---

react hooks?

In React, **Hooks** are special functions that let you "hook into" React features like state and lifecycle methods in function components — without using class components.

Common React Hooks:

1. **`useState()`**
   - Adds state to functional components.

```
js

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

2. **`useEffect()`**
   - Runs side effects (e.g., data fetching, DOM updates).

```
js

 useEffect(() => {
   console.log('Component mounted or updated');
 }, [dependency]);
```

3. **`useContext()`**
   - Accesses context values without writing a Consumer.

```
js

 const value = useContext(MyContext);
```

4. **`useRef()`**
   - Accesses DOM elements or keeps mutable variables that don’t trigger re-renders.

```
js

 const inputRef = useRef(null);
```

5. **`useMemo()`**
   - Memoizes expensive computations to avoid re-computation.

```
js
```

```
const result = useMemo(() => computeExpensiveValue(a, b), [a, b]);
```

6. **`useCallback()`**
   - Memoizes functions to avoid re-creating them on each render.

```
js

const handleClick = useCallback(() => doSomething(), []);
```

7. **`useReducer()`**
   - Alternative to `useState` for complex state logic.

```
js

const [state, dispatch] = useReducer(reducer, initialState);
```

8. **`useLayoutEffect()`**
   - Like `useEffect`, but fires synchronously after all DOM mutations.
9. **`useImperativeHandle()`**
   - Customizes the instance value exposed when using `ref` with a component.

---

give me a table so i can easily which does what works and where should i use which one

Here's a clear table summarizing **React Hooks**, what they do, and when to use them:

Hook	Purpose	When to Use
`useState()`	Adds state to functional components	When you need to store and update simple state (like a counter, form input)
`useEffect()`	Handles side effects (e.g., fetch, timer, DOM updates)	When you want to run code after render or when dependencies change
`useContext()`	Accesses values from a React Context	When you want to share data (like themes or auth) globally without prop drilling
`useRef()`	Stores a mutable reference to a DOM element or value	When you need to reference DOM or keep values without triggering re-renders
`useMemo()`	Caches the result of a computation	When you have an expensive calculation that shouldn't re-run on every render
`useCallback()`	Caches a function definition	When passing functions to child components to avoid unnecessary re-renders

Hook	Purpose	When to Use
`useReducer()`	Manages complex state logic with reducers	When `useState` becomes too complex (e.g., managing form inputs, toggles, etc.)
`useLayoutEffect()`	Like `useEffect` but runs **before** painting the DOM	When you need to measure layout or apply DOM changes **before** the screen paints
`useImperativeHandle()`	Customizes the instance value exposed by `ref`	When building a component that exposes methods to its parent (advanced use)
