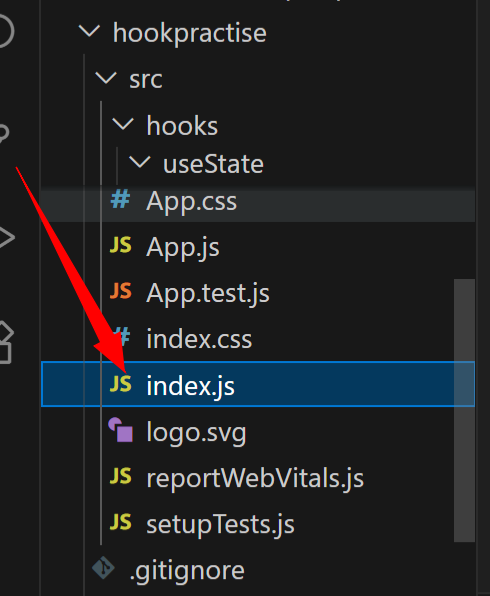
**);**

**}**

Like above

\*3. Wrap index.jsx or **App.jsx** with State file that is **MoneyState.jsx**

**Why to** wrap: so that **is available in all places** means in all child or other components



In index.js file we have

**<React.StrictMode>**

**<App />**

**</React.StrictMode>**

Above

Remove it and replace with MoneyState and also import MoneyState

Now indx.js looks like

**const root = ReactDOM.createRoot(document.getElementById('root'));**

**root.render(**

**<MoneyState>**

**<App />**

**</MoneyState>**

**);**

Above means our app is inside MoneyState so our MoneyState can be accessed by all components

Above we did it wrap up of MoneyState

\*4. Next step use MoneyState in People.jsx component

So we can use it directly because we wrapped MoneyState into app.js inside index.jsx

Now in People.jsx first import Context.jsx

Now inside People.jsx we have to use **useContext** hook why because in my index.js file **I can have multiple contexts wrappers wrapped App**

**MoneyState** is not only our state if we have multiple different states like cartState we wrap like below

**Lets** say we have multiple contexts they look like below example

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<FlipkartCart>

<DMartCart>

<User>

< **MoneyState**>

<App />

</ **MoneyState** >

</User>

</DMartCart>

</FlipkartCart>

);

we

FOODDELIVERY can be there

Now we put MyContext into useContext in our people.jsx

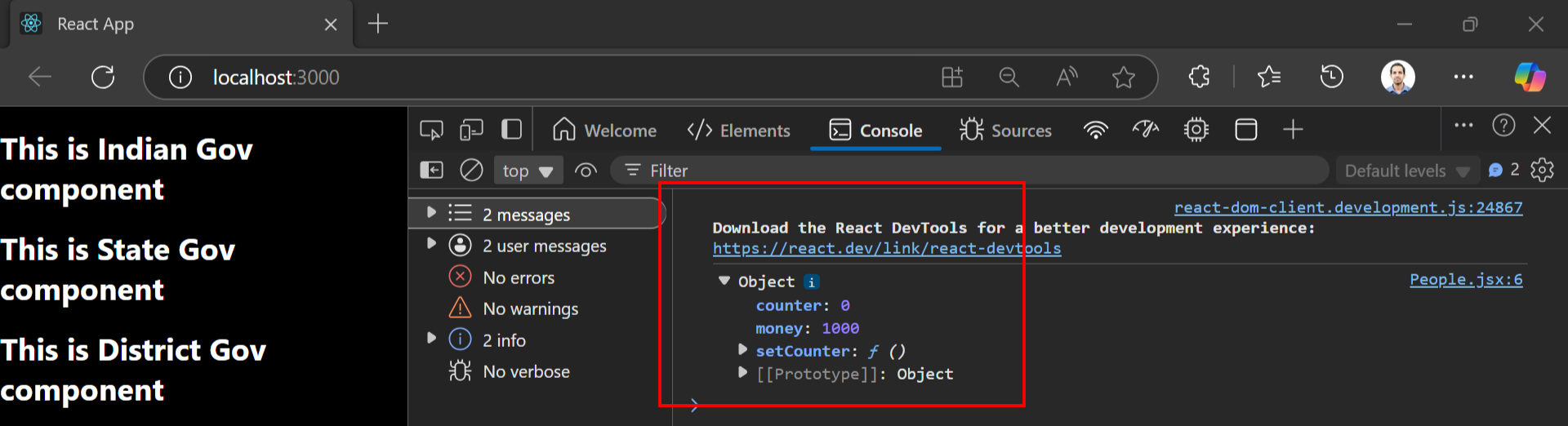
I made a mistake while printing mycontext because of that nothing printed in browser

**console.log(useContext(“MyContext”));**

Before above

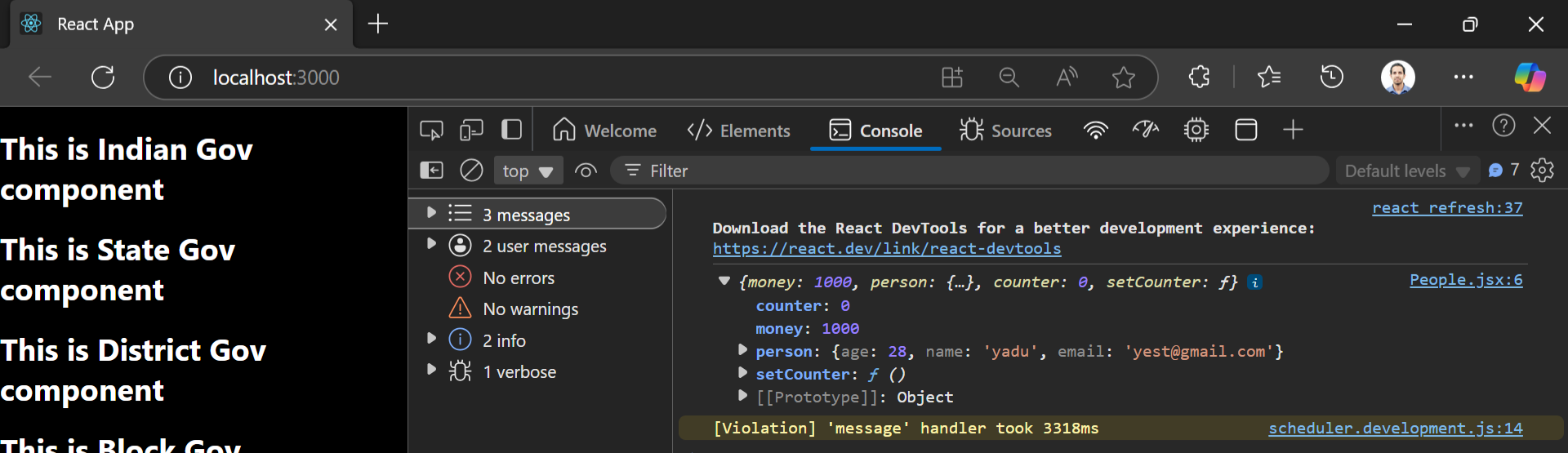
**console.log(useContext(MyContext));**

After changes working code



Output like above

Now from moneystate component if we pass big object dictionary that also will be printed in browser



We don’t have to do any prop drilling

If I want to show only money do

**const People = () =>{**

**const firstMyContext = useContext(MyContext);**

**return (**

**<div>**

**<h1>This is People Gov component and money is {firstMyContext.money} </h1>**

**</div>**

**);**

**}**

Above

Till now we printed data or sent data from parent component from moneystate using mycontext into child component people.jsx

Now we want send data from child component to parent component

Send data from child people.jsx to moneystate.jsx parent component

We use setCounter

**const People = () =>{**

**const firstMyContext =  useContext(MyContext);**

**return (**

**<div>**

**<h1>This is People Gov component and money is {firstMyContext.money} </h1>**

**<h1>counter is {firstMyContext.counter} </h1>**

**<button onClick={()=>firstMyContext.setCounter(firstMyContext.counter + 1)}>Increase counter</button>**

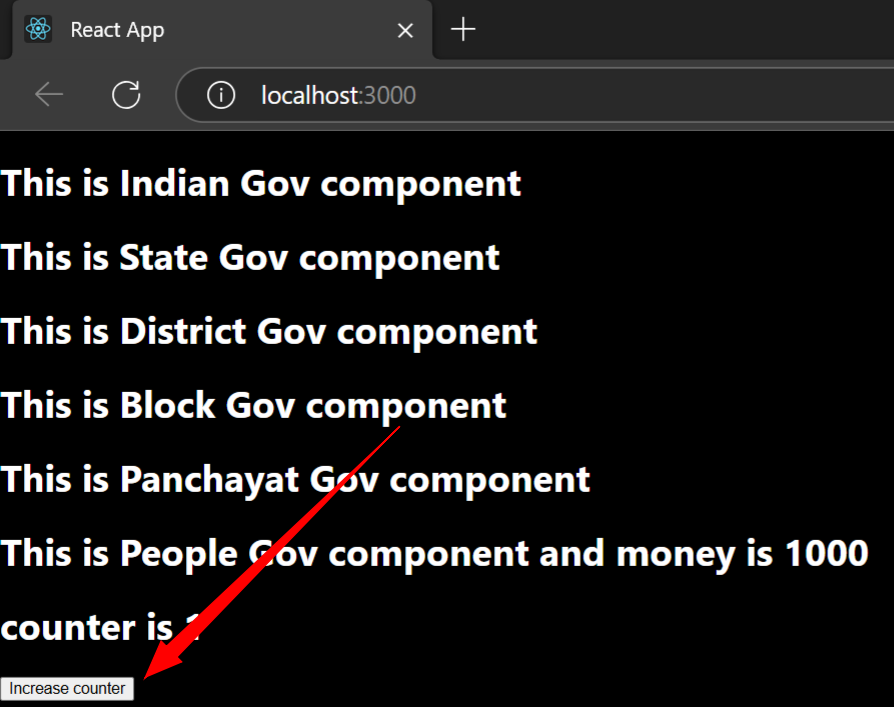
**</div>**

**);**

**}**

Above button click we update counter which will be displayed in people.jsx file itself

**firstMyContext.setCounter(firstMyContext.counter + 1)}**



#. useReducer hook

Not that much useful he told

useReducer is a React hook that's used for managing **complex state logic** in components, especially when:

* You have multiple state variables that depend on each other.
* The next state depends on the previous state.
* You want a centralized way to manage state (like Redux, but local).

It works like a mini version of Redux inside a component.

Used in complex states

Folder structure

📂 **Folder Structure**

/useReducerExample

├── src

│ ├── components

│ │ ├── Counter.jsx

│ │ ├── counterReducer.js

│ ├── App.jsx

│ ├── main.jsx

He told either we can do all work in a single file or multiple file

We will have to use switch case because we’re using complex state

Usereducer is similar to usestate

If we want to create reducer for counter then coutnerReducer, for ecart ecartReducer

For booking we have bookingReducer

const counterReducer = (state,action){

}

Above means state we pass as parameter and also action also as parameter action tells what action to perform

Inside that we use switc case

For switch we pass action.type

Now in switch case’s case we’ll have many cases increment decrement like that

In increment we increase count of count variable if case matches INCREMENT

**const counterReducer = (state,action) => {**

**switch(action.type){**

**case 'INCREMENT' :**

**return {count : state.count + 1};**

**}**

**}**

As said above method in switch we check action type if it matches case increment we increase count

In switch case default we keep initial state

Means we don’t change anything

**const counterReducer = (state,action) => {**

**switch(action.type){**

**case 'INCREMENT' :**

**return {count : state.count + 1};**

**case 'DECREMENT' :**

**return {count : state.count - 1};**

**default : return state;**

**}**

**}**

Till above we created counterreducer function next what to do?

Similar to useState here also we do in use state

We did like this const [count, setCount] = useState(0) like this we do for useReducer

**const [state, dispatch] = useReducer(counterReducer,{count:0})**

our snippet looks like above we pass state and dispatch and inside usereducer we call our method counterReducer and after comma we pass our state which is variable count and 0 is initial value

if we give state name as count then inside counterReducer function’s switch case while updating also state name should be count only it should match or else it gives error

next we display that count

**const counterReducer = (state,action) => {**

**switch(action.type){**

**case 'INCREMENT' :**

**return {count : state.count + 1};**

**case 'DECREMENT' :**

**return {count : state.count - 1};**

**default : return state;**

**}**

**}**

**function Reducer(){**

**const [state, dispatch] = useReducer(counterReducer,{count:0})**

**return (**

**<div>**

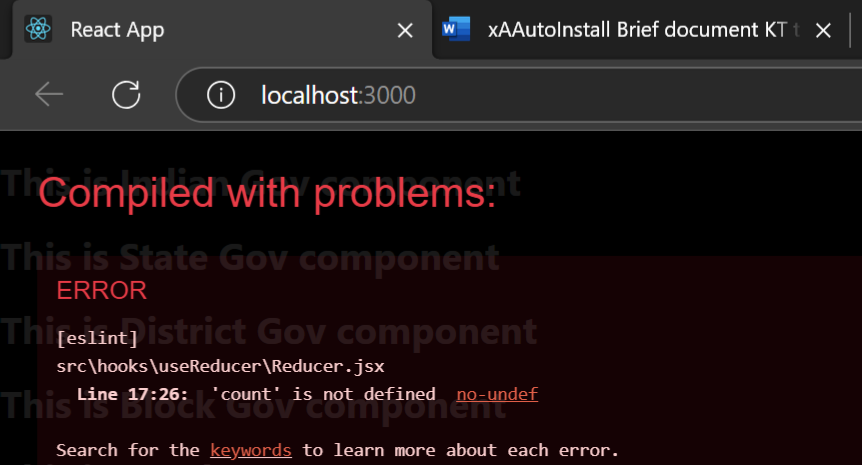
**<h1>count value {count}</h1>**

**</div>**

**);**

**}**

But with above code in reducer.jsx we get error



Why we used only {count} in h1 tag it should be state.count

**function Reducer(){**

**const [state, dispatch] = useReducer(counterReducer,{count:0})**

**return (**

**<div>**

**<h1>count value {state.count}</h1>**

**</div>**

**);**

**}**

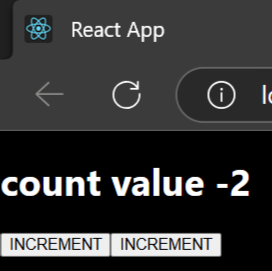
S

Next in button onclick using dispatch method we defined using that we call both types like increment or decrement types present with switch case

**<button onClick={()=> dispatch({type:'INCREMENT'})}>INCREMENT</button>**

We add button like above where same type present in our counterReducer method’s switch case

Same we do for DECREMENT also we create button for it



Looks likeabove

#. Usememo hook important

Use when performing expensive calculations to prevent unnecessary recomputation.

✅ Benefits:

● Caches the result, preventing unnecessary recalculations.

When we want to save

**useCallBack:** memorize the function

**useMemo :** memorize the function value and returns value result

we’ll see with example before and after using useMemo

create new folder useMemo inside it we create new file cart.jsx

first we **see problem without using useMemo**

we do some calculation like assume we have flipkart cart we have added 10laptops now we find totalPrice for all 10 laptops

next we also find discount calculation now discount calculation happens using button click: **every time button click we increase discount price**

now problem is every time discount changes it re renders entire Cart component

we create 2 usestates cart and discount cart has object

discount is integer

now he**’s using array.reduce** to find totalPrice you can loop thourhg items of array add it to variable

but **array.reduce** make our code clean

**const totalPrice = cart.reduce((total,item)=>{**

**console.log("Calculation total price...");**

**return total + item.price**

**}, 0);**

reduce will loop though each item of array and we keep adding it to total variable **less lines of code**

above we see in reduce at the end we put **0** it’s for initial value we have to give

after calculating totalprice

next we show cart items in page using p tag using **car.map**

cart.map inside we do callback function or arrow function

**<button onClick={()=>setDiscount(discount + 10)}>Increase discount</button>**

For discount we do above button click

Now main problem is with discount: every time discount changes it re renders entire cart **component** with this rerendering again tot**alcalculation** which is time consuming calculation will be calculated again this takes more times

**Problem Explanation:**

● Every time **discount** changes, the component **re-renders**.

● The totalPrice **recalculates unnecessarily**, even though cart hasn't changed.

● This **wastes performance**.

Time: 1:53:26

Code:

**function Cart() {**

**const [cart, setCart] = useState([**

**{ id: 1, name: "Laptop", price: 50000 },**

**{ id: 2, name: "Phone", price: 30000 },**

**{ id: 3, name: "Headphones", price: 2000 },**

**]);**

**const [discount, setDiscount] = useState(0);**

**const totalPrice = cart.reduce((total, item) => {**

**console.log("Calculation total price...");**

**return total + item.price**

**}, 0);**

**return (**

**<div>**

**<h2>Shopping cart</h2>**

**{cart.map((item) => (**

**<p style={{ marginLeft: '5px' }} key={item.id}>{item.name} : price is {item.price}</p>**

**))}**

**<h3>Total price: {totalPrice}</h3>**

**<button onClick={()=>setDiscount(discount + 10)}>Increase discount</button>**

**</div>**

**);**

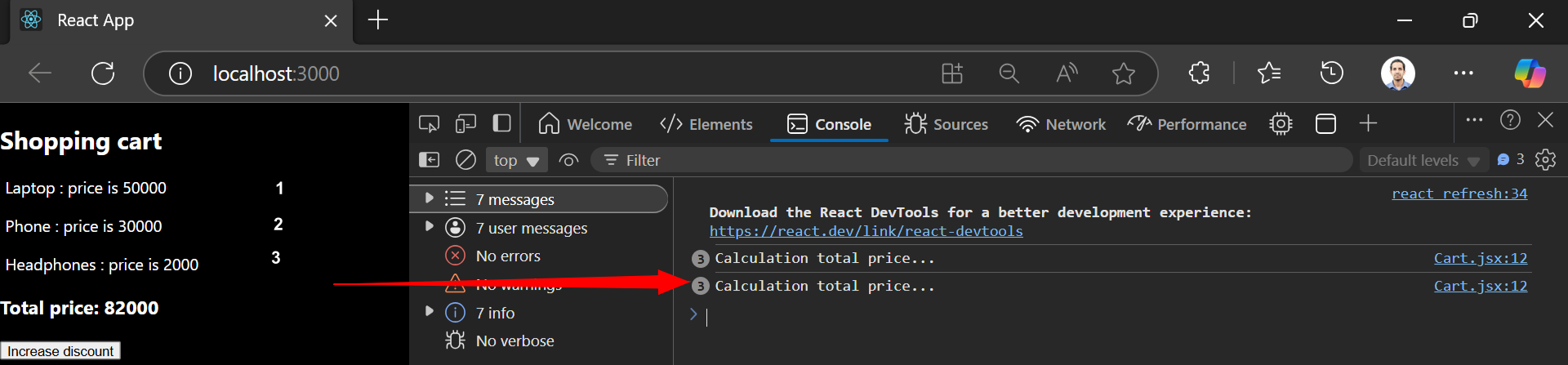
**}**

We have object list cart based on it we find totalprice

On button click I’m just increasing discount but in UI I’m not at all showing also **discount** I’ve not used anywhere inside function also only in discount I’ve used it

How to check is entire **Cart component rerendering** there I have put log inside **totalCalculation** method

When you click button many times **that many times same log displayed on console**

 **3, 12** numbers tells that many times that

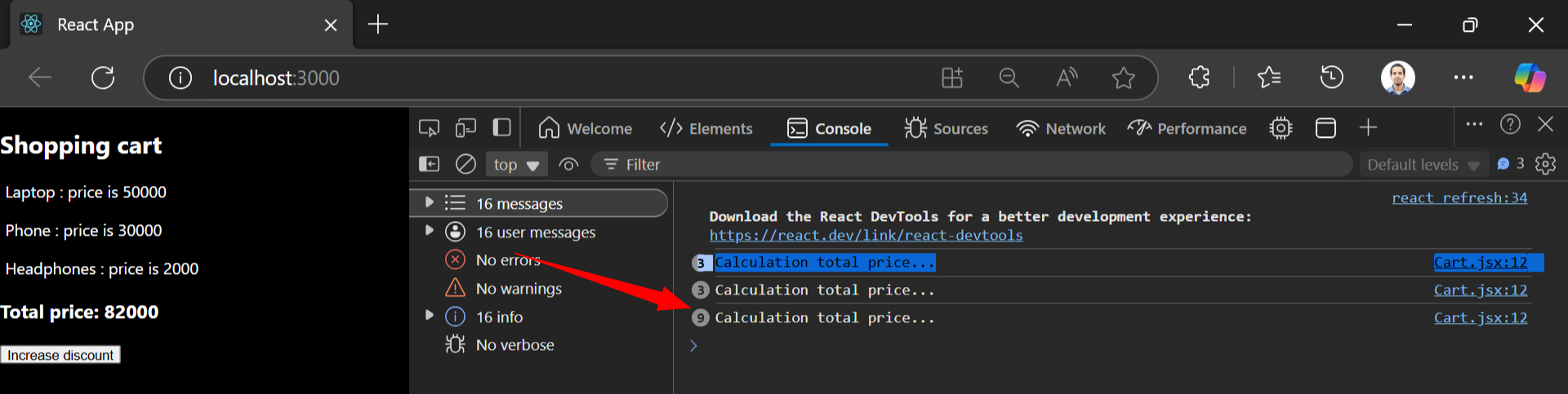
**console.log("Calculation total price...");**

log is printed

1:54:09

Above browser image it shows printed **3 Cart.jsx:12 Calculation total price**...

Why because in our cart object list we have 3 items so for 3 items it finds total price 3 time it’s printed



3 times I clicked button so 3,9 is showing 9 is showing because in same console line browser printed

When I click on button again it again calculates same total price for same items why because component re renders

To prevent rerender we use useMemo

#. **How to use usememo**

Earlier we calculated total price with cart.reduce, now here itself we use usememo

Syntax of usememo is same as useeffect remember useeffect we have to pass empty dependency array

Or else when content of array changes tat time only useeffect recalls

const totalPrice = useMemo(()=>{},[cart])

similarly above when usememo in that when cart changes that time on we call usememo again

total price logic code put inside usememo

**const totalPrice = useMemo(() => {**

**console.log("Calculation total price...");**

**return cart.reduce((total, item) =>**

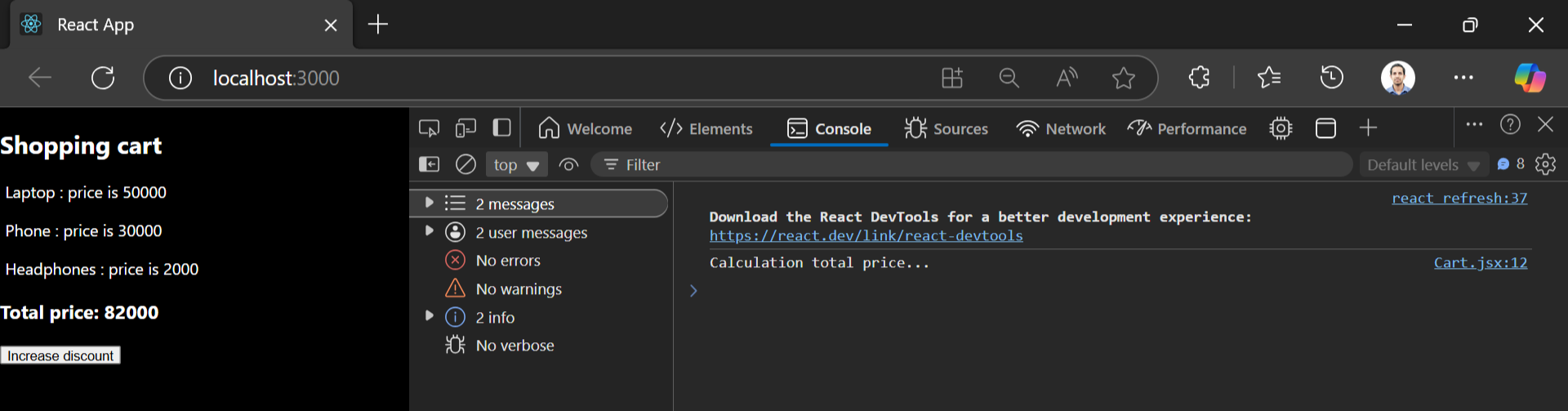
**total + item.price**

**, 0);**

**}, [cart])**

We write like above, cart we put as **dependency array** so when it changes that time only usememo recalls

Now because of using usememo



You click button any no of times see right side console log It’s printed only once

Discount button click method there also you can put log whenever you click button that time re log printed in browser but because usememor totalprice won’t calculate again

While optimizing usememo is useful

**#. Usecallback hook**

Similar to usememo but usememo prevents recalculation of total price result

But usecallback will prevent function from recalling

We explain both without usecallback and with usecallback

We have 2 components for this example

ChildButton.jsx child component and index.jsx parent component

From parent component index.jsx we call ChildButton.jsx which is child component and also we pass handleClick method to child component during calling childbutton.jsx

Below is code from pdf doc

**File:** ChildButton**.js (Child Component)**

**import React from "react";**

**const Button = ({ onClick }) => {**

**console.log("Button re-rendered!");**

**return <button onClick={onClick}>Click Me</button>;**

**};**

**export default Button;**

📄 **File: App.js (Parent Component)**

Created 2 usestate: count and darkmode

Next we import ChildButton into ParentIndex.jsx file

Next we write **handleClick** method which will be re-rendered every time

He told this method gets recreated every time

Next we give style inside ParentIndex.jsx it’s div related to darkmode means if our usestate variable darkMode is true then give one background or else other background

**<div style={{background:darkMode ? "#333" : "#fff",**

**color:darkMode ? "#fff" : "#333"**

**}}>**

For background => if darkmode true then show #fff darkmode false show #333

For color: if darkmode true then text color show white because in darkmode black text we can’t see

If darkmode false

Next print count

Next we call ChildButton and for that onclick we pass handleclick

**<ChildButton onClick={handleClick}/>**

Like above ChildButton is component we are calling and inside we pass onclick method

Next we create click button inside parentIndex.js on clicking it we toggle dark mode means if true make it false if fasle make it true

**setDarkMode(!darkMode)**

above code does that if true it makes it false if false makes it true

next in ChildButton.js child component we pass parameter onclick which we passed from Parentindex.js component

**function ChildButton({onClick}) {**

**console.log("Button re renders " + onClick);**

**return (**

**<div>**

**<button onClick={onClick}>Click me</button>**

**</div>**

**);**

**}**

Above see first we print console log in browser how many times it’s printed it tells **how many times childbutton rerenders**

Inside parent component we called Childbutton passed onclick handleclick event so with this in browser we see button **click me** onclicking this call comes to parenet component it calls handleclick and it increases **count**

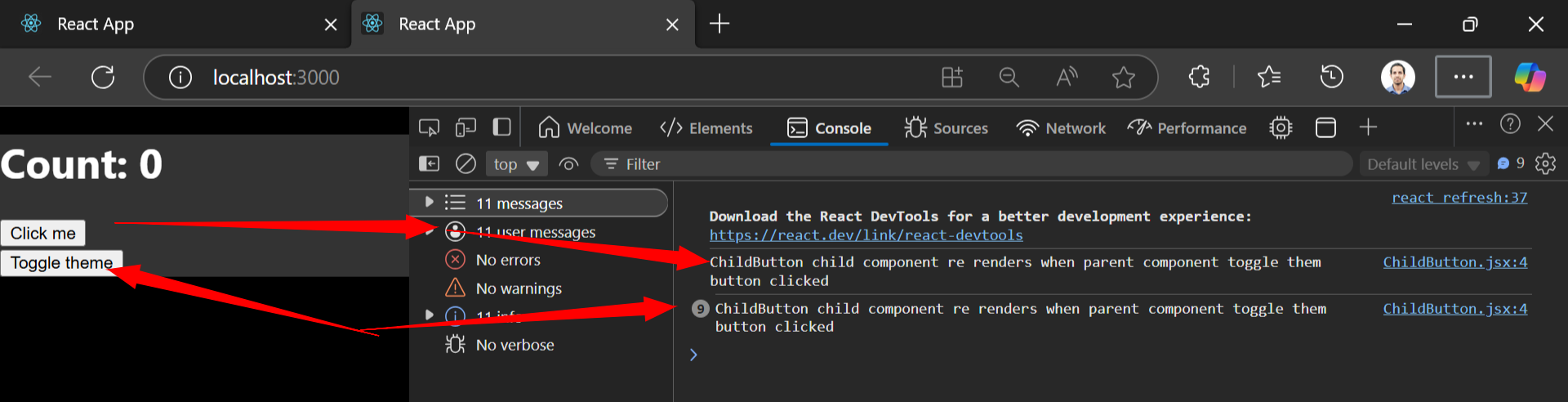
Now in parenet component there is another button **toggle theme** on clicking this button background changes to dark and normal it toggles true or false

Now imp part open developer tools when we click on **toggle theme button** it re renders child component

We see browser console

**Button re renders again and again whenever we click on** toggle theme button

Above



First time when page loads first log printed

Now next toggle theme button I clicked 9 times that’s why that log printed 9 times from child component

It’s unnecessary that while **clicking toggle them** **button** calling child component we have to stop it

Click me button from child component, toggle them button from parentcomponent

🛑 **Problem Explanation:**

● The handleClick function is **re-created** every time App re-renders.

● Since handleClick **changes on each render**, the Button component **also re-renders unnecessarily**.

● This **wastes performance**, especially when there are multiple child components.

We’ve to stop rendering child component on click toggle theme button

\*. **Use usecallback hook**

Handleclick fuction in parenetcomponent call it by using **useCallBack hook**

In ParentIndex.js we do this

useCallback similar to useeffect and usememo it’s inside we put callback and inside dependency array we put empty nothing

**const handleClick = useCallback(()=>{**

**setCount(count + 1);**

**},[])**

Like

Next go to childButton childcomponenet there we need to use useMemo

He told entire function in child component cover by curly bracket

**const ChildButton = React.memo(({onClick}) => {**

**console.log("ChildButton child component re renders when parent component toggle them button clicked ");**

**return (**

**<div>**

**<button onClick={onClick}>Click me</button>**

**</div>**

**);**

**})**

Like this we covered entire function with react.memo

A screenshot of a computer

AI-generated content may be incorrect.

Now if we click toggle them button childcomponent not re rendering

2:10

#. useTransition hook

**He told useTransition** not used much but in interview they may ask tell **some new hooks you used**

He told when we have to do some transition that time this hook will be useful

📌 **When to use?** Use when transitioning between UI states without blocking interactions.

✅ **Benefits:**

● Prioritizes rendering, improving performance in complex UIs.

**Without useTransition**

If you remove useTransition, the UI may freeze momentarily while filtering a large list.

He told like with useTransition: useTransition wil render first after that ui will display

If we’re searching in usetransition: then in search time whatever transition we want to display that will run after that it’l work

He told we’ve to write css also for transition

First we put input textbox to save value of input textbox we create state variable

**const [search, setSearch] = useState(0);**

**const [isPending, startTransition] = useTransition();**

Like above next

We create handleSearch method in that callback we call startTransition again we create new call back inside starttransition there we set value of search usestate variable

Next in input textbox onchange we call **handleSearch** method

To improve **performance before rendering** some ui reflection we want to show for that we

**const handleSearch = (e) => {**

**// setSearch(e.target.value)**

**startTransition(()=> setSearch(e.target.value));**

**console.log(search);**

**}**

I used withtout startTransition just by updating using setSearch and print log with console in browser and also

With startTransition

But I didn’t see any difference

**const Index = () => {**

**const [search, setSearch] = useState("");**

**const [isPending, startTransition] = useTransition();**

**const handleSearch = (e) => {**

**// setSearch(e.target.value)**

**startTransition(()=> setSearch(e.target.value));**

**console.log(search);**

**}**

**return (**

**<div >**

**<h1>This is usetransition hook</h1>**

**<input type="text" value={search} onChange={handleSearch}></input>**

**</div>**

**);**

**}**

Code

#. useRef hook

He told as like we use document.getelementbyid with some tag name like that only we use **useRef hook**

Document.getelementbyid we do that in react using useref

If we’re changing image: then in **image tag** we’ve to give ref **that’s how we use useref**

<https://pixabay.com/> from here took one image

right click on image and click **copy image addres**s you’ll get url that you can paste in src **of image tag instead** of manual downloading and giving system path

next we put **button** on clicking that **button image** should change

for that we use imgRef

const imageRef = useRef(null);

like above initial var we create

**imageRef** pass it to existing image

<img ref={imageRef} src="https://cdn.pixabay.com/photo/2017/10/10/07/48/hills-2836301\_1280.jpg"></img>

        Like how I take using document.getelementbyid **similarly** I take here img **using imageRef**

**Next** create one state in that show your image Boolean variable isFirstImage

Next we write method to call on button click handleImageChange

Here imageref.current means are we displaying current image if it’s true then

Set imageref.current.src= isFirstImage

isFirstImage will hold true or false

It’s like if already current image exists we check it and change url means we put new image url

**const handleImageChange = () => {**

**if (imageRef.current) {**

**imageRef.current.src = isFirstImage**

**?**

**"https://cdn.pixabay.com/photo/2017/08/15/09/40/church-2643296\_1280.jpg"**

**:**

**"https://cdn.pixabay.com/photo/2013/10/12/17/45/temple-194617\_1280.jpg"**

**setIsFirstImage(!isFirstImage);**

**}**

**}**

If our isFirstImage is true then we show different image if it’s false then also we show different image

Next we toggle setIsFirstImage means if it’s true make it false if it’s false make it true

Next onclicking button we change image so we add button click method that is onclick we call handleImageChange

**const RefIndex = () => {**

**const imageRef = useRef(null);**

**const [isFirstImage, setIsFirstImage] = useState(true);**

**const handleImageChange = () => {**

**if (imageRef.current) {**

**imageRef.current.src = isFirstImage**

**?**

**"https://cdn.pixabay.com/photo/2017/08/15/09/40/church-2643296\_1280.jpg"**

**:**

**"https://cdn.pixabay.com/photo/2013/10/12/17/45/temple-194617\_1280.jpg"**

**setIsFirstImage(!isFirstImage);**

**}**

**}**

**return (**

**<div >**

**<h1>Change image using useref</h1>**

**<img ref={imageRef} src="https://cdn.pixabay.com/photo/2017/10/10/07/48/hills-2836301\_1280.jpg"></img>**

**<button onClick={handleImageChange}>Change image</button>**

**</div>**

**);**

**}**

Above is button

#. useId hook very easy hook

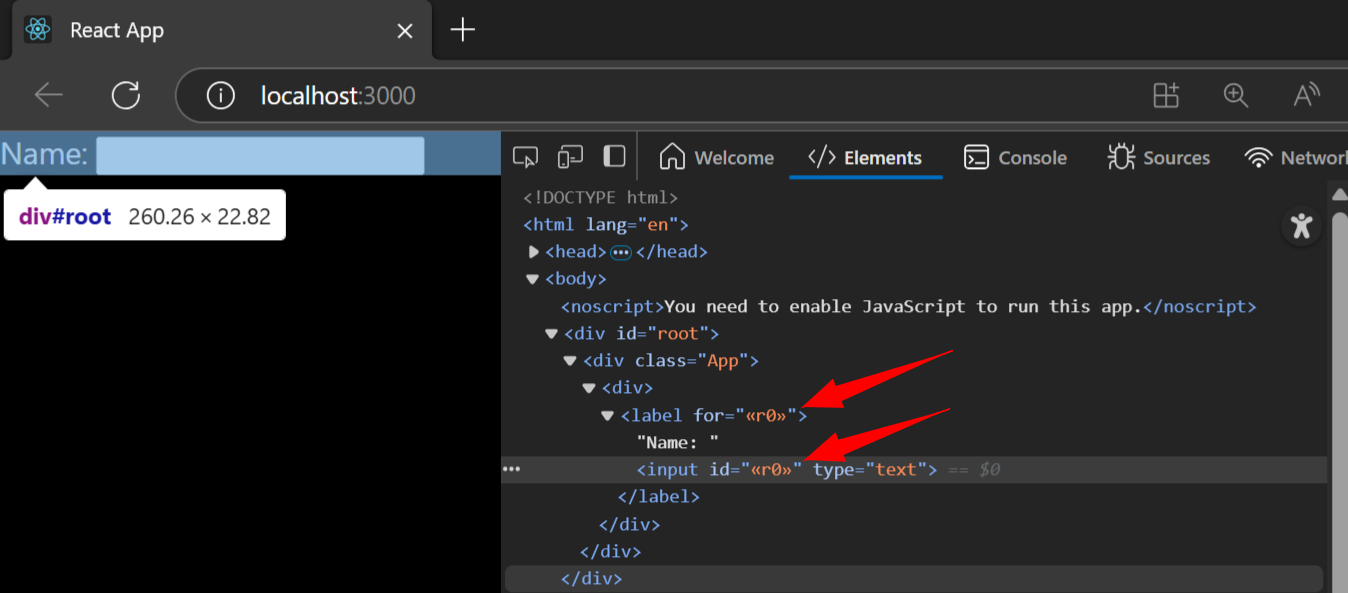
He told useId is very easy

When we have to generate many uuids

When we do map that time it’s necessary to generate random ids to generate random ids we use **math.random**

Using math.random we can generate unique ids

Our idIndex will generate random ids



See above we used useid in id and used for label and input both same unique id comes

He told we can also use this map but recommended not to use

**Why Not Use Math.random() or manual ids?**

* useId ensures **consistent IDs between server and client**, preventing hydration mismatches in SSR.
* It's **unique**, even across multiple instances of a component.
* It’s **stable** (doesn’t change on re-render).

#. Custom hook using useFetch fetching data

2:31

Create usestate variable data initial value null

Create other variables also loading and error

Next for UseEffect method pass url as parameter

Next we create async method

When we fetch data asynchrounously make setloading as true

useEffect(()=>{

        const fetchData = async()=>{

            setLoading(true)

            try{

                const response = await fetch(url)

                if(!response.ok){

                    throw new Error("Failed to fetch data")

                }

                const result = await response.json()

                setData(result)

            }

            catch (error){

            }

        }

    })

Like above we use useFetch

2:36

Again now we use or call api data from jsonplaceholder.com

In CustomIndex.jsx we don’t have to use usestate and all

Just import UseFetch.jsx and use it call it

Next destructure UseFetch we destructure data,loading,error so while destructuring this will be created as separate variable from usefetch

<https://jsonplaceholder.typicode.com/users>

above url we use

**const {data,loading,error} = UseFetch("https://jsonplaceholder.typicode.com/users");**

like above destructure UseFetch and pass url to UseFetch where UseFetch is filename

next in

CustomIndex.jsx using map we show those data

import { useState, useEffect } from "react";

**const UseFetch = (url) => {**

**const [data, setData] = useState(null);**

**const [loading, setLoading] = useState(true);**

**const [error, setError] = useState(null);**

**useEffect(()=>{**

**const fetchData = async()=>{**

**setLoading(true)**

**try{**

**const response = await fetch(url)**

**if(!response.ok){**

**throw new error("Failed to fetch data")**

**}**

**const result = await response.json()**

**setData(result)**

**setError(null)**

**}**

**catch (error){**

**setError(error)**

**}**

**finally{**

**setLoading(false)**

**}**

**};**

**fetchData();**

**},[url]);**

**return {data,loading,error};**

**}**

**export default UseFetch;**

above UseFetch

**const CustomIndex = () => {**

**const {data,loading,error} = UseFetch("https://jsonplaceholder.typicode.com/users");**

**if(loading) return <p>Loading...</p>**

**if(error) return <p>Error : {error}</p>**

**return (**

**<ul >**

**{data.map(user =>**

**<li key={user.id}>{user.name}</li>**

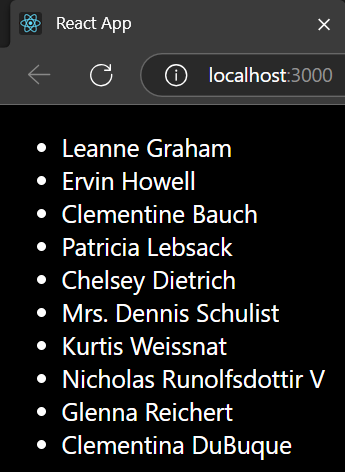
**)}**

**</ul>**

**);**

**}**

and above is CustomIndex



Output looks like above