Routing, Prop drilling, Context API

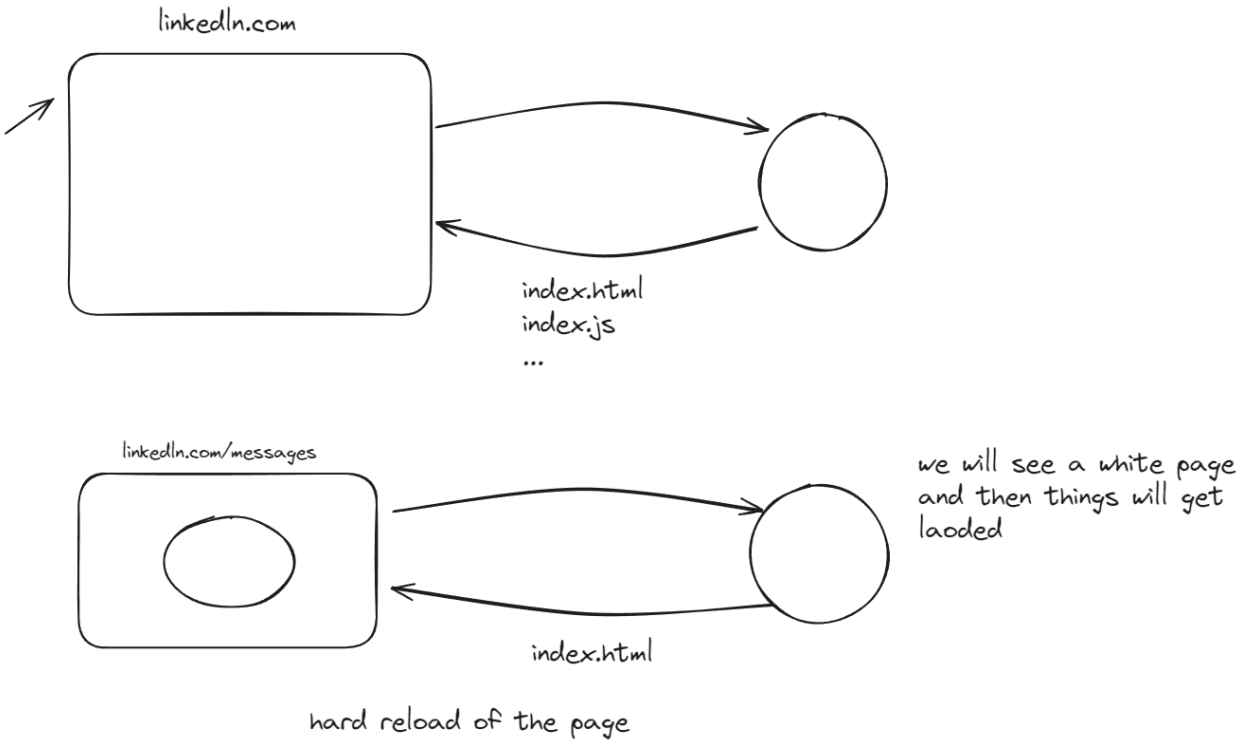
Routing

Jargon

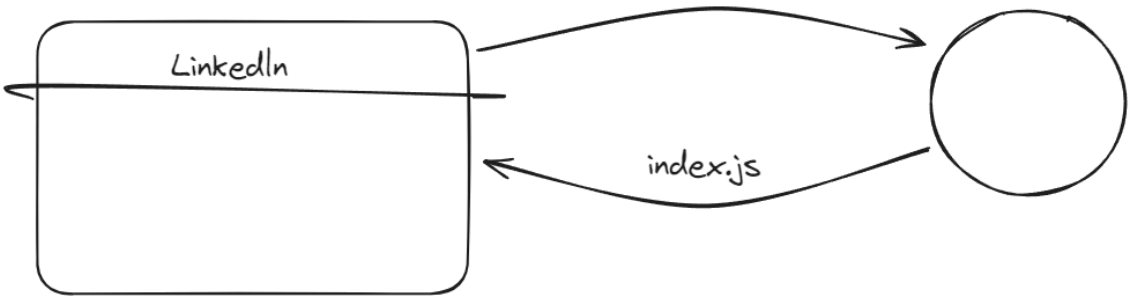
1. Single page application
2. Client side bundle
3. Client side routing

React is a single page application.

Before React:



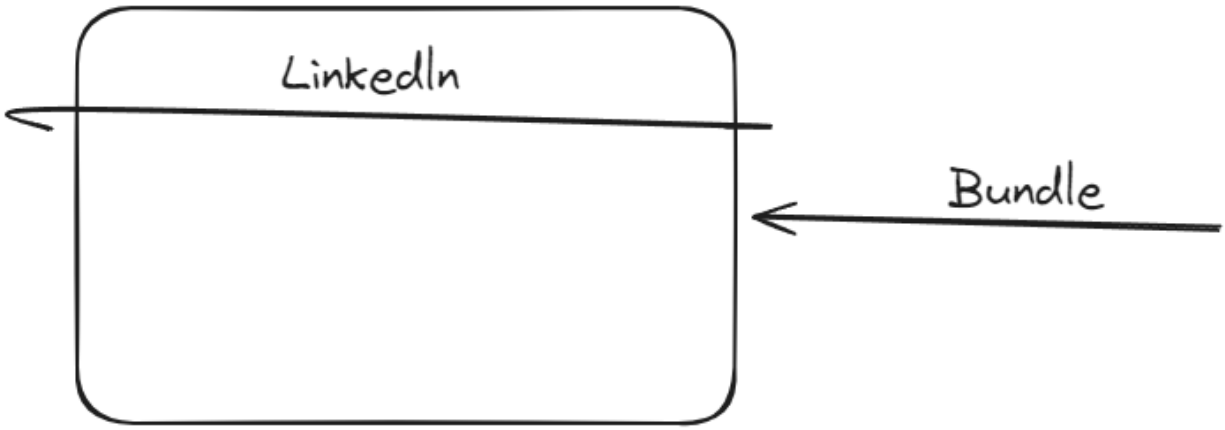
React -> Single Page Applications



If we go to another tab there is no more need for more html javascript etc . All it come in single go
We are routing on client (that this will render -this will render etc)

Does no more index.js etc comes once page is loaded , sort off
Only Client-side routing happens

There is optimization which is sending bundles.



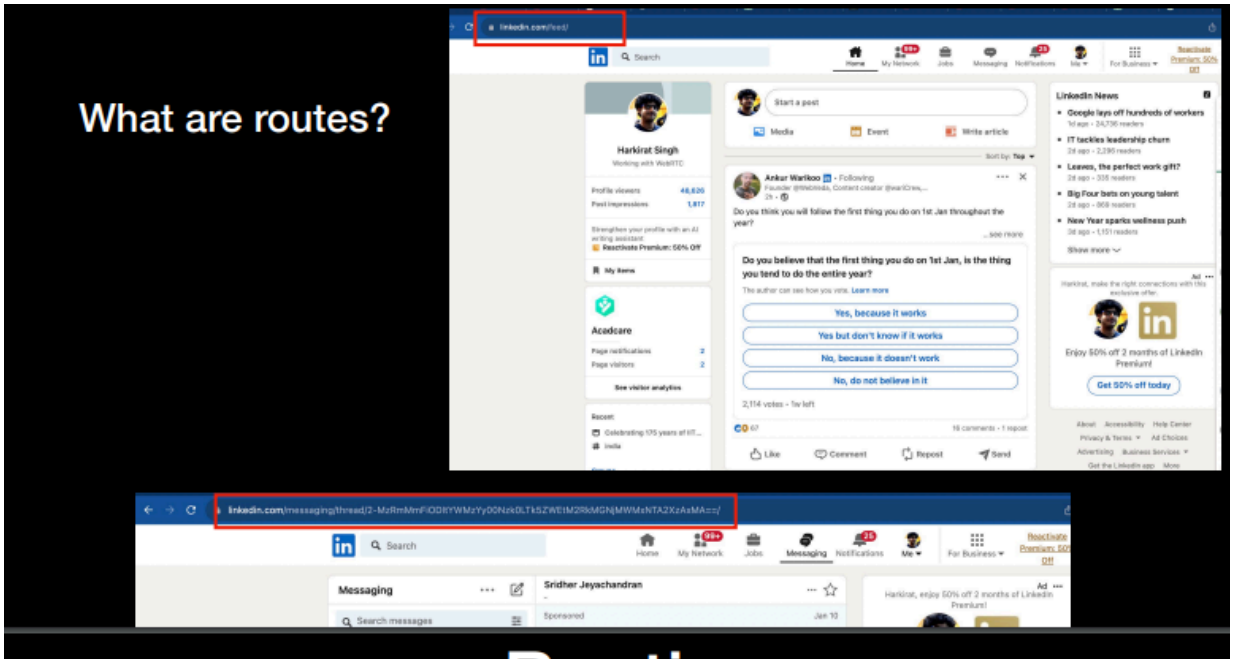
Client-side bundle

Bundle we get from server
Single js file contain all the page and do client side routing

Client side routing

The code which we write as coder , which is users are in messages page then we will show them their messages.

Routing



How to do routing in react??

react-router-dom

(https://reactrouter.com/)

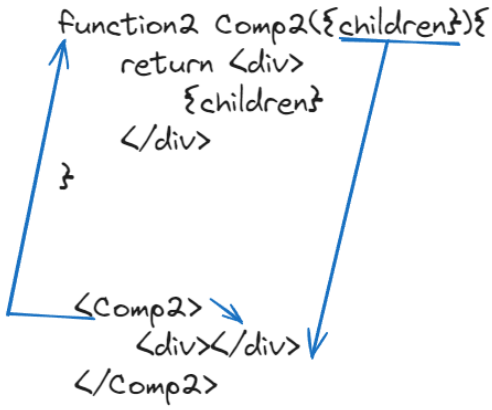
In react component can take props as input can be a number, string or another component.

One component can use another component and render it inside.

```
function App() {

  return (
    <BrowserRouter>
      <Routes>
        <Route path="/dashboard" element={<Dashboard />} />
        <Route path="/" element={<Landing />} />
      </Routes>
    </BrowserRouter>
  )
}
```

“/dashboard” : dashboard page

“/” : Landing page



Lets first create two component for landing page and dashboard

Dashboard.jsx

```
export function Dashboarc(){
    return <div>
        Dashboard page
    </div>
}
```

Landing.jsx

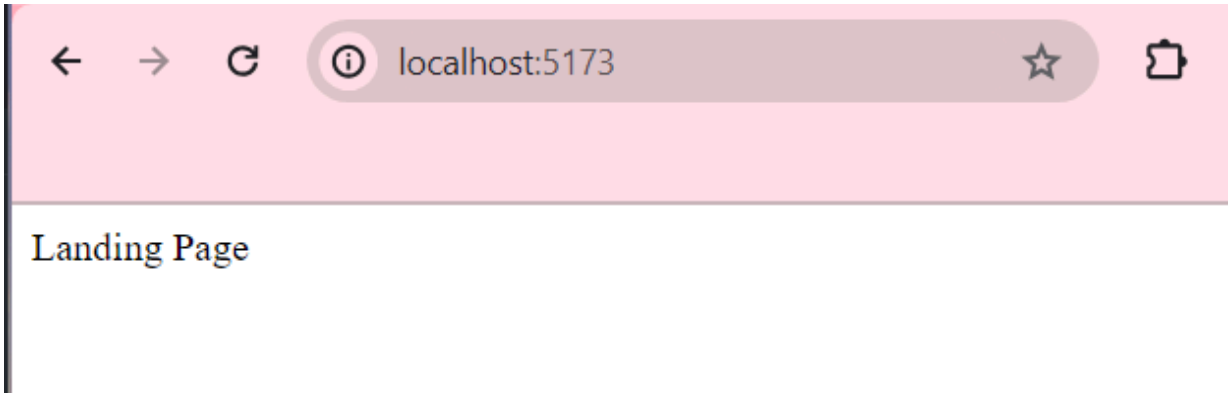
```
export function Landing(){
    return <div>
        Landing Page
    </div>
}
```

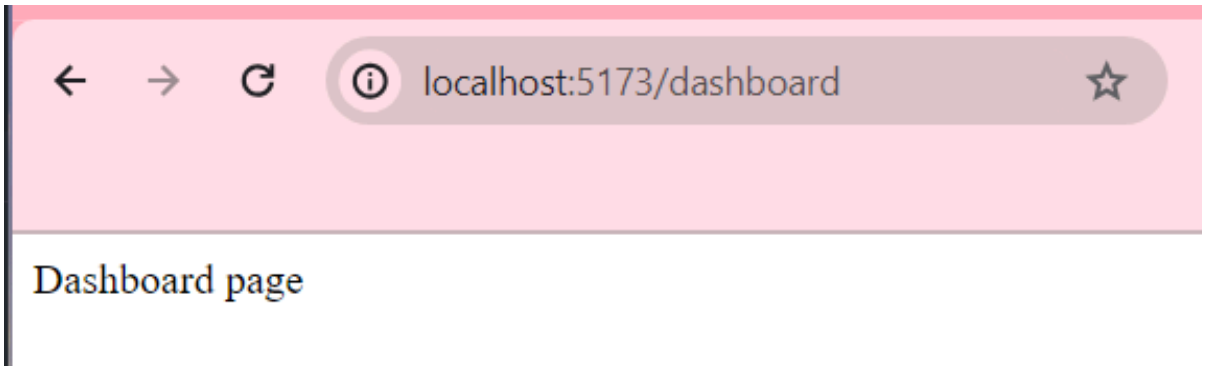
Now lets works on our logic to do client side routing

```
import { BrowserRouter, Route, Routes } from 'react-router-dom'
import { Dashboard } from './components/Dashboard'
import { Landing } from './components/Landing'
```

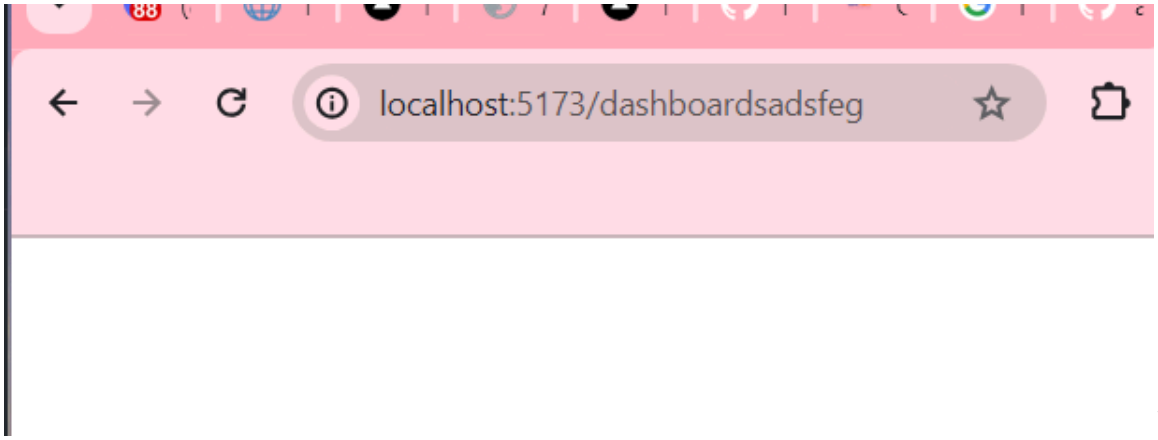
```jsx
function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path='/dashboard' element={<Dashboard />} />
        <Route path='/' element={<Landing/>} />
      </Routes>
    </BrowserRouter>
  )
}

export default App
```

Focus on the URL of each page





What if we go to random route

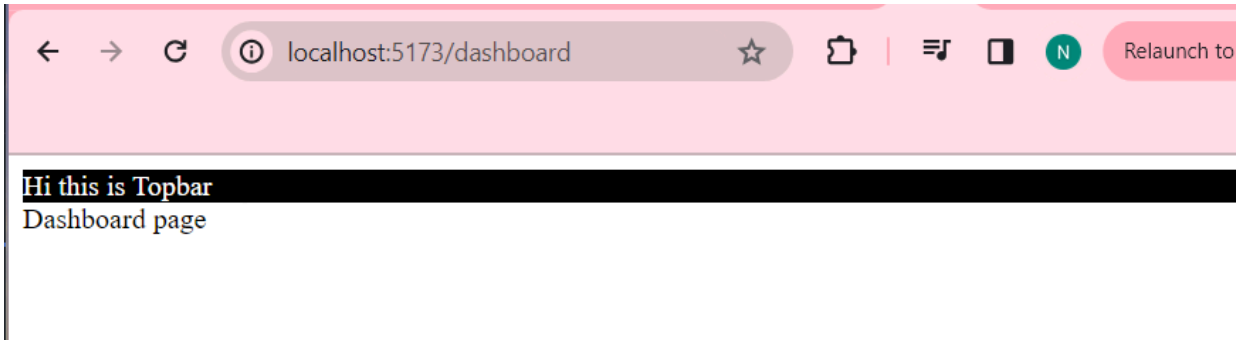


\

We see nothing when we enter the wrong url

If we see LinkedIn we notice that they have fixed topbar

```
function App() {
  return (
    <div>
      <div style={{background:"black", color:"white"}}>
        Hi this is Topbar
      </div>
      {/* Browser router is just a component like any other route */}
    <BrowserRouter>
      <Routes>
        <Route path='/dashboard' element={<Dashboard />} />
        <Route path='/' element={<Landing/>} />
      </Routes>
    </BrowserRouter>
    </div>
  )
}
```

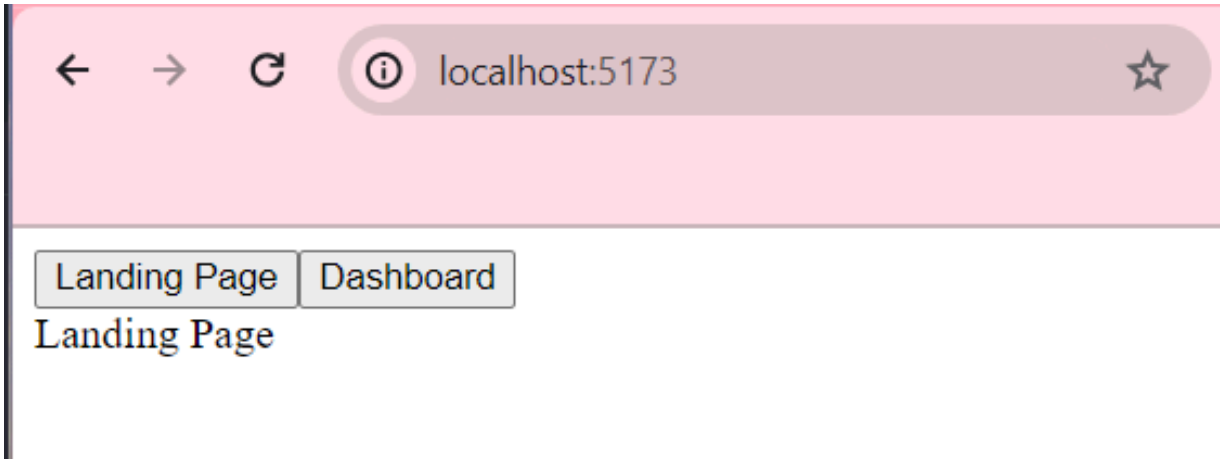


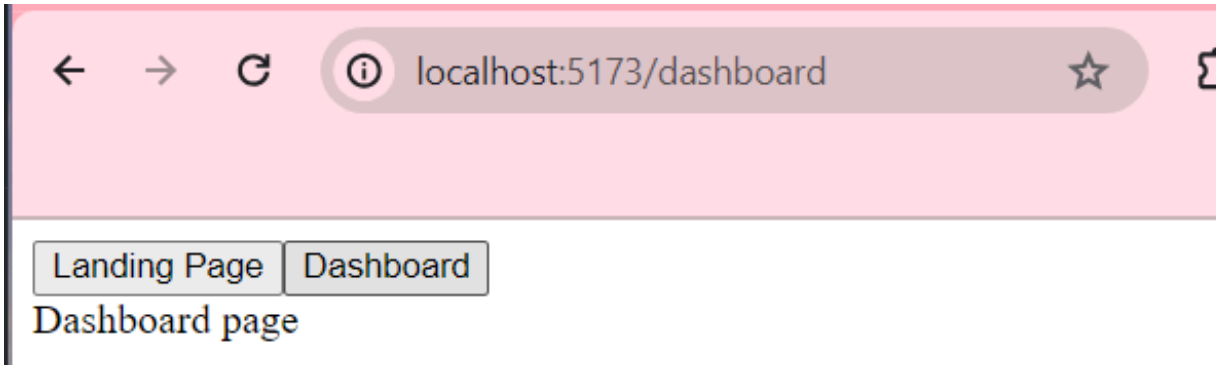
When we switch between two pages (dashboard and landing) does a fresh index.html comes from the server??

If we are doing client side routing properly then we will not get the fresh bundle every time. We dont need server anymore.

```
import { BrowserRouter, Route, Routes } from 'react-router-dom'
import { Dashboard } from './components/Dashboard'
import { Landing } from './components/Landing'

function App() {
  return (
    <div>
     <div>
      <button onClick={()=>{
        window.location.href = '/'; //global location object we have
access to it in DOM
      }}>Landing Page</button>
     </div>
     <button onClick={()=>{
        window.location.href = '/dashboard';
      }}>Dashboard</button>
    <BrowserRouter>
      <Routes>
        <Route path='/dashboard' element={<Dashboard />} />
        <Route path='/' element={<Landing/>} />
      </Routes>
```

```
    </BrowserRouter>
    </div>
  )
}

export default App
```



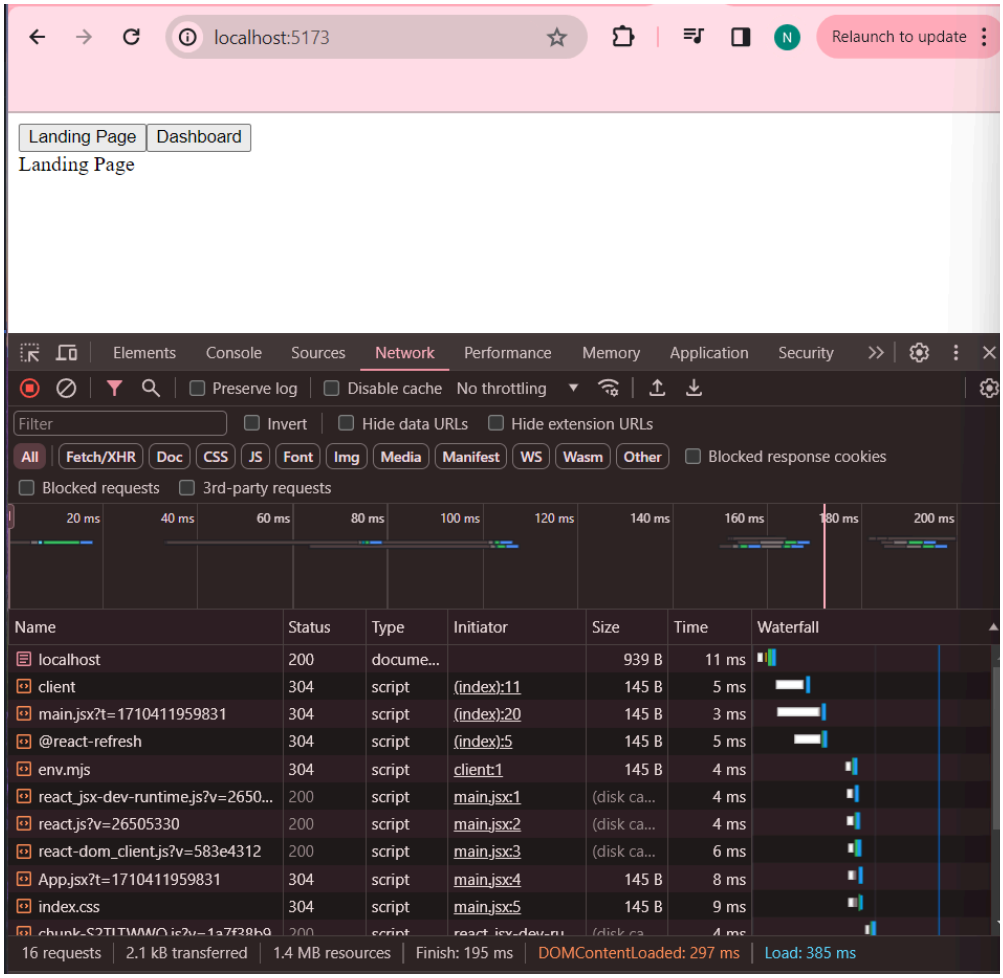


We can see more about location on the browser console

We can change url by location.href= “/dashboard”;

Is there a flaw here???

Lets switch from dashboard page to Landing page and see whether we properly doing cliend side rendering or not. We can verify it by checking the network tab in inspect.



Solution is **useNavigate() hook**

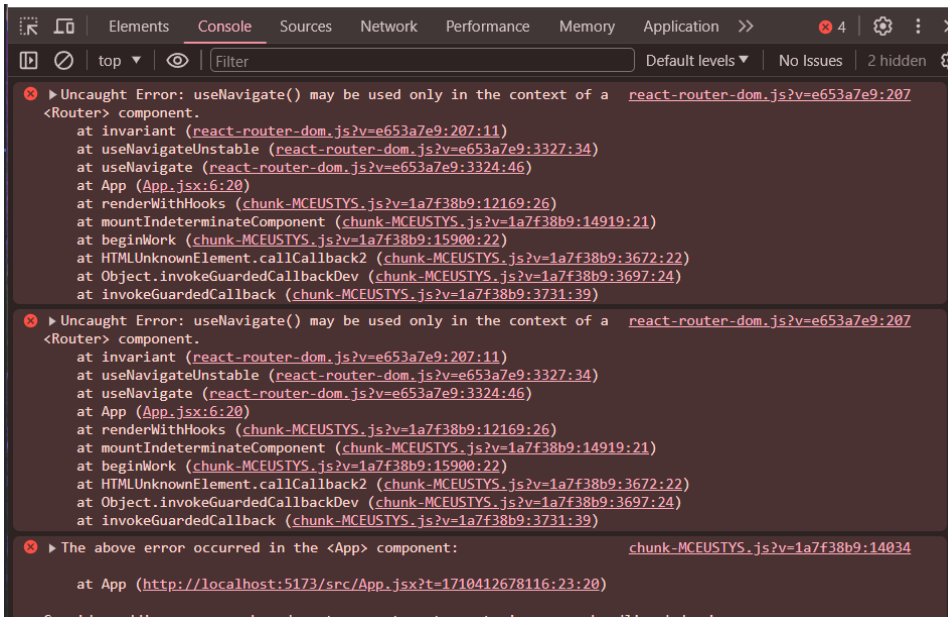
It will lets us navigate from one page to another. It makes sure it is not doing hard reload of the page . It is basically changing the route and keeping the same client bundles and changing the page because route has changed.

src > components > Dashboard.jsx > Dashboard

```jsx
import { useNavigate } from "react-router-dom"


export default function Dashboard() {
    const navigate = useNavigate();

    function handleClick() {
        navigate('/')
    };

    return <div>
        Dashboard
        <button onClick={handleClick}>Click to navigate</button>
    </div>
}
```

```jsx
function App() {
  const navigate = useNavigate();
  return (
    <div>
     <div>
      <button onClick={()=>{
        navigate("/");
      }}>Landing Page</button>
     <button onClick={()=>{
        navigate("/dashboard");
      }}>Dashboard</button>
      </div>
    <BrowserRouter>
      <Routes>
        <Route path='/dashboard' element={<Dashboard />} />
        <Route path='/' element={<Landing/>} />
      </Routes>
    </BrowserRouter>
    </div>
  )
}
```

It is not working



This **useNavigate()** hook expects that whenever it is used it is inside a component that is inside **<BrowserRoute>** component

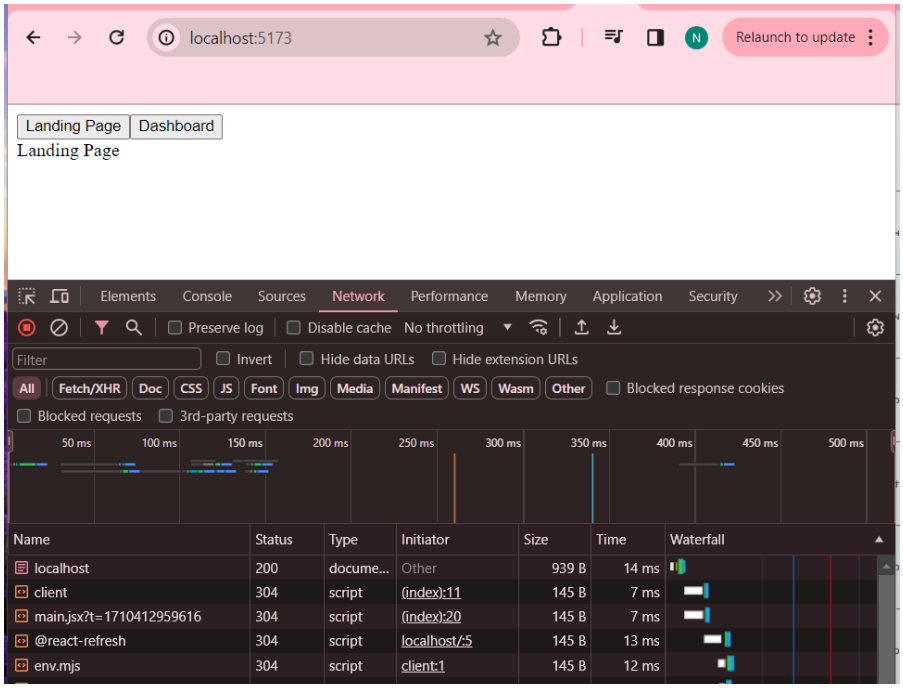
useNavigate() hook can be invoked only inside a component which is inside the **<BrowserRoute>** component .

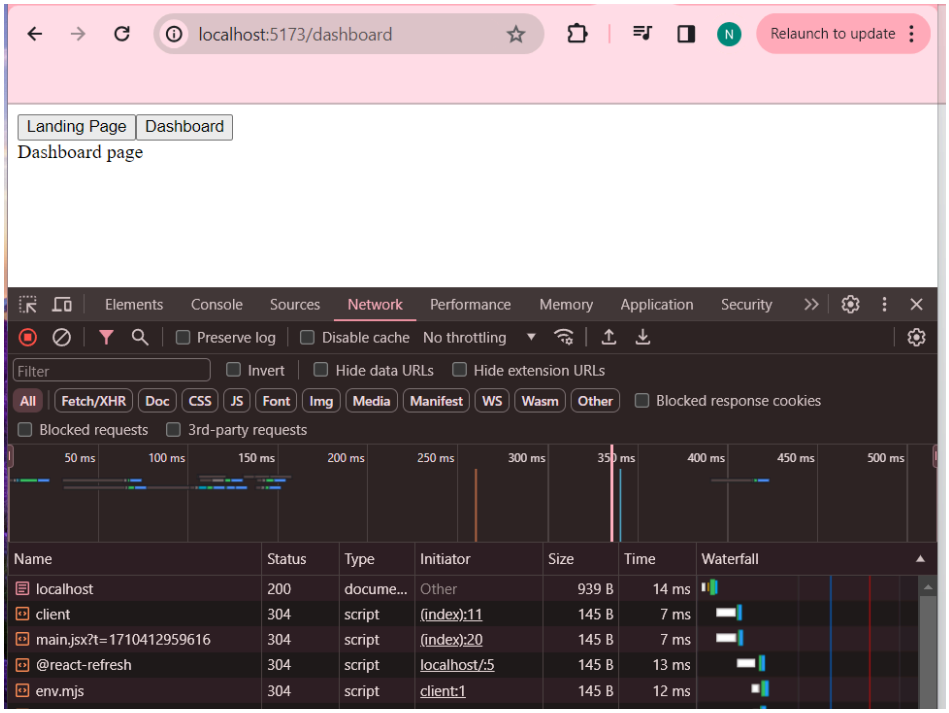
Solution:

```
import { BrowserRouter, Route, Routes, useNavigate } from
'react-router-dom'
import { Dashboard } from './components/Dashboard'
import { Landing } from './components/Landing'

function App() {

  return (
    <div>
    <BrowserRouter>
    <Appbar/>
      <Routes>
        <Route path='/dashboard' element={<Dashboard />} />
        <Route path='/' element={<Landing/>} />
```

```
      </Routes>
    </BrowserRouter>
    </div>
  )
}

function Appbar(){
  const navigate = useNavigate();
  return(
    <div>
    <button onClick={()=>{
      navigate("/");
    }}>Landing Page</button>
   <button onClick={()=>{
      navigate("/dashboard");
    }}>Dashboard</button>
    </div>
  )
}

export default App
```

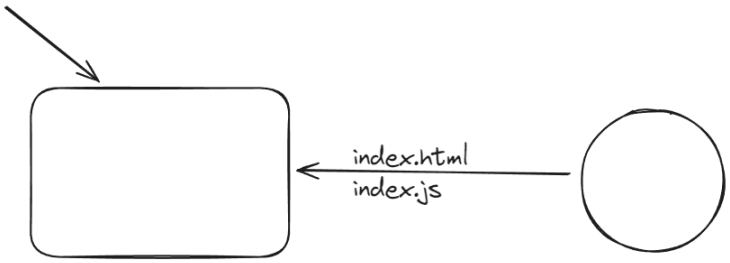
Lets see network tab to verify our solution





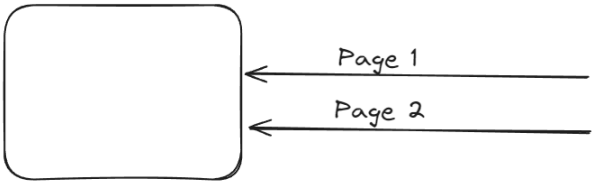
Lazy Loading

Client comes to the page



The problem with this approach is that suppose a client want to just visit the landing page why should we send him the whole index.js which contains logic of multipe pages

Suppose a person is in page 1 should it get only the code of that page only and it will get code of other page only when it route changes



Incrementali giving the website when the route changes this

Lazy Loading

Wrap your component inside the React.lazy() take function as input and start lazily import the component when it is needed.

First add default export

We can add a single default export of the component and multiple const export in a component.

```
export default function Dashboard(){
    return <div>
        Dashboard page
    </div>
}
```

We will import it like this now

```
import  Dashboard  from './components/Dashboard'
```

SolutionPrototype

```
import { lazy } from 'react';
import { BrowserRouter, Route, Routes, useNavigate } from
'react-router-dom'
const  Dashboard = lazy(()=> import('./components/Dashboard'));
const  Landing  = lazy(()=> import('./components/Landing'));

function App() {

  return (
    <div>
    <BrowserRouter>
    <Appbar/>
      <Routes>
        <Route path='/dashboard' element={<Dashboard />} />
        <Route path='/' element={<Landing/>} />
      </Routes>
    </BrowserRouter>
    </div>
```

```
  )
}
```

This doesn’t work properly there is one error which needs to be fixed.

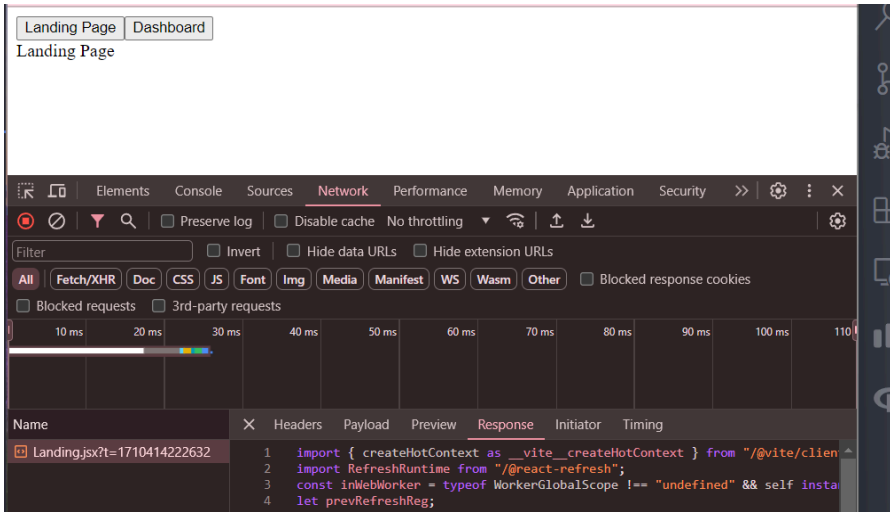
Suspense API

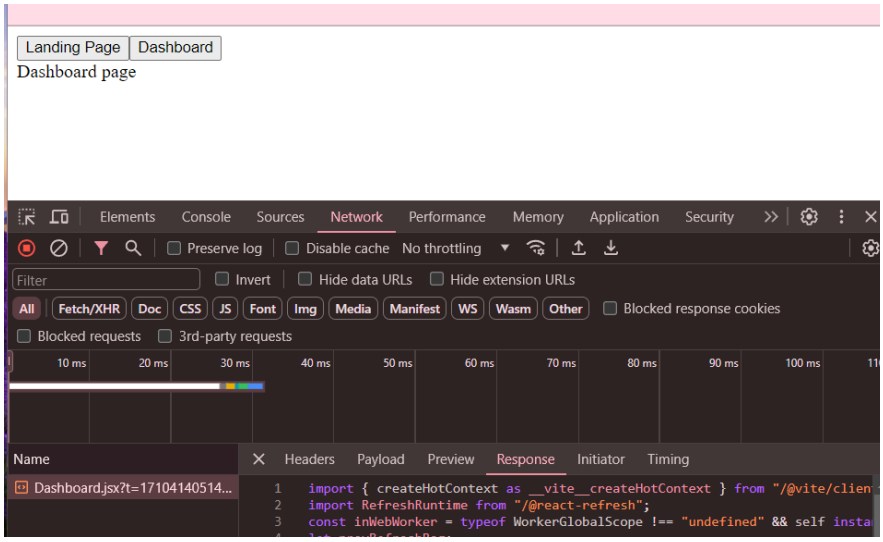
```
function App() {

  return (
    <div>
    <BrowserRouter>
    <Appbar/>
      <Routes>
        <Route path='/dashboard' element={<Suspense
fallback={"loading..."}><Dashboard /></Suspense>} />
        <Route path='/' element={<Suspense
fallback={"loading..."}><Landing /></Suspense>} />
      </Routes>
    </BrowserRouter>
    </div>
  )
}
```

Suspense is use in case when we are doing, Asynchronous component fetching , asynchronous data fetching. We don’t have immediately access to component . It comes from backend to frontend .

During that time what will it render??

If this data is not yet there render the fallback which in case is “Loading…”



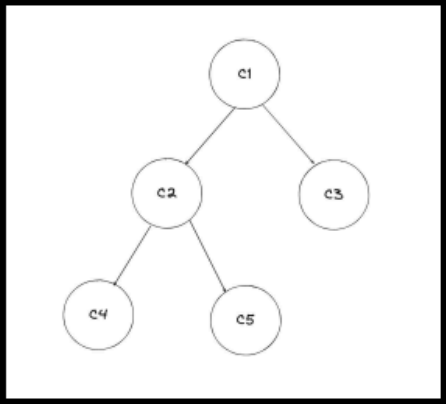


Q/Na

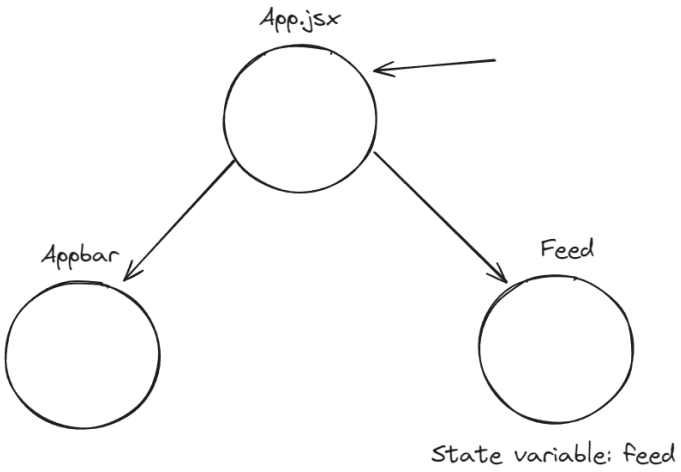
1. Can we **<Link />** instead of the using useNavigation() ??? Yes But there are some usecase where we have to use , useNavigation()

Prop Drilling

Before we begin, how do you think one should one manage state?
1. Keep everything in the top level component (C1)
2. Keep everything as low as possible
   (at the LCA of children that need a state)



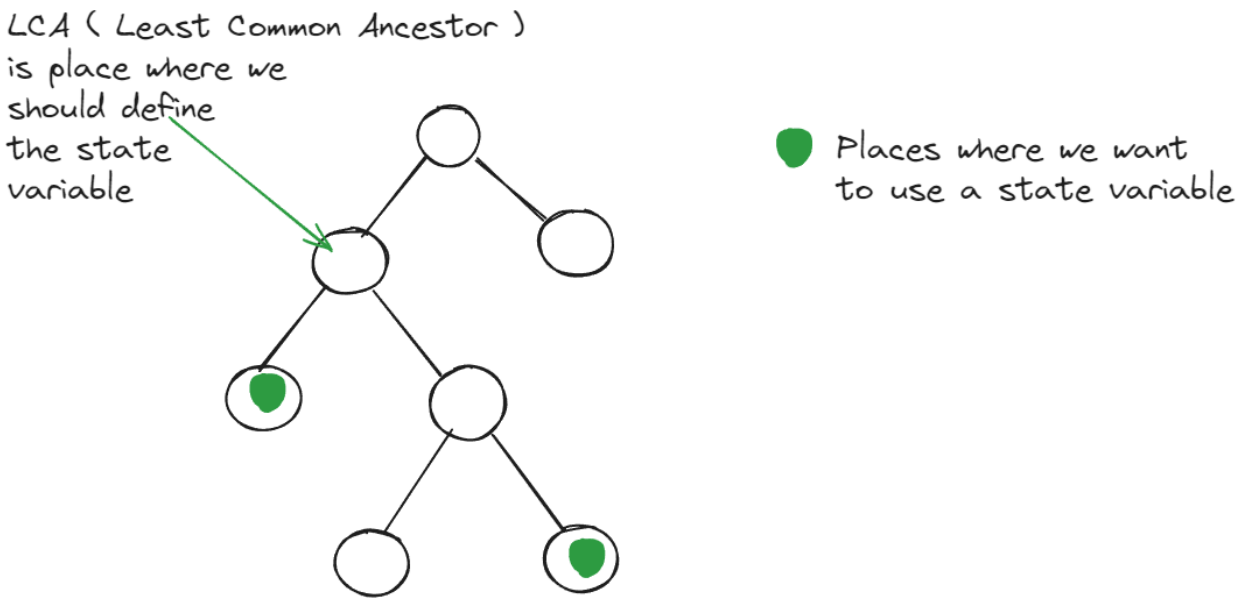
Ideal answer is to keep it at low level component



We will store the feed state variable in the Feed Component , this will decrease unnecessary re-rendering whenever feed changes

Hence we push the state down. and prevent App.jsx and Appbar.jsx from re-rendering

Least Common Ancestor(LCA)



const [feed,setFeed] = useState([])

Push it down as much as possible

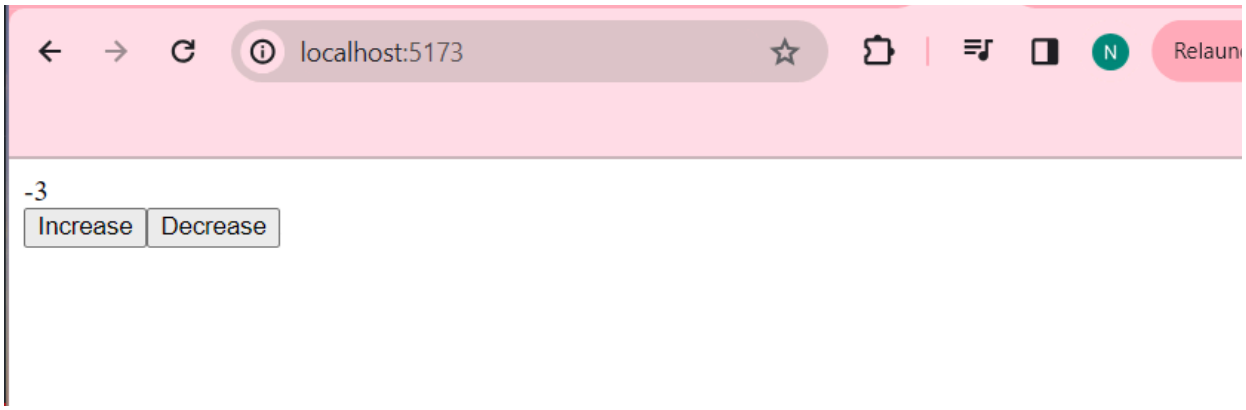
Lets understand more with an example:

Global count.

Two button one for increasing and other for decreasing.

```
import { useState } from "react"

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

  return (
    <div>
      <Count count={count}/>
      <Buttons count={count} setCount={setCount}/>
    </div>
  )
}
```

```
function Count({count}){
  return <div>
    {count}
  </div>
}

function Buttons({count,setCount}){
  return <div>
    <button onClick={()=>{
      setCount(count+1)
    }}>Increase</button>
    <button onClick={()=>{
      setCount(count-1)
    }}>Decrease</button>
  </div>
}

export default App
```



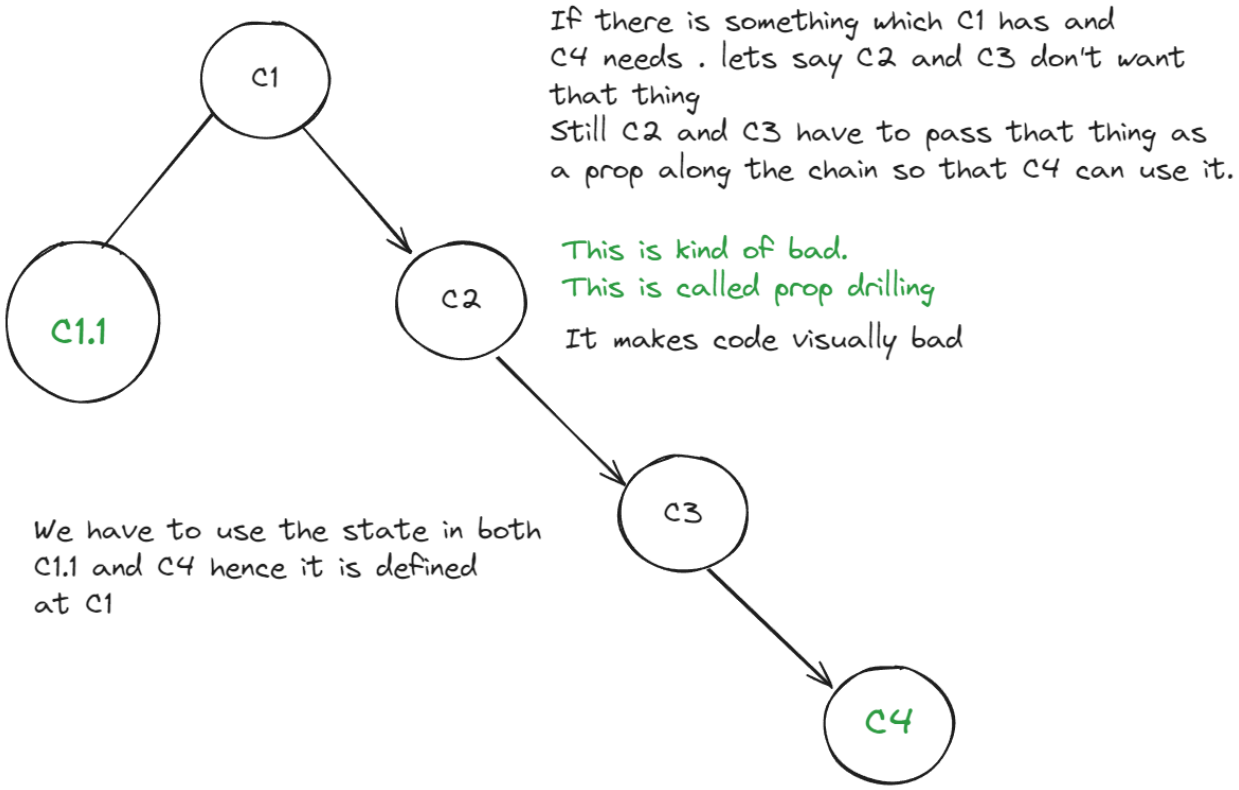
Now lets say we want Buttons to be part of the Count component .

To access it we have to pass it down the chain.

```
function Count({count,setCount}){
  return <div>
    {count}
    <Buttons count={count} setCount={setCount}/>
```

```
  </div>
}
```

It will start to look ugly as the chain grows. It get slightly unmanagable to keep on passing props , this is called **prop drilling.**

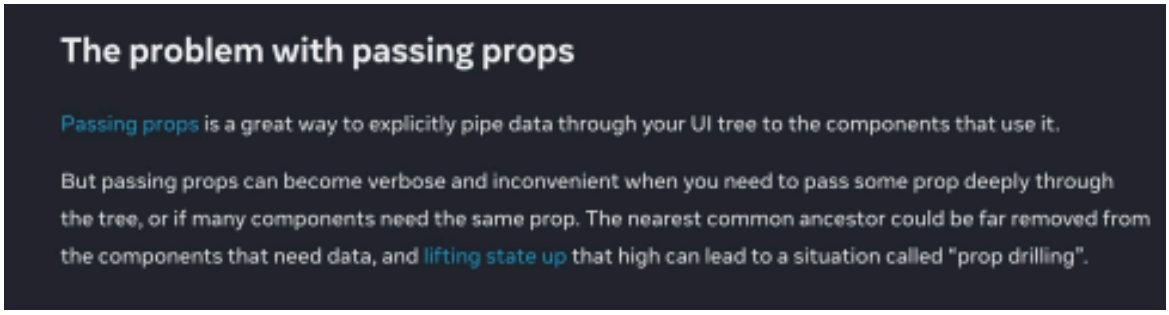


Prop drilling has nothing to do with re-renders!!!

Prop drilling is just means the syntactic uneasiness when writing code.

It makes code highly unreadable.

Problem with passing props



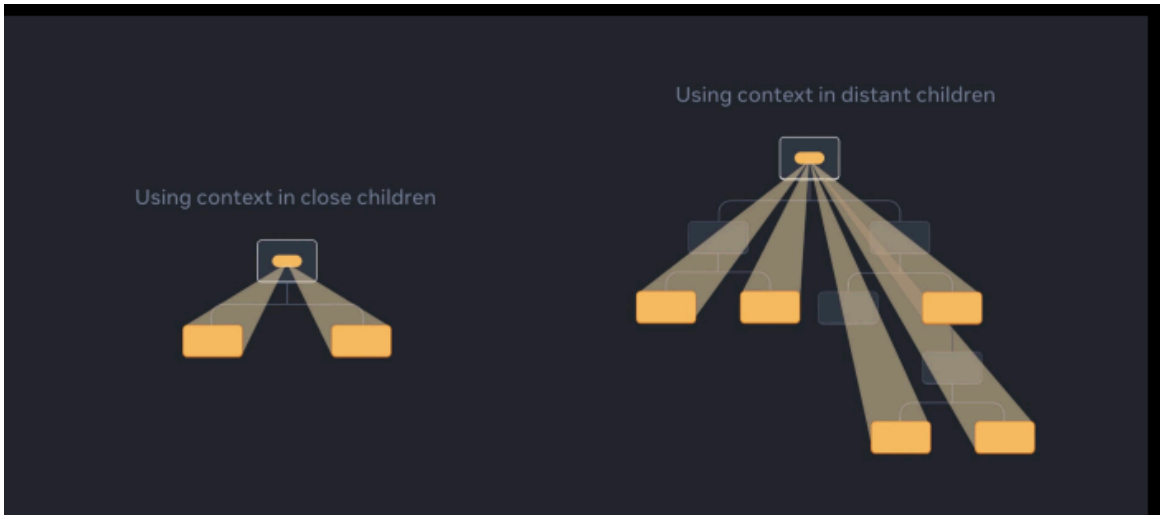
https://react.dev/learn/passing-data-deeply-with-context

Context API

It let us teleport state component

Very good resource to learn:

https://react.dev/learn/passing-data-deeply-with-context



Teleport the state variable without prop drilling.

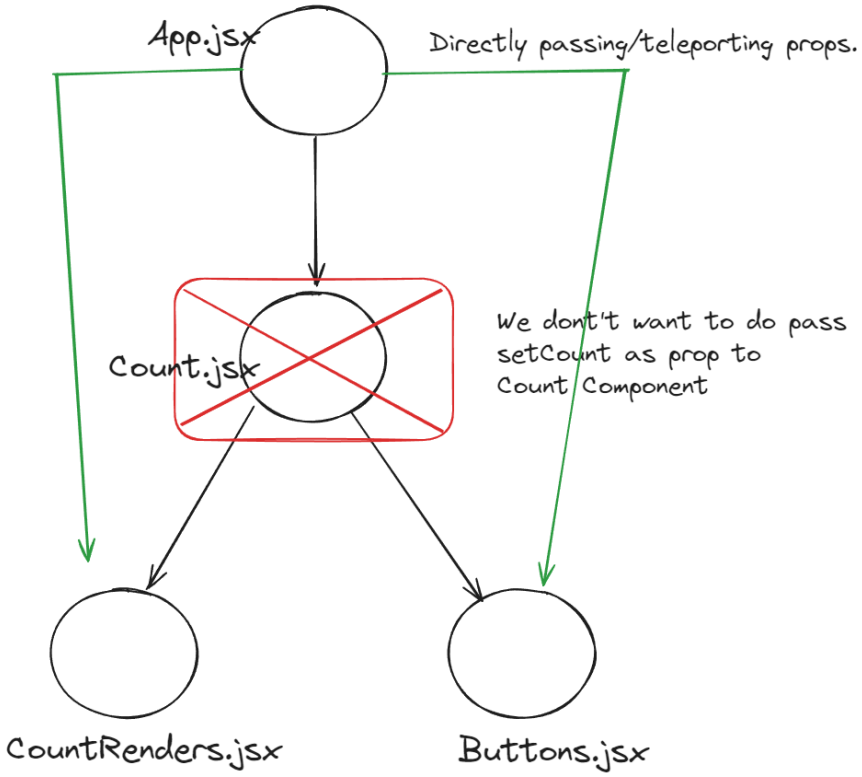
If you use context api , you are pushing your state management outside the core react components.

Lets us keep all state logic outside your core react component.

Let's create a simple Counter application, first without the context API and then with it

Things to learn -
createContext
Provider
useContext hook

Lets see our Component structure



First thing we have to create is context which will help us teleport state variable.

Write the context

Wrap anyone that wants to use the teleported value inside a provider.

```
import { useContext, useState } from "react"
import { CountContext } from "./Context";

function App() {
```

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

  return (
    <div>
      {/* we need to wrap all the component inside provider */}
      <CountContext.Provider value={count}>
        <Count count={count} setCount={setCount}/>
      </CountContext.Provider>
    </div>
  )
}

function Count({setCount}){
  return <div>
    <CountRenderer />
    {/* Directly teleported */}
    <Buttons  setCount={setCount}/>
  </div>
}

function CountRenderer(){
  const count =useContext(CountContext);
  // acess to count without being passed as props
  return <div>
    {count}
  </div>
}

function Buttons({setCount}){
  const count = useContext(CountContext);
// const {count,setCount} = useContext(CountContext);
  return <div>
    <button onClick={()=>{
      setCount(count+1)
    }}>Increase</button>

    <button onClick={()=>{
      setCount(count-1)
    }}>Decrease</button>
  </div>
```

```
}

export default App
```

Q/Na:

1. What is difference between Redux/Recoil and ContextAPI
2. Most efficient way to write React code without using state management tools like Redux and Recoil was Reducer.
3.

```
useEffect(()=>{
//does authentication
        setLoading(false);
        setAuth(“falsed”);
},[])
if(loading){
        return <div>
        </div>
}
```