REACT HOOKS ROLE IN DEVELOPMENT

[1]SUDESH BHAT
Co-author: - Prof. Alamma B.H

[1]PG SCHOLAR, DEPT of MCA, DSCE
CA - ASST. PROF, DEPT of MCA, DSCE

ABSTRACT-

React hooks introduced in 2018 as a way to use state and side effects in react function components. Functional components are called as functional stateless components. They allow us to use state with react. React hooks are basically functions that able hook into react state and also life cycle feature from function. It solve wide verity of problems like maintaining thousands of components. In this paper will describe about some practical ways of overcome from the problem of using state.

Keywords:- ReactJs, componentDidMount, componentDidUpdate, componentWillUnmount, useEffect, useState.

INTRODUCTION

React is invented to state management and side effect. It make more efforts less without use this.setState in a class. React hooks allow us to write react application only with function component. React hooks don't work inside a class because they let us use react without class. By doing this we can totally avoid the life cycle method like component Did Mount, component Did Update, component Will Unmount. Insted of this we use react hooks such as useEffect.

LITERATURE SURVEY

Hooks are the new feature introduced in the React 16.8 version. It allows you to use state and other React features without writing a class. useState will be used to add states in functional stateless components. useEffects will constantly considered when the component is gathered and updated.

METHODOLOGY

For the analysis of hooks I'm going to use ReactJs. React is a JavaScript library can be used as a base in the development of single-page or mobile applications it has the powerful hooks called useState, useEffect. You know that the states cannot be used in functions, but with hooks, we can use states. Another reason is the handle side effect in react component. It means, now you can use newly introduced state such as useEffect

useState

useState is one of basic react hook. With which we can add states in functional stateless components. It takes a parameter which is states initial value and return two properties in array one is state and other one is method which used to update it.

The useState hook accepts an initial state as argument and returns, by using array destructing. Two variables that we want to name them. Where the first variable is actual state, the second variable is a function to update the state by providing a new state.

After we call useState

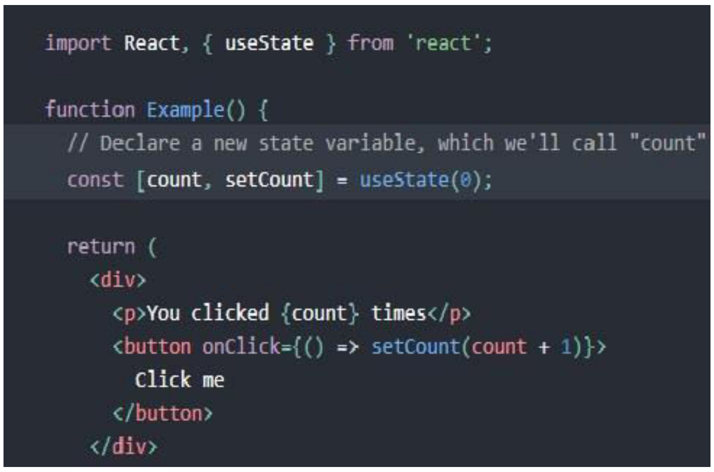
It declares a state variable. Our variable is called count however we could call it whatever else, like to banana. This is an approach to protect a few values between the function calls — useState is another approach to utilize precisely the same abilities that this.state gives in a class. Ordinarily, factors disappear when the capacity exits yet state variable are safeguarded by React

Arguments need to useState

The main contention to the useState() Hook is the initial state. Not at all like with classes, the state doesn't need to be an object. We can keep a number or a string if that is all we need. In our example, we simply need a number for how often the client clicked, so pass 0 as initial state for our variable. (On the off chance that we needed to store two unique values in state, we would call useState() twice.

Output of useState

It return a pair of values: the present state and a function that refreshes it. This is the reason we write const [count, setCount] = useState(). This is like this.state.count and this.setState in a class, with the exception of you get them in a pair. In case you're curious about the linguistic structure we utilized, we'll return to it at the base of this page



```
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>
```

[fig.1] Demonstration of useState.

useEffect

The useEffects technique is constantly considered when the component is gathered and updated. With it we can replace the component Did Mount, component Did Update and component Will Unmount lifecycles. It executes the function inside it and has a discretionary second parameter, which is an array of properties to be seen inside the scope of the stateless part. At whatever point any of them are updated, the function is executed once more.

useEffect Purpose

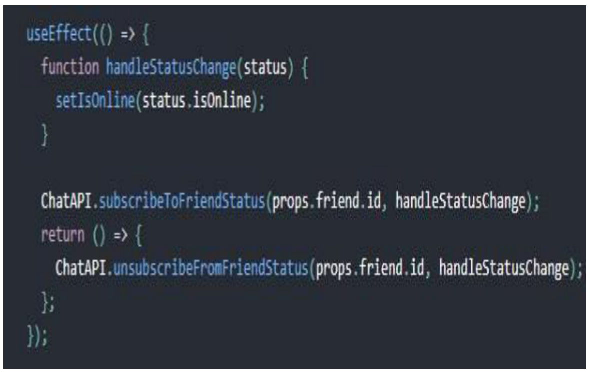
By utilizing this Hook, you disclose to React that your part needs to accomplish something after the render. React will recall the function we passed (we'll allude to it as our effect), and call it later in the performing the DOM updates. Right now, set the document title, yet we could likewise perform data fetching or call some other basic API

Call useEffect inside a component

Setting useEffect inside the component lets us get to the count state variable (or any props) directly from the effect. We needn't bother with an other API to read it — it's as of now in the function scope. Hook grasp JavaScript terminations and abstain from presenting React-specific APIs where JavaScript as of now gives an answer.

useEffect run after every render

It runs both the main render and after each update. Instead of intuition regarding "mounting" and "refreshing", you may think that its simpler to believe that effect happen "after render". Respond ensures the DOM has been refreshed when it runs the effects.



```
useEffect(() => {
  function handleStatusChange(status) {
    setIsOnline(status.isOnline);
  }

  ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
  return () => {
    ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
  };
});
```

[fig.2] Demonstration of useEffect

Rules for using Hooks

Must Call Hooks at the Top Level.

Try not to call Hooks inside loops, conditions, or nested functions.

Must Call Hooks from React Functions.

Try not to call Hooks from ordinary JavaScript function.

CONCLUSION

However Hooks solve a wide variety of seemingly unconnected issues in React. With Hooks, you can extract stateful logic from a component so it very well may be tried freely and reused. Hooks permit you to reuse stateful logic without changing our component hierarchy. This makes it simple to share Hooks among numerous components or with the network. Hooks let us split one component into littler functions dependent on what pieces are connected (for example, setting up a membership or fetching information), instead of forcing a split dependent on lifecycle techniques.

REFERENCE

•https://reactjs.org/doc

•Darrel Greenhill, jack Francik, jay kirithika "UX Design in web Application", IEEE 2016

•"Web Workers API," MDN Docs, developer.mozilla.com