EXPERIMENT - 22

Explain React Hooks with an Example

React Hooks are special functions that let you "hook into" React features, such as state and lifecycle methods, in functional components. Before Hooks were introduced, you could only use state and other React features in class components. With Hooks, you can use state and other features in functional components, making them more powerful and versatile.

1] UseState Hook:

The useState hook is one of the most commonly used hooks in React. It allows you to add state to functional components, enabling them to hold and manage data that can change over time, such as user input, dynamic content, or UI interactions.

Program: App2.js import React

,{useState}from 'react' export default

function App2() {

const[color,setcolor]=useState("Red");

return (

<h1>My favorite color={color}</h1>

<button onClick={()=>setcolor("Blue")}>Change Color</button>

);

} **Index.js** import React from 'react'; import ReactDOM from 'react-

dom/client'; import App2 from './App2'; const root =

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

root.render(<App2/>);

OUTPUT:

2]UseEffect Hook:

useEffect is a Hook in React that allows you to perform side effects in your functional components. Side effects can include things like fetching data from an API, subscribing to a service, or manually updating the DOM. The useEffect hook enables you to run this code after the component has rendered and manage it throughout the component's lifecycle.

Program: App3.js import React, { useState,

useEffect } from 'react'; export default function

App3() { const [count, setCount] = useState(0);

useEffect(() => { const timer = setTimeout(() =>

{ setCount((prevCount) => prevCount + 1);

}, 1000); return () =>

clearTimeout(timer);

}, [count]); return

(

<h1>Counter value = {count}</h1>

<h2>I have rendered {count} times</h2>

);

} **Index.js** import React from 'react';

import ReactDOM from 'react-dom/client';

import App3 from './App3';

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

root.render(<App3/>);

OUTPUT:

Counter value = 20

I have rendered 20 times

3]UseContext Hook:

The useContext hook in React is used to access the value of a context within a functional component. Context provides a way to pass data through the component tree without having to

pass props down manually at every level. It's particularly useful when you have global data or functionality that needs to be accessed by many components at different levels of the tree.

Program: App4.js import React, { useContext, createContext,

useState } from 'react'; const UserContext = createContext();

export default function Component1() { const [user, setUser] =

useState("Rakshitha S R"); return (

<>

<UserContext.Provider value={user}>

<h1>Hello {user}</h1>

<h2>Component 1</h2>

<Component2 />

</UserContext.Provider>

</>

);

}

function Component2() { const user =

useContext(UserContext); return (

<h1>Hello {user}</h1>

<h2>Component 2</h2>

<Component3 />

);

} function Component3() { const user

= useContext(UserContext); return (

<h1>Hello {user}</h1>

<h2>Component 3</h2>

);

}

Index.js

import React from 'react'; import ReactDOM from 'react-dom/client';

import Component1 from './App'; const root =

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

root.render (<Component1/>);

OUTPUT:

Hello Nischal k

Component 1

Hello Nischal k

Component 2

Hello Nischal k

Component 3

4]UseRef Hook:

The useRef hook in React is used to persist values across renders without causing a re-render when the value changes. It can also be used to directly access and manipulate DOM elements.

Program: App5.js

import React, { useRef } from

'react'; function Textfocus() { const

inputRef = useRef(null); const

handleFocus = () => {

inputRef.current.focus();

}; return

(

<h1>useRef Example</h1>

<input ref={inputRef} type="text" />

<button onClick={handleFocus}>click to focus on Input</button>

 ); } export default Textfocus; **index.js** import React from 'react'; import ReactDOM from 'react-dom/client'; import Textfocus from './App5'; const root = ReactDOM.createRoot(document.getElementById('root')); root.render(<Textfocus/>);

OUTPUT:

EXPERIMENT - 23

React Router Examples

Index.js import {createRoot} from 'react-dom/client'; import { BrowserRouter, Routes,Route,Link } from 'react-router-dom'; function Home()
{ return <h1> Home page</h1>;
} function About()
{ return <h1> About page</h1>;
} function Contact()
{ return<h1> Contact page</h1>;
} function App()
{
return (
<BrowserRouter>

```
      <nav>
        <Link to='/'>Home</Link>|{""}
        <Link to='/About'>ABOUT</Link>|{""}
        <Link to='/Contact'>CONTACT</Link>
        </nav>
        <Routes>
          <Route path="/" element={<Home/>}/>
          <Route path="/About" element={<About/>}/>
          <Route path="/Contact" element={<Contact/>}/>
        </Routes>
        </BrowserRouter>
    );
} createRoot(document.getElementById("root")).render(<App/>);
```

OUTPUT:

EXPERIMENT-24

In react, Create a form that should contain Present Address And Permanent Address in two Text Area Box. When the User Enters any text in any of the one text area box itshould dynamically reflect in another textarea box.implement using react.

Index.js import React from 'react'; import ReactDOM from 'react-

dom/client'; import ParentComponent from './App'; const root =

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

root.render(<ParentComponent/>);

App.js:

```
import ReactDom from 'react-dom/client';
import React, { Component } from 'react';
class ParentComponent extends Component {
constructor(props){ super(props);
this.state={name:''};
}
nameChanged=(val)=>{
this.setState({name:val});
} render()
{ return (
<div>
<PresentAddress name={this.state.name} onNameChange={this.nameChanged}/>
<PermanentAddress name={this.state.name} onNameChange={this.nameChanged}/> </div>
)
}
} class PresentAddress extends Component
{ handleChange=(e)=>{
this.props.onNameChange(e.target.value);
} render()
{ return (
<div>
<form>
<label>
Present Address:<textarea rows='3' cols='20' value={this.props.name}
onChange={this.handleChange}/>
</label>
</form>
</div>
```

```
)
}
} class PermanentAddress extends Component
{ handleChange=(e)=>{
this.props.onNameChange(e.target.value);
} render()
{ return (
<div>
<form>
<label>
Permanent Address:<textarea rows='3' cols='20'
value={this.props.name} onChange={this.handleChange}/>
</label>
</form>
</div>
)
} } export default
ParentComponent;
```

OUTPUT: