1. **Component:**

Its describe a part of the user interface. They are re-usable and can be nested inside other components.

* 1. Stateless Functional Component( Java script Functions)

E.g.

function Welcome(props){

return <h1>Hello, {props.name}</h1>;

}

* 1. Stateful Class Component (Class extending Component class)

E.g.

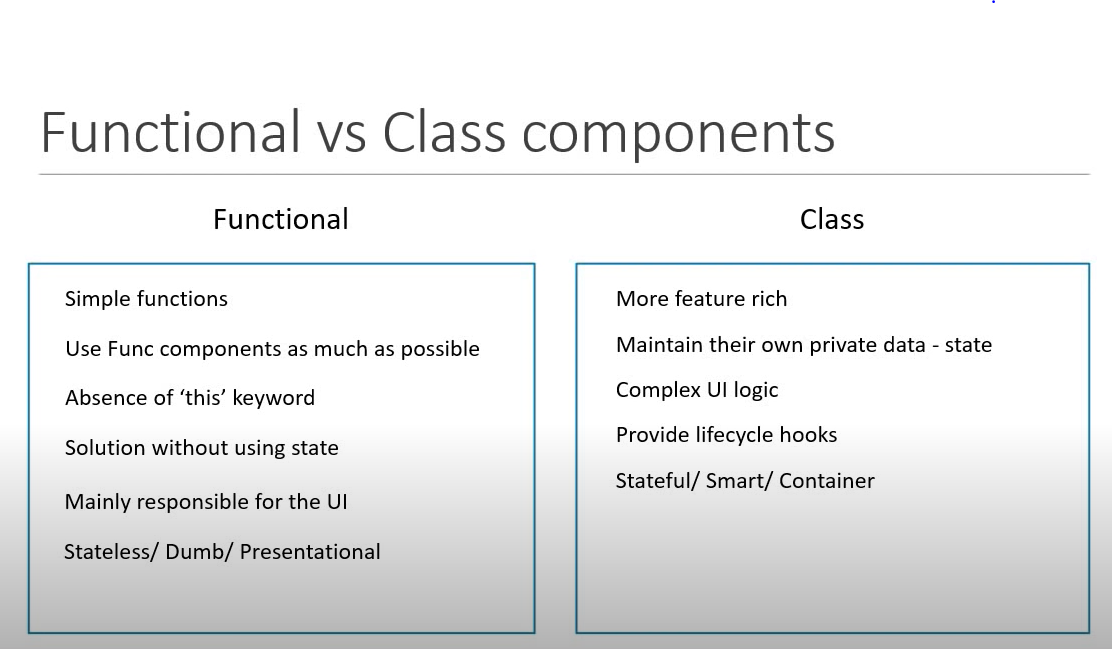
Class Welcome extends React.Component {

render(){

return <h1>Hello, {this.props.name}</h1>

}

}



1. **Functional Component:**

Properties(props) converts into html(jsx)

const Greet = () => <h1>Hello World ! 2</h1>

-----------or --------------

function Greet(){

    return <h1>Hello World!</h1>

}

1. **Class Component:**

ES6 convert from properties(props) into html(jsx)

import React,{ Component } from 'react'

class Welcome extends Component {

    render(){

        return <h1>Class Component</h1>

    }

}

export default Welcome

1. **Hooks**

They let you use state and other react features without writing a class.

import React, { useState } from 'react';

function Counts() {

  // Declare a new state variable, which we'll call "count"

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

  return (

    <div>

      <p>You clicked {count} times</p>

      <button onClick={() => setCount(count + 1)}>

        Click me

      </button>

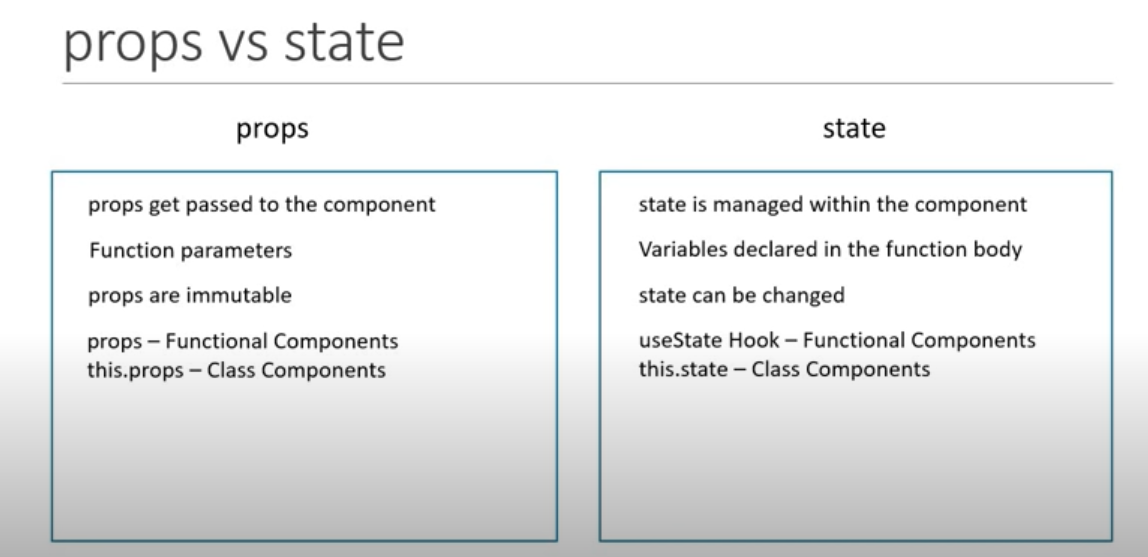
    </div>

  );

}

export default Counts

1. **JSX**
   1. JavaScript XML(JSX) – Extension to JavaScript language syntax
   2. Write XML- like code for elements and components
   3. JSX tags have a tag name, attribute, and children
   4. JSX is not a necessity to write React applications
   5. JSX makes your code simpler and elegant
   6. JSX ultimately transpiles to pure JavaScript which is understood by the browsers



1. **Props in Functional Component**

const Greet = (props) => {

console.log(props)

return (

<div>

<h1>Hello {props.name} a.k.a {props.heroName}</h1>

{props.children}

</div>

)

}

export default Greet

<Greet name="Bruce" heroName="Batman">

<p>This is children props</p>

</Greet>

1. **state in Class Component**

this.state ={

message : 'Welcome visitor'

}

<h1>{this.state.message}</h1>

1. **Event Handling**

function clickHandler(){

console.log('Button Clicked')

}

return (

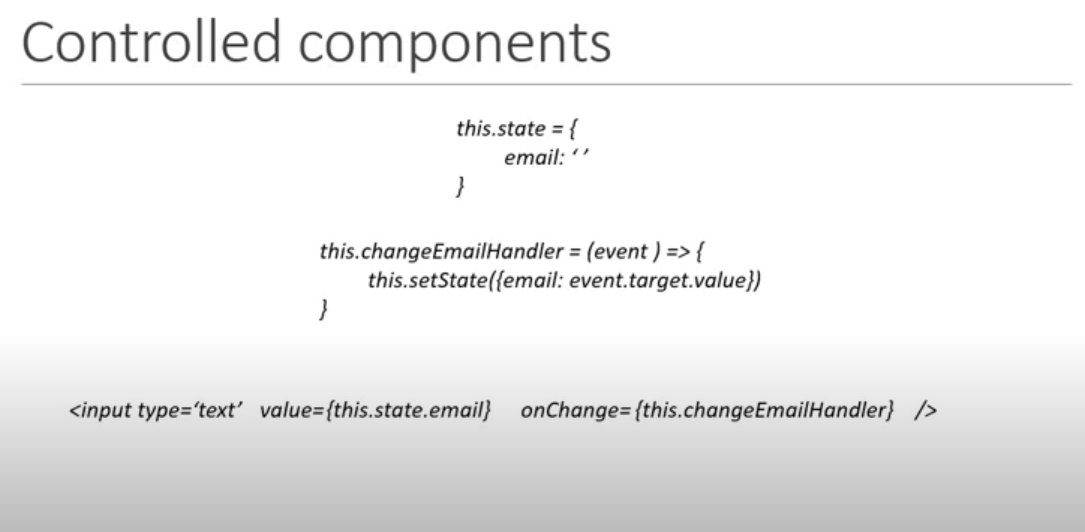
<div>

<button onClick={clickHandler}>Click</button>

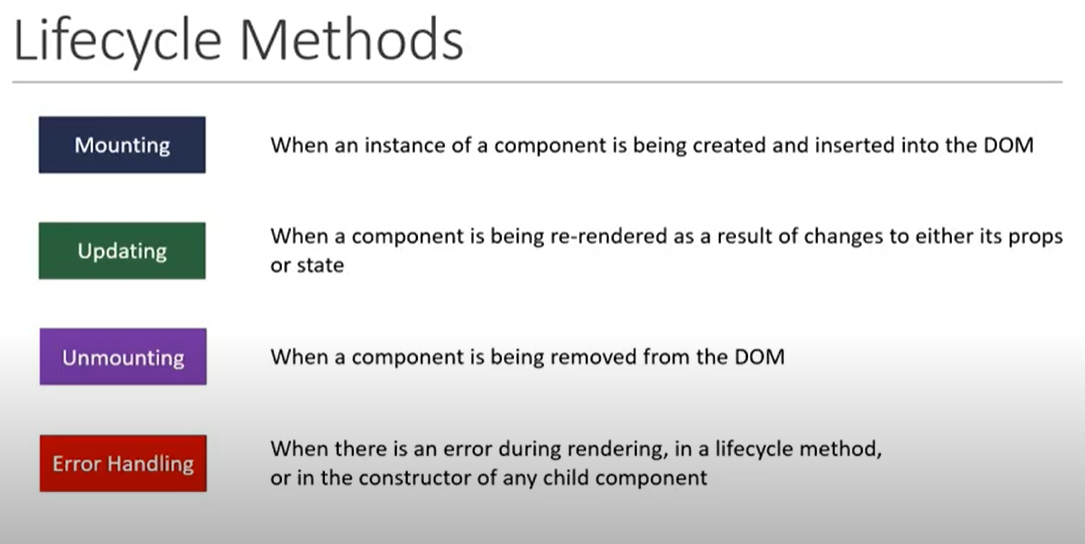
</div>

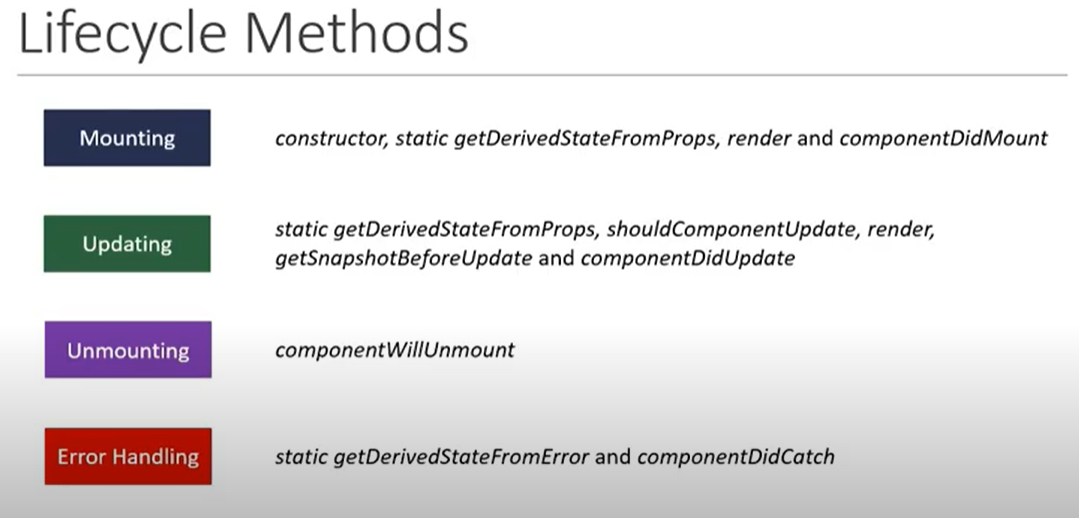
)

1. **Styling React Component**
   1. CSS style sheet
   2. Inline styling
   3. CSS Modules
   4. CSS in JS Libraries
2. **Controlled components**

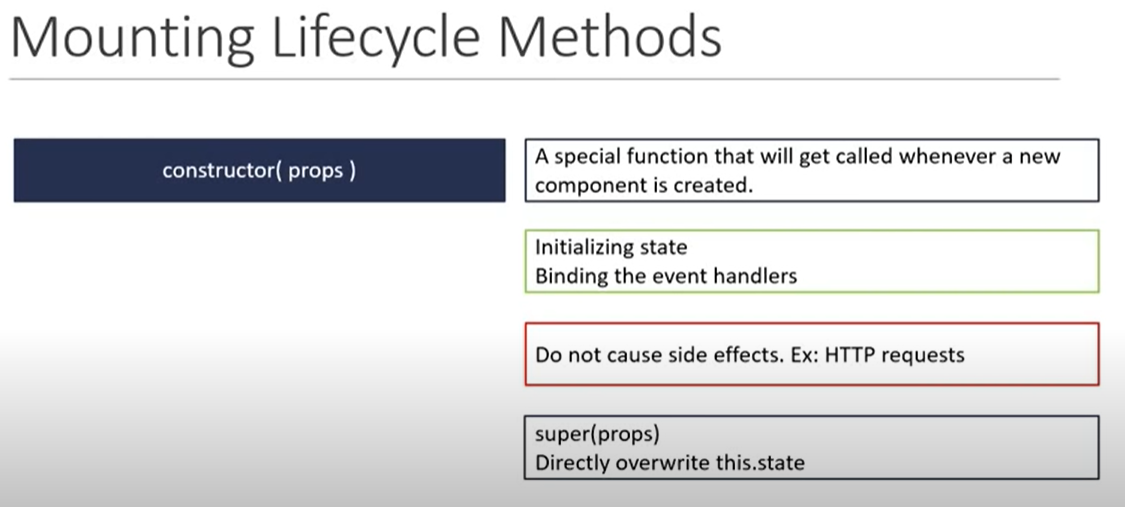


1. **Life cycle methods**





1. **Mounting Lifecycle Methods**



1. **Fragments**
   1. Fragments are used to include element as child directly.

<React.Fragment>

            <h1>Fragment Demo</h1>

            <p>This is paragraph</p>

        </React.Fragment>

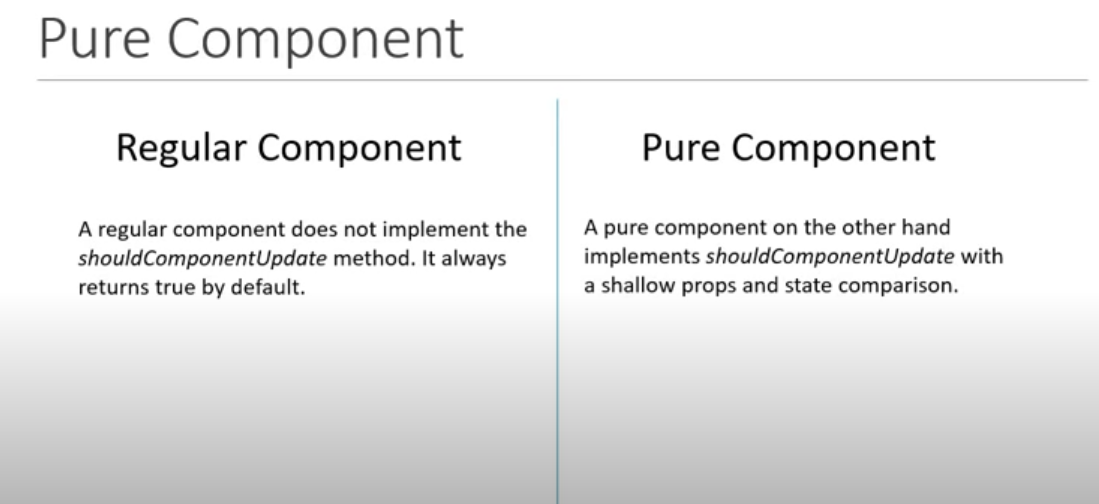
  <>

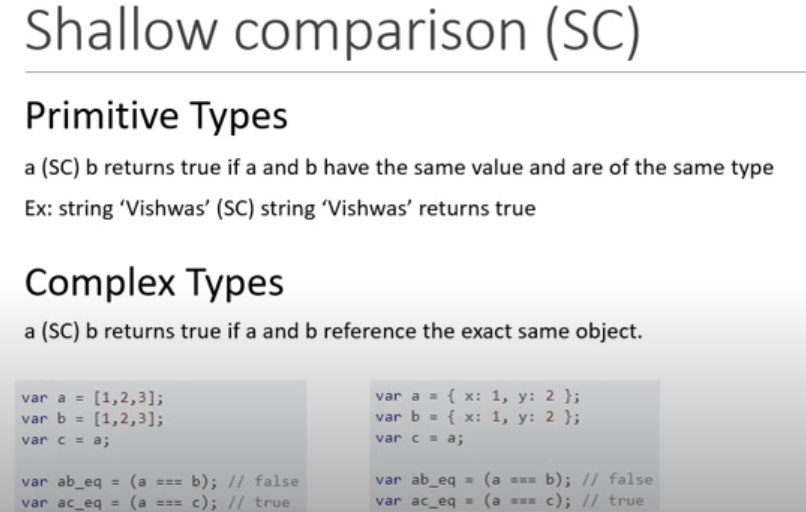
            <td>Name</td>

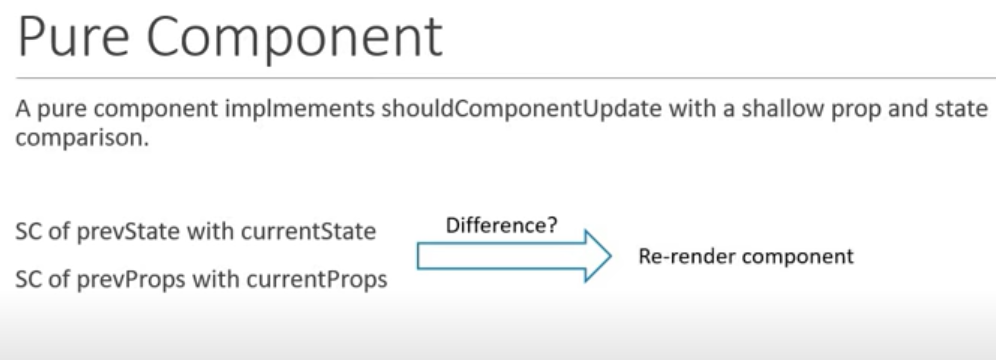
            <td>Ankit</td>

        </>

1. **Pure Components**







1. **Memo Component**
   1. React.memo is a higher order component. It’s similar to React.PureComponent but for function components instead of classes.
   2. If your function component renders the same result given the same props, you can wrap it in a call to React.memo for a performance boost in some cases by memoizing the result. This means that React will skip rendering the component, and reuse the last rendered result.
2. **Refs** 
   1. React.createRef creates a ref that can be attached to React elements via the ref attribute.
   2. React.forwardRef creates a React component that forwards the ref attribute it receives to another component below in the tree.

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.inputRef = React.createRef();

}

render() {

return <input type="text" ref={this.inputRef} />;

}

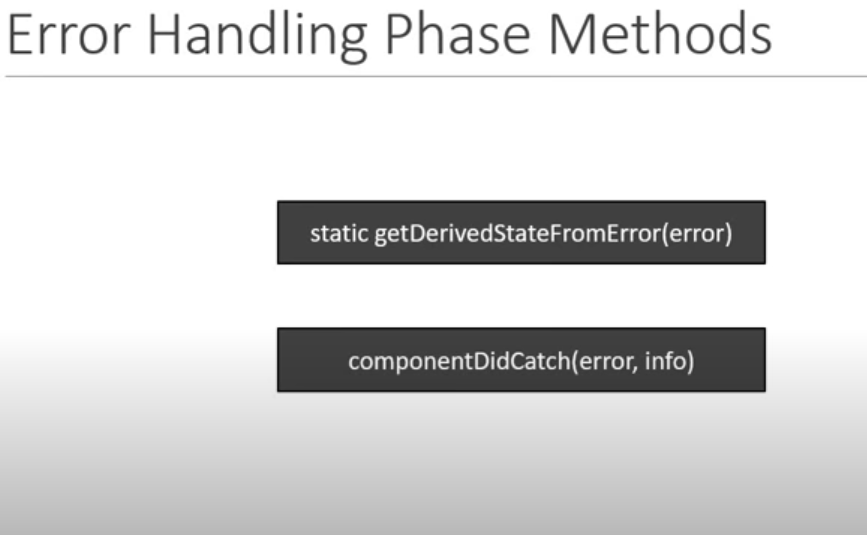
componentDidMount() {

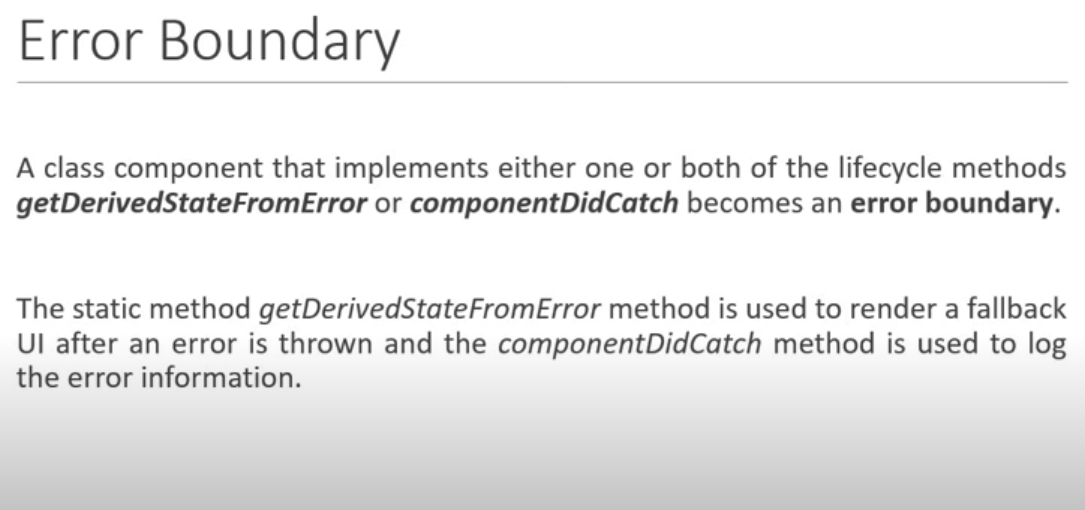
this.inputRef.current.focus();

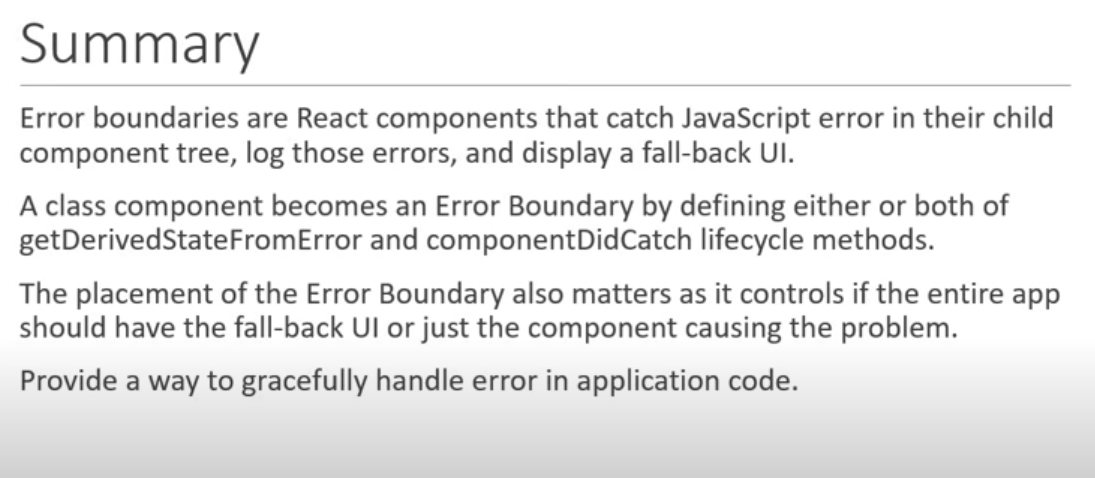
}

}

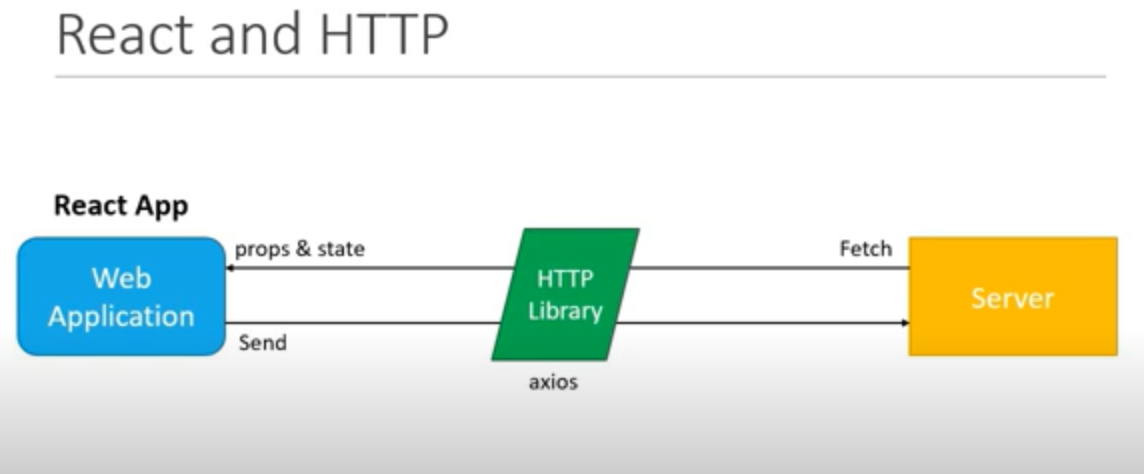
1. **Portals**
   1. Portals provide a first-class way to render children into a DOM node that exists outside the DOM hierarchy of the parent component.
2. **Error Handling**







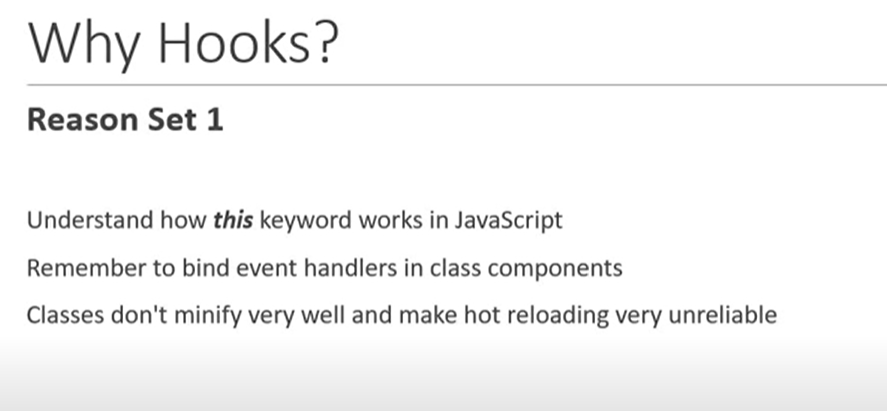
1. **High Order Components**
2. **Render props**
   1. It refers to a technique for sharing code between React components using a prop whose value is a function.
3. **Context**
   1. Context provides a way to pass data through the component tree without having to pass props down manually at every level.
4. **React and HTTP**

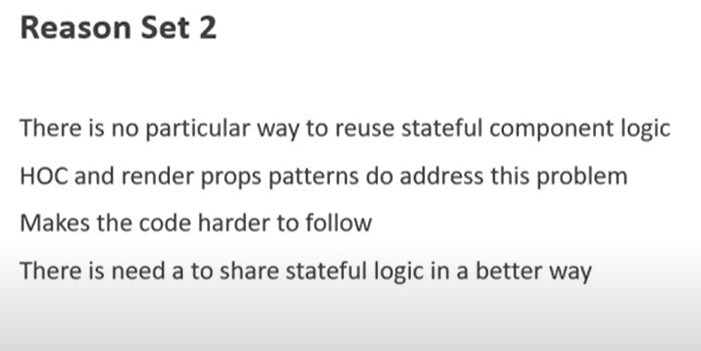


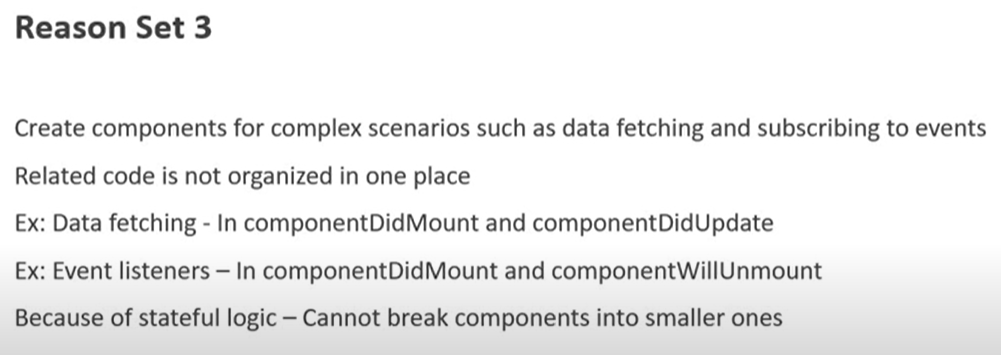
1. **React hooks**
   1. Hooks are a new feature addition in React version 16.8 which allow you to use React features without having to write a class

Ex: State of a component

Hooks don’t work inside classes







1. **useState**

import React, { useState } from 'react';

function Example() {

// Declare a new state variable, which we'll call "count"

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

return (

<div>

<p>You clicked {count} times</p>

<button onClick={() => setCount(count + 1)}>

Click me

</button>

</div>

);

}

1. **useState with previous state**

const [count, setCount] = useState(initialCount)

const incrementFive = () => {

for(let i=0;i<5;i++){

setCount(prevCount => prevCount + 1)

}

}

1. **useState with object**

const [name, setName] = useState({firstName:'',lastName:''})

return (

<form>

<input type="text"

value={name.firstName}

onChange={e => setName({...name,firstName : e.target.value})}/>

<input type="text"

value={name.lastName}

onChange={e => setName({...name,lastName : e.target.value})}/>

<h2>Your first name is - {name.firstName}</h2>

<h2>Your last name is - {name.lastName}</h2>

<h2>{JSON.stringify(name)}</h2>

</form>

)

1. useState with array

const [items, setItems] = useState([])

const addItem = () => {

setItems([...items, {

id: items.length,

value: Math.floor(Math.random() \* 10) + 1

}])

}

return (

<div>

<button onClick={addItem}>Add a number</button>

<ul>

{

items.map(item => (

<li key={item.id}>{item.value}</li>

))

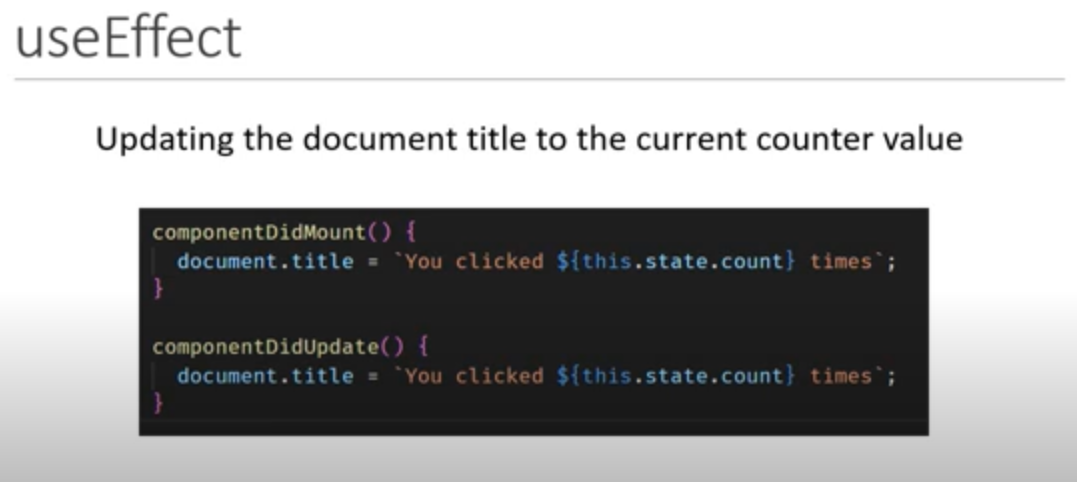
}

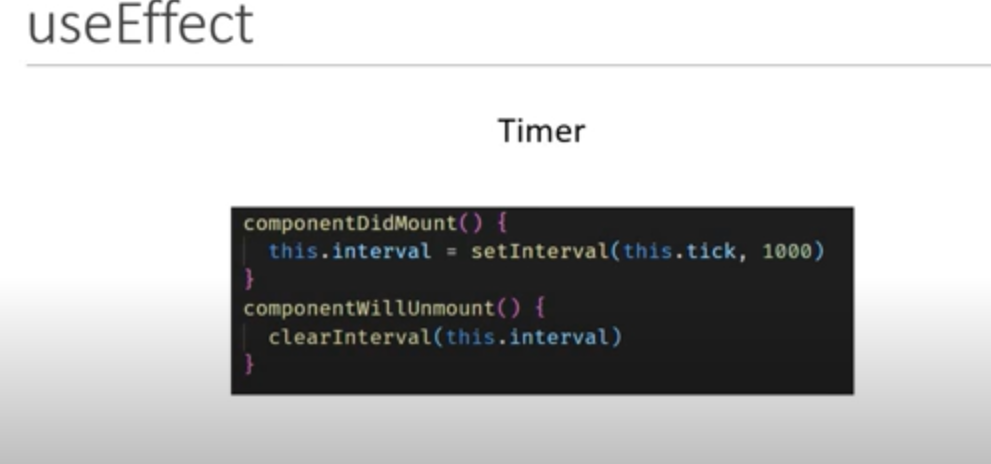
</ul>

</div>

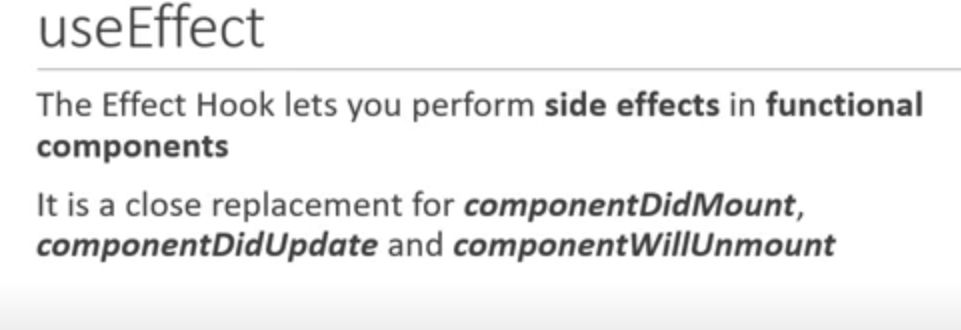
)

1. useEffect hooks









const [count, setCount]= useState(0)

useEffect(() => {

document.title = `Clicked ${count} times`

})

return (

<div>

<button onClick={() => setCount(count +1)}>Click {count} times</button>

</div>

)

1. Run useEffect only once

useEffect(() => {

console.log("useEffect called")

window.addEventListener('mousemove',logMousePosition)

},[])

1. useEffect with cleanup

useEffect(() => {

console.log("useEffect called")

window.addEventListener('mousemove',logMousePosition)

return () => {

console.log('component unmounting code')

window.removeEventListener('mousemove',logMousePosition)

}

},[])

1. Fetching data with useEffect

const[post ,setPost] = useState({})

const [id, setId] = useState(1)

const [idFromButtonClick, setIdFromButtonClick] = useState(1)

useEffect(() => {

axios.get(`https://jsonplaceholder.typicode.com/posts/${idFromButtonClick}`)

.then(res => {

console.log(res)

setPost(res.data)

})

.catch(error => {

console.log(error)

})

},[idFromButtonClick])

1. useContext

export const UserContext = React.createContext()

export const ChannelContext = React.createContext()

function App() {

return (

<div className="App">

<UserContext.Provider value={'Ankit'}>

<ChannelContext.Provider value={'Sharma'}>

<ComponentC/>

</ChannelContext.Provider>

</UserContext.Provider>

</div>

);

}

const user = useContext(UserContext)

const channel = useContext(ChannelContext)

return (

<div>

{user} - {channel}

</div>

)

1. useReducer

