ANECO ACADEMY

REACTJS - HOOKS

TOPICS

What are Hooks?

Why Hooks?

What is useState?

What is useEffect?

What is useContext?

What is useReducer?

What is useCallback?

What is useRef?

HOOKS

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

HOOKS

Understand how ***this*** keyword works in JavaScript

Remember to bind event handlers in class components

Classes don't minify very well and make hot reloading very unreliable

There is no particular way to reuse stateful component logic

HOC and render props patterns do address this problem

Makes the code harder to follow

There is need a to share stateful logic in a better way

Create components for complex scenarios such as data fetching and subscribing to events

Related code is not organized in one place

Ex: Data fetching - In componentDidMount and componentDidUpdate

Ex: Event listeners – In componentDidMount and componentWillUnmount

HOOKS - BENEFITS

The useState hook lets you add state to functional components

In classes, the state is always an object.

With the useState hook, the state doesn't have to be an object.

The useState hook returns an array with 2 elements.

The first element is the current value of the state, and the second element is a state setter function.

New state value depends on the previous state value? You can pass a function to the setter function.

HOOKS - USESTATE

```jsx
import React, {useState} from 'react'

function HookCounter() {

  const [count, setCount] = useState(0)

  return (
    <div>
      <button onClick={() => setCount(count + 1)}>Count {count}</button>
    </div>
  )
}

export default HookCounter
```

The Effect Hook lets you perform **side effects** in **functional components**

It is a close replacement for ***componentDidMount***, ***componentDidUpdate*** and ***componentWillUnmount***

HOOKS - USESTATE

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

const App = () => {
  const [name, setName] = useState("Ceci");

  useEffect(() => {
    document.title = name;
  }, [name]);

  return (
    <input value={name} onChange={event => setName(event.target.value)} />
  );
}
```

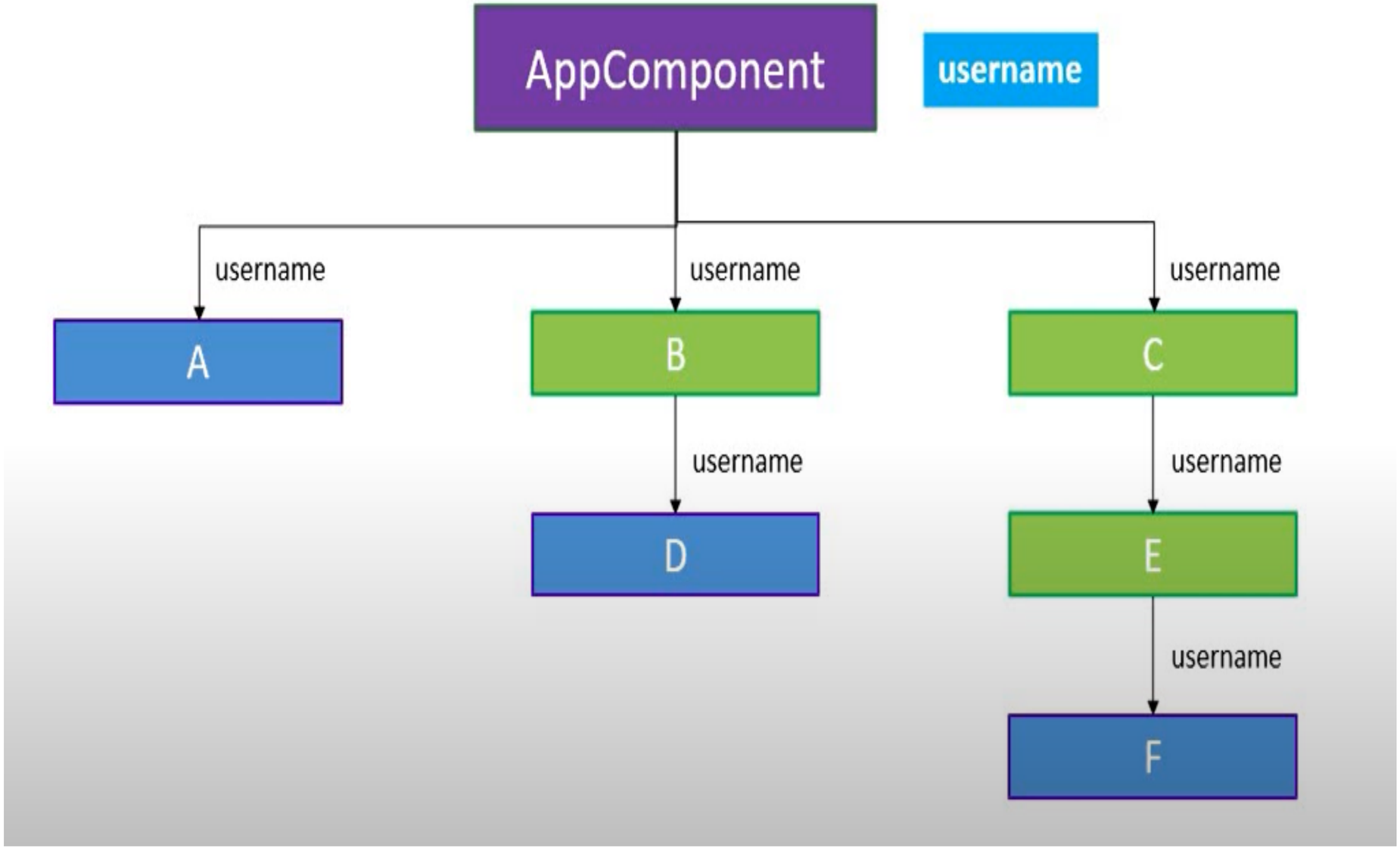
HOOKS - USESTATE

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

  useEffect(() => {
    let timer = setTimeout(() => {
    setCount((count) => count + 1);
  }, 1000);

  return () => clearTimeout(timer)
  }, []);

  return <h1>I've rendered {count} times!</h1>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Timer />);
```

HOOKS - CLASS

```
componentDidMount() {
  document.title = `You clicked ${this.state.count} times`;
  this.interval = setInterval(this.tick, 1000)
}
componentDidUpdate() {
  document.title = `You clicked ${this.state.count} times`;
}
componentWillUnmount() {
  clearInterval(this.interval)
}
```

HOOKS - USECONTEXT



HOOKS - USECONTEXT

Context provides a way to pass data through the component tree without having to pass props down manually at every level.

HOOKS - USECONTEXT

```jsx
import React from 'react'
import './App.css'
import ComponentC from './components/ComponentC'

const UserContext = React.createContext()

function App() {
  return (
    <div className='App'>
      <UserContext.Provider value={'Vishwas'}>
        <ComponentC />
      </UserContext.Provider>
    </div>
  )
}

export default App
```

HOOKS - USECONTEXT

```jsx
import React from 'react'
import {UserContext} from '../App'

function ComponentF() {
  return (
    <div>
      <UserContext.Consumer>
        {
          user => {
            return <div>User context value {user}</div>
          }
        }
      </UserContext.Consumer>
    </div>
  )
}

export default ComponentF
```

HOOKS - USECONTEXT

```jsx
import React, {useContext} from 'react'
import ComponentF from './ComponentF'
import { UserContext, ChannelContext } from '../App'

function ComponentE() {

  const user = useContext(UserContext)
  const channel = useContext(ChannelContext)

  return (
    <div>
      {user} - {channel}
    </div>
  )
}

export default ComponentE
```

HOOKS - USEREDUCER

useReducer is a hook that is used for state management

It is an alternative to useState

What's the difference?

useState is built using useReducer

When to useReducer vs useState?

HOOKS - USEREDUCER

reduce in JavaScript	useReducer in React
array.**reduce**(***reducer***, initialValue)	**useReducer**(***reducer***, initialState)
singleValue = ***reducer***(accumulator, itemValue)	newState = ***reducer***(currentState, action)
reduce method returns a single value	**useReducer** returns a pair of values. [newState, dispatch]

HOOKS - USEREDUCER

```jsx
import React, { useReducer } from 'react'

const initialState = 0
const reducer = (state, action) => {
    switch (action) {
        case 'increment':
            return state + 1
        case 'decrement':
            return state - 1
        case 'reset':
            return initialState
        default:
            return state
    }
}

function CounterOne() {
    const [count, dispatch] = useReducer(reducer, initialState)

    return (
    <div>
      <div>Count = {count}</div>
      <button onClick={() => dispatch('increment')}>Increment</button>
            <button onClick={() => dispatch('decrement')}>Decrement</button>
            <button onClick={() => dispatch('reset')}>Reset</button>
        </div>
    )
}

export default CounterOne
```

HOOKS - USECALLBACK

What?

useCallback is a hook that will return a memoized version of the callback function that only changes if one of the dependencies has changed.

Why?

It is useful when passing callbacks to optimized child components that rely on reference equality to prevent unnecessary renders.

HOOKS - USEMEMO

```
import React from 'react'

function Count({ text, count }) {
    console.log(`Rendering ${text}`)
    return <div>{text} - {count}</div>
}

export default React.memo(Count)
```

HOOKS - USECALLBACK

```jsx
import React, { useState, useCallback } from 'react'
import Count from './Count'
import Button from './Button'
import Title from './Title'

function ParentComponent() {
    const [age, setAge] = useState(25)
    const [salary, setSalary] = useState(50000)

    const incrementAge = useCallback(() => {
        setAge(age + 1)
    }, [age])

    const incrementSalary = useCallback(() => {
        setSalary(salary + 1000)
    }, [salary])

    return (
        <div>
            <Title />
            <Count text="Age" count={age} />
            <Button handleClick={incrementAge}>Increment Age</Button>
            <Count text="Salary" count={salary} />
            <Button handleClick={incrementSalary}>Increment Salary</Button>
        </div>
    )
}

export default ParentComponent
```

HOOKS - USEREF

```jsx
import React, { useRef, useEffect } from 'react'

function FocusInput() {
    const inputRef = useRef(null)
    useEffect(() => {
        inputRef.current.focus()
    }, [])
    return (
        <div>
            <input ref={inputRef} type="text" />
        </div>
    )
}

export default FocusInput
```