Introduction to ReactJS

Aim -

To understand the creation of React App and understand its basic workflow using Components.

Task 1 – Real DOM vs Virtual DOM

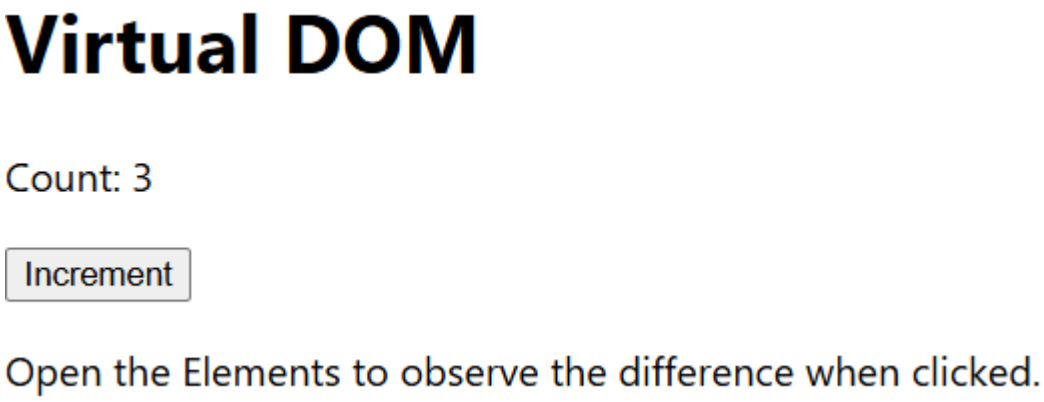
Write a program to understand the difference between a real DOM and Virtual DOM.

Code –

DOMExample.js –

```
import React, { useState } from 'react';
function DOMExample() {
  const [count, setCount] = useState(0);
  const increment= () => {
    setCount(count + 1);
  };
  return (
    <div>
    <h1>Virtual DOM</h1>
    <p>Count: {count}</p>
    <button onClick={increment}>Increment</button>
    <p>Open the Elements to observe the difference when clicked.</p>
  </div>);}
export default DOMExample;
```

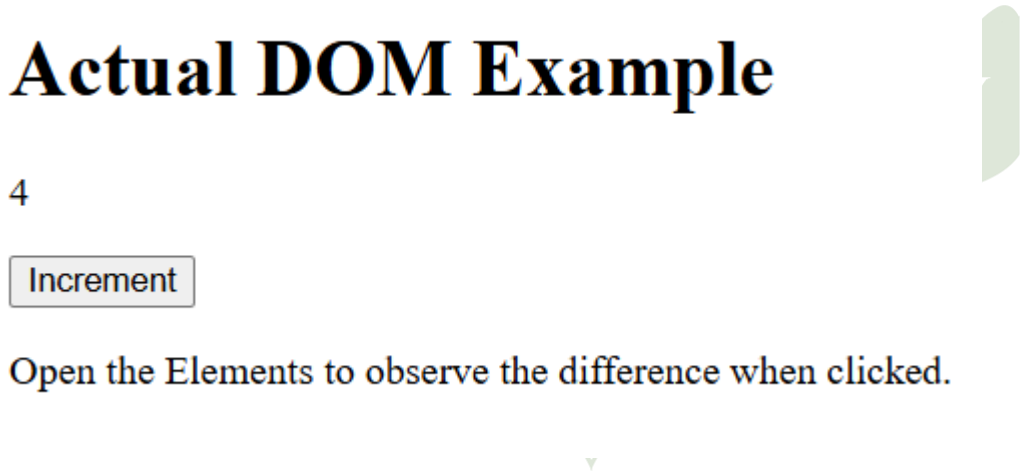
Output –



RealDOM.html -

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Actual DOM</title>
</head>
<body>
    <h1>Actual DOM Example</h1>
    <p id="count">0</p>
    <button onclick="increment()">Increment</button>
    <p>Open the Elements to observe the difference when clicked.</p>
    <script>
        let count = 0;
        function increment() {
            count++;
            document.getElementById("count").innerHTML = count;
        }
    </script>
</body>
</html>
```

Output –



Task 2 – Component Calling Component

Write a program to demonstrate calling a component from another component.

Code –

Inner.js -

```
function Inner() {
   return (
     <div>
       <p>This is the Inner component which is called in Outer Component</p>
     </div>
   )
  }

export default Inner;
```

Outer.js –

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

function Outer() {
 return (
   <div>
     <h1>Outer component</h1>
     <h3>Now a component will be called</h3>
     <Inner />
     <h3>Back in the outer component</h3>
   </div>
 )
}
export default Outer;
```

Output –

Outer component

Now a component will be called

This is the Inner component which is called in Outer Component

Back in the outer component

Task 3 – React Class Component

Write a program to demonstrate React Class Components.

Code –

ClassComponent.js –

```
import React, { Component } from 'react'
export default class ClassComponent extends Component {
 render() {
   return (
     <div>
       <h1>Class Component</h1>
       <p>Class Component is a type of component in React that is defined using a class.</p>
     </div>
   )
 }
}
```

Output –

Class Component

Class Component is a type of component in React that is defined using a class.

Task 4 – React Props

Write a program to demonstrate React props.

Code –

Props.js –

```
import React from 'react'

function Props(props) {
 return (
   <div>
     <h1>Player details</h1>
     <p>Name: {props.name}</p>
     <p>Nationality: {props.nation}</p>
   </div>
 )
}

export default Props
```

Output –

Player details

Name: Virat Kohli

Nationality: India

Task 4 – React Lifecycle

Write code to illustrate the lifecycle of React JS.

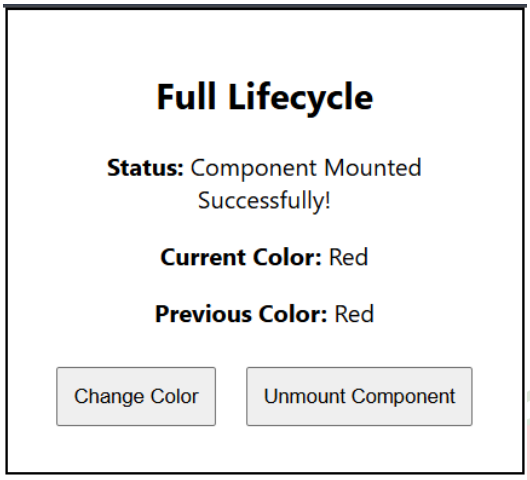
Code –

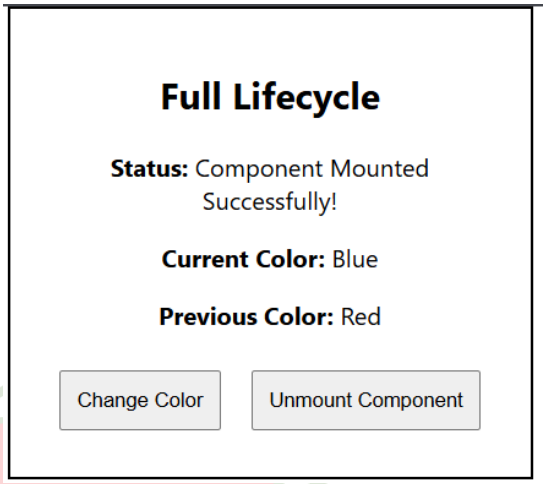
Lifescycle.js –

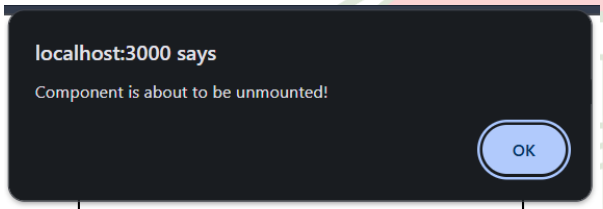
```
import React, { Component } from "react";
class FullLifecycleDemo extends Component {
  constructor(props) {
    super(props);
    this.state = { color: "Red", prevColor: "", status: "Mounting Component..." };
    console.log("Mounting: Constructor");
  }
  static getDerivedStateFromProps(props, state) {
    console.log("Mounting/Updating: getDerivedStateFromProps");
    return { color: props.favColor || state.color };
  }
  componentDidMount() {
    console.log("Mounting: componentDidMount");
    this.setState({ status: "Component Mounted Successfully!" });
  }
  shouldComponentUpdate(nextProps, nextState) {
    console.log("Updating: shouldComponentUpdate");
    return true;
  }
  getSnapshotBeforeUpdate(prevProps, prevState) {
    console.log("Updating: getSnapshotBeforeUpdate");
    return prevState.color;
  }
  componentDidUpdate(prevProps, prevState, snapshot) {
    if (snapshot && snapshot !== this.state.prevColor) {
      console.log(`Updating: componentDidUpdate  (Prev Color: ${snapshot})`);
      this.setState({ prevColor: snapshot });
    }}
```

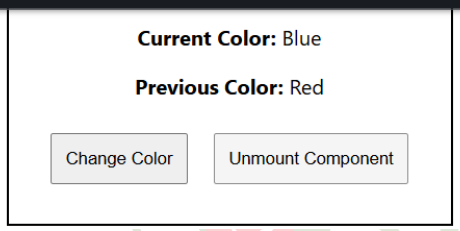
```
  componentWillUnmount() {
    console.log("Unmounting: componentWillUnmount");
    alert("Component is about to be unmounted!");
  }
  changeColor = () => {
    const newColor = this.state.color === "Red" ? "Blue" : "Red";
    this.setState({ color: newColor });
  };
  render() {
    console.log(" 🔹 Updating: render()");
    return (
      <div style={styles.container}>
        <h2>Full Lifecycle</h2>
        <p><b>Status:</b> {this.state.status}</p>
        <p><b>Current Color:</b> {this.state.color}</p>
        {this.state.prevColor && <p><b>Previous Color:</b> {this.state.prevColor}</p>}
        <button onClick={this.changeColor} style={styles.button}>
          Change Color
        </button>
        <button onClick={this.props.unmount} style={styles.button}>
          Unmount Component
        </button>
      </div>
    );
  }}
class Parent extends Component {
  state = { showChild: true };
  toggleChild = () => this.setState({ showChild: !this.state.showChild });
  render() {
    return (
      <div>
        {this.state.showChild ? (
          <FullLifecycleDemo unmount={this.toggleChild} />
        ) : (
          <h2 style={{ textAlign: "center" }}>Component Unmounted!</h2>
        )}
      </div>
    );
  }
}
const styles = {
  container: { padding: "20px", textAlign: "center", border: "2px solid black", width: "300px",
margin: "auto" },button: { margin: "10px", padding: "10px" }};
export default Parent;
```

Output –

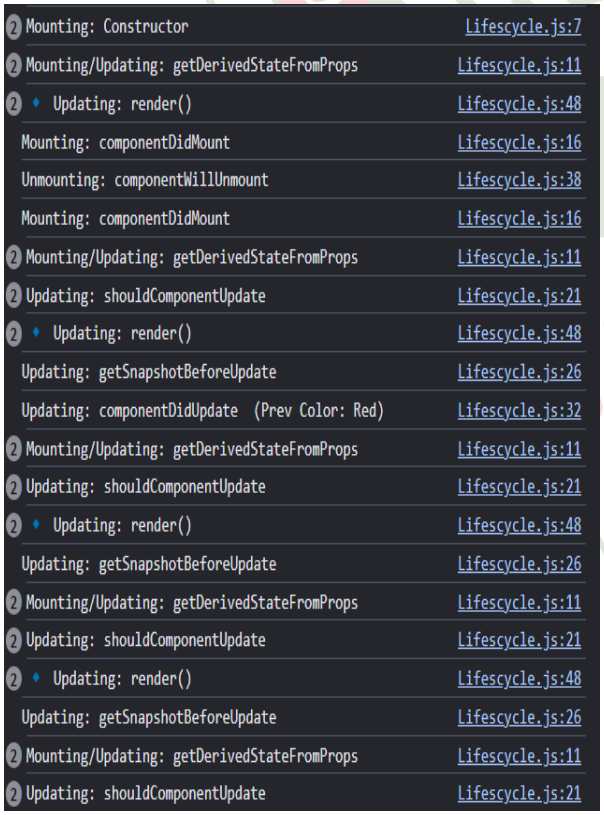

Full Lifecycle
Status: Component Mounted Successfully!
Current Color: Red
Previous Color: Red
Change Color
Unmount Component

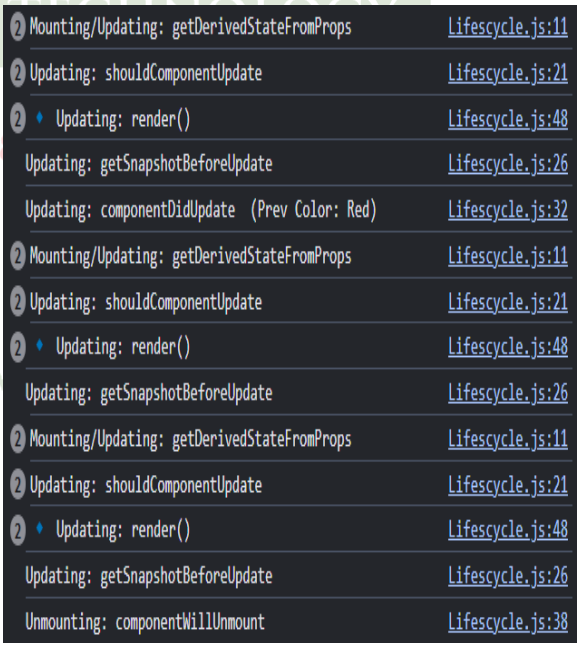

Full Lifecycle
Status: Component Mounted Successfully!
Current Color: Blue
Previous Color: Red
Change Color
Unmount Component


localhost:3000 says
Component is about to be unmounted!
OK


Current Color: Blue
Previous Color: Red
Change Color
Unmount Component


Component Unmounted!


Mounting: Constructor Lifecycle.js:7
Mounting/Updating: getDerivedStateFromProps Lifecycle.js:11
Updating: render() Lifecycle.js:48
Mounting: componentDidMount Lifecycle.js:16
Unmounting: componentWillUnmount Lifecycle.js:38
Mounting: componentDidMount Lifecycle.js:16
Mounting/Updating: getDerivedStateFromProps Lifecycle.js:11
Updating: shouldComponentUpdate Lifecycle.js:21
Updating: render() Lifecycle.js:48
Updating: getSnapshotBeforeUpdate Lifecycle.js:26
Updating: componentDidUpdate (Prev Color: Red) Lifecycle.js:32
Mounting/Updating: getDerivedStateFromProps Lifecycle.js:11
Updating: shouldComponentUpdate Lifecycle.js:21
Updating: render() Lifecycle.js:48
Updating: getSnapshotBeforeUpdate Lifecycle.js:26
Mounting/Updating: getDerivedStateFromProps Lifecycle.js:11
Updating: shouldComponentUpdate Lifecycle.js:21
Updating: render() Lifecycle.js:48
Updating: getSnapshotBeforeUpdate Lifecycle.js:26
Mounting/Updating: getDerivedStateFromProps Lifecycle.js:11
Updating: shouldComponentUpdate Lifecycle.js:21


Mounting/Updating: getDerivedStateFromProps Lifecycle.js:11
Updating: shouldComponentUpdate Lifecycle.js:21
Updating: render() Lifecycle.js:48
Updating: getSnapshotBeforeUpdate Lifecycle.js:26
Updating: componentDidUpdate (Prev Color: Red) Lifecycle.js:32
Mounting/Updating: getDerivedStateFromProps Lifecycle.js:11
Updating: shouldComponentUpdate Lifecycle.js:21
Updating: render() Lifecycle.js:48
Updating: getSnapshotBeforeUpdate Lifecycle.js:26
Mounting/Updating: getDerivedStateFromProps Lifecycle.js:11
Updating: shouldComponentUpdate Lifecycle.js:21
Updating: render() Lifecycle.js:48
Updating: getSnapshotBeforeUpdate Lifecycle.js:26
Unmounting: componentWillUnmount Lifecycle.js:38