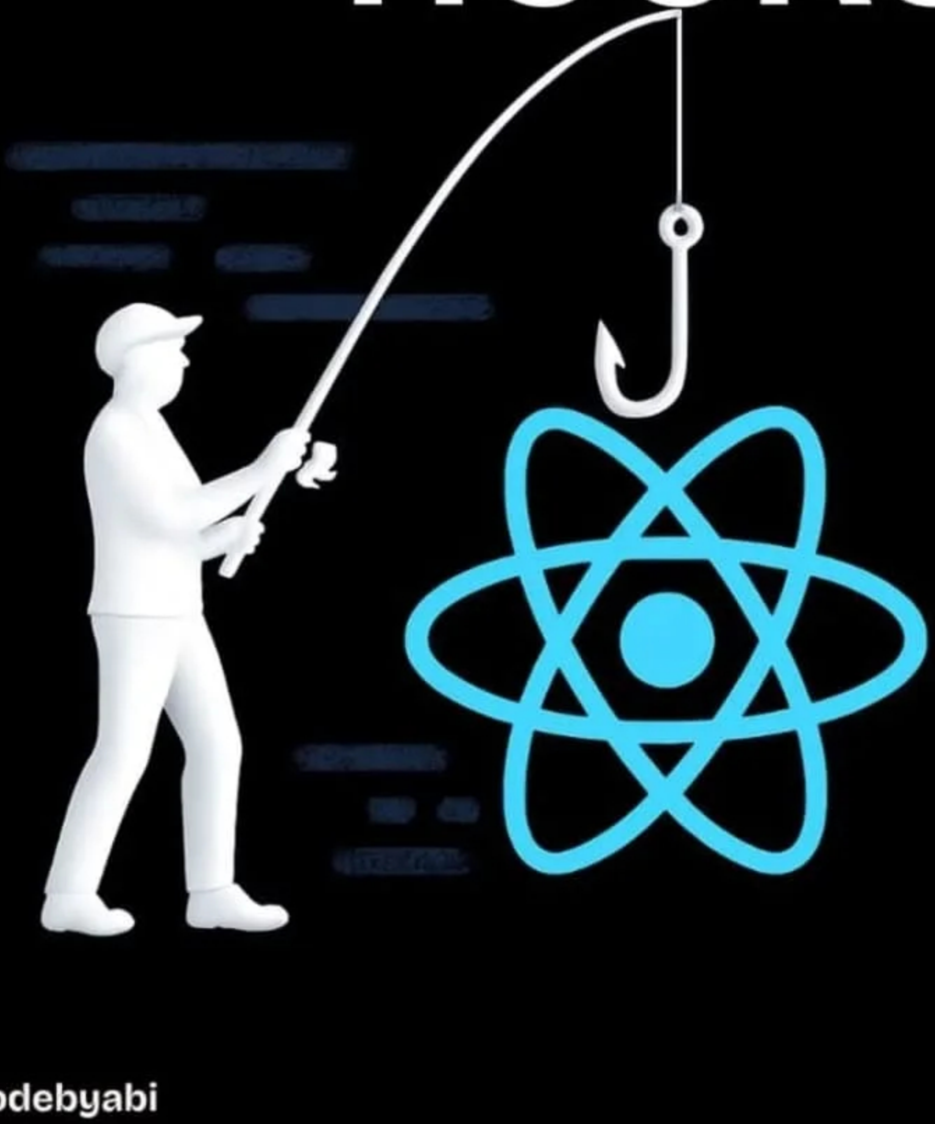
save the post

REACT HOOKS



@codebyabi

1. useState

Creates an "state variable" which updates the component on change

useState is the most common hook in React. The hook needs 3 inputs to create a state.

- Current state (count): The name of the variable, which is equal to the current state.
- Function (setCount): A function which gets called to change the state.
- Initial value (0): The default value when page initialised

```
import { useState } from "react";

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

 return (
 <div>
        <h1>{count}</h1>
       <button onClick={() => setCount(count + 1)}>
             Add 1 to count
       </button>
</div>
);
};
```

2. useEffect

A function which gets called every time the view gets mounted

useEffect is the most often used hook in React. The hook get 2 things

- **[]: Runs only once on mount.**
- **[count]: Runs on mount and whenever count changes.**

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

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

 useEffect(() => {
 console.log("View Mounted");
 }, []);

 useEffect(() => {
 console.log("View Mounted or Count updated");
 }, [count]);

 // .. //
};
```

3. useRef

It stores a value that can change without causing re-renders.

The most common use of useRef is to store a DOM element.
For example, in an input field, you can use it to access the current value directly.

```
import { useRef } from "react";

const App = () => {
 const inputRef = useRef();

 return (
  <input
   ref={inputRef}
   onChange={() => {
   console.log(inputRef.current.value);
   }}
  />
 );
};
```

4. useMemo

A function that runs only when a specific state or value changes

useMemo runs only when its dependencies change useful to avoid re-running expensive code like API calls on every render.

```
import { useState, useMemo } from "react";

const App = () => {
 const [url, setUrl] = useState("https://api.com");

 const data = useMemo(() => {
  console.log("API called");
 }, [url]);

 return (
  <div>
   <button onClick={() => setUrl("https://newapi.com")}>
    Change URL
   </button>
  </div>
 );
};
```

5. useCallback

Works exactly like useMemo, but instead of returning a value, it returns a function.

useCallback is like useMemo, but it returns a function instead of a value.
It's useful when passing stable functions as props, so they only change when needed.

```
import { useState, useCallback } from "react";

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

 const handleClick = useCallback(() => {
  console.log("Clicked! Count is:", count);
 }, [count]);

 return (
  <div>
   <button onClick={handleClick}>Click Me</button>
   <button onClick={() => setCount(count + 1)}>
  Add Count  </button>
  </div>
 );
};
```

6. useCallback

Works exactly like useMemo, but instead of returning a value, it returns a function.

useCallback is like useMemo, but it returns a function instead of a value.
It's useful when passing stable functions as props, so they only change when needed.

```
import { useState, useCallback } from "react";

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

  const handleClick = useCallback(() => {
    console.log("Clicked! Count is:", count);
  }, [count]);

  return (
    <div>
      <button onClick={handleClick}>Click Me</button>
      <button onClick={() => setCount(count + 1)}>
    Add Count  </button>
    </div>
  );
};
```

7.useLayoutEffect

useLayoutEffect works like useEffect, but it runs before the browser paints the screen.
Useful for measuring DOM size or making visual adjustments before render.

useEffect : Runs after the view is rendered (mounted).

useLayoutEffect : Runs before the view is rendered (mounted).

```
import { useLayoutEffect} from "react";

const App = () => {
 useLayoutEffect(() => {
  console.log("View has not mounted yet");
 }, []);

 // .. //
};
```