Intermediate useEffect Concepts in React

1. Execution Timing & Lifecycle

useEffect(() => { ... }) runs after render, but:

- No dependencies: runs after every render.
- Empty array []: runs only once after initial render (like componentDidMount).
- [someValue]: runs when someValue changes.

React batches state updates in events, but useEffect always runs after paint,

so don't rely on it for blocking logic or syncing the DOM before user sees it.

2. Cleanup Function Behavior

The return inside useEffect is a cleanup function.

Example:

```
useEffect(() => {
  const id = setInterval(() => console.log("Tick"), 1000);
  return () => {
    clearInterval(id); // cleanup
  };
}, []);
```

React will call the cleanup:

- Before the component unmounts
- AND before re-running the effect (if dependencies change)

3. Dependency Array Pitfalls

Example:

```
useEffect(() => {
  console.log(user.name);
}, []);
```

React will warn: "user.name is missing in the dependency array".

Objects are reference-based, even same content can mean a new object.

Fix:

```
const { name } = user;
useEffect(() => {
  console.log(name);
}, [name]);
```

Avoid [user] unless necessary, as it causes re-renders when object references change.

4. Avoiding Infinite Loops

Example of infinite loop:

```
useEffect(() => {
  setState("newValue");
}, [state]);
```

Fix with conditional logic or functional update:

```
setState(prev => {
  if (prev === "newValue") return prev;
  return "newValue";
});
```

5. Syncing with External Systems

To access the latest state in async or interval:

```
const countRef = useRef(count);

useEffect(() => {

  countRef.current = count;

}, [count]);


useEffect(() => {

  const interval = setInterval(() => {

    console.log("Current count:", countRef.current);

  }, 1000);

  return () => clearInterval(interval);

}, []);
```

Bonus: Custom Hook Example

```
function useWindowWidth() {

  const [width, setWidth] = useState(window.innerWidth);

  useEffect(() => {

    const handler = () => setWidth(window.innerWidth);

    window.addEventListener("resize", handler);

    return () => window.removeEventListener("resize", handler);

  }, []);

  return width;

}
```