**Gaza Gateway**

**React Training (Report Tenth)**

**React Hooks**

Author

Yazan Abu Al Kass

Direction

Mohammed Shamia

8- May-2022

* **About**

In this report, I will write about React Hooks in my own way as I understand it.

* **Introduction**

Hooks were added to React in version 16.8.

Hooks allow function components to have access to state and other React features. Because of this, class components are generally no longer needed.

**What is a Hook?**

Hooks allow us to "hook" into React features such as state and lifecycle methods.

* **Subject**

1. **React useState Hook:**

useState generally refers to data or properties that need to be tracking in an application.

To use the useState Hook, we first need to import it into our component.

import {useState} from "react";

useState accepts an initial state and returns two values:

- The current state.

- A function that updates the state.

const [state, setState] = useState("red");

The useState Hook can be used to keep track of strings, numbers, Booleans, arrays, objects, and any combination of these.

1. **React useEffect Hook:**

The useEffect Hook allows you to perform side effects in your components

Some examples of side effects are: fetching data, directly updating the DOM, and timers.

useEffect accepts two arguments. The second argument is optional.

**Ex:** useEffect(<function>, <dependency>)

There are several ways to control when side effects run.

We should always include the second parameter which accepts an array. We can optionally pass dependencies to useEffect in this array.

**1. No dependency passed:**

When used Runs on every render.

**Example**

useEffect(() => {

});

**2. An empty array:**

When used Runs only on the first render.

**Example**

useEffect(() => {

}, []);

**3. Props or state values:**

When used Runs on the first render and any time any dependency value changes.

**Example**

useEffect(() => {

}, [prop, state]);

**4. Effect Cleanup:**

Some effects require cleanup to reduce memory leaks.

Timeouts, subscriptions, event listeners, and other effects that are no longer needed should be disposed.

We do this by including a return function at the end of the useEffect Hook.

**Example**

useEffect(() => {

let timer = setTimeout(() => {

setCount((count) => count + 1);

}, 1000);

return () => clearTimeout(timer)

}, []);

1. **React useRef Hook:**

The useRef Hook allows you to persist values between renders.

It can be used to store a mutable value that does not cause a re-render when updated.

It can be used to access a DOM element directly.

In general, we want to let React handle all DOM manipulation.

But there are some instances where useRef can be used without causing issues.

In React, we can add a ref attribute to an element to access it directly in the DOM.

**Example**

const inputElement = useRef();

return (

<>

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

</>

);

1. **React useId Hook:**

useId is a hook for generating unique IDs that are stable across the server and client while avoiding hydration mismatches.

**Example**

function NameFields() {

const id = useId();

return (

<div>

<label htmlFor={id + '-firstName'}>First Name</label>

<div>

<input id={id + '-firstName'} type="text" />

</div>

</div>

);

}

1. **React useCallback Hook:**

This allows us to isolate resource-intensive functions so that they will not automatically run on every render.

The useCallback Hook only runs when one of its dependencies is updated this can improve performance.

**Example**

const addTodo = useCallback(() => {

setTodos((t) => [...t, "New Todo"]);

}, [todos]);

1. **React useMemo Hook:**

The useMemo Hook only runs when one of its dependency's updates.

This can improve performance.

The useCallback and useMemo Hooks are similar. The main difference is that useMemo returns a memoized value and useCallback returns a memoized function. You can learn more about useMemo in the useMemo.

**Example**

const calculation = useMemo(() => expensiveCalculation(count), [count]);

const expensiveCalculation = (num) => {

console.log("Calculating...");

for (let i = 0; i < 1000000000; i++) {

num += 1;

}

return num;

};

* **Conclusions**

After researching and reading about the topic, I learned the difference between class components and functional components and Hooks, their states, and how to use them to manage components in a function.