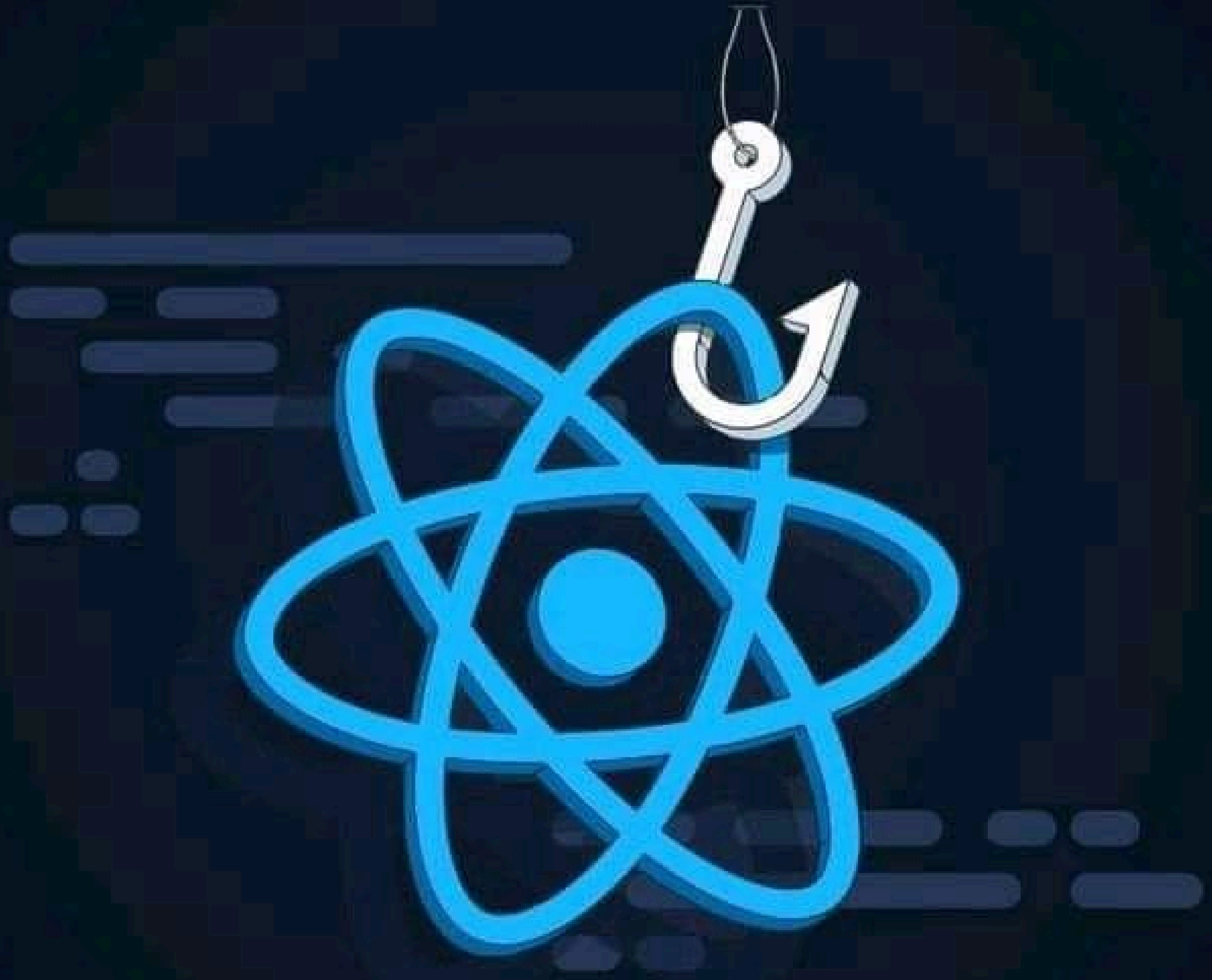
LEARN with SUMIT

React Hooks



Cheat Sheet

SWIPE

LEARN with SUMIT

State Management - useState()

DECLARE STATE

```
const [name, setName] = useState('initial value');
```

UPDATE STATE

```
setName('new value'); // directly
// or
setName((value) => 'new ' + value); // based on previous
                                                  state
```

SWIPE

LEARN with SUMIT

Side Effects useEffect()

TRIGGERS CALLBACK FUNCTION ONLY ONCE WHEN COMPONENT IS MOUNTED

```
useEffect(() => {
  // Side effects - HTTP request, setTimeout, etc.
}, []);
```

TRIGGERS CALLBACK FUNCTION WHEN DEPENDENCY 'VALUE' IS CHANGED

```
useEffect(() => {
  // Side effects - HTTP request, setTimeout, etc.
}, [value]);
```

CLEANUP SIDEEFFECTS WHEN COMPONENT IS UNMOUNTED

```
useEffect(() => {
  let timeout = setTimeout(doSomething, 5000);
  return () => clearTimeout(timeout);
}, [value]);
```

SWIPE

LEARN with SUMIT

Memoize a callback with useCallback( )

RETURNS NEW FUNCTION ONLY WHEN DEPENDENCIES CHANGE

```
const handleClick = useCallback(() => {
 doSomethingWith(param1, param2)
}, [param1, param2])
```

MEMOIZE CALLBACK FOR A DYNAMIC LIST OF ELEMENTS

```
const handleClick = useCallback((event) => {
  const button = event.target
  const value = button.getAttribute('data-value')
  doSomethingWith(value)
}, [])

<ul>
 {objects.map((obj) => (
   <li key={obj.id}>
     <button data-value={obj.value} onClick={handleClick}>
             {obj.value}
      </button>
   </li>
 ))}                                         // memoized
</ul>
```

SWIPE

LEARN with SUMIT

Memoize a value with useMemo( )

WILL TRIGGER ONLY WHEN DEPENDENCIES CHANGE

```
const value = useMemo(() => {
 // evaluates only when param1 or param2 change
 return expensiveOperation(param1, param2)
}, [param1, param2])
```

LEARN with SUMIT

Context api with useContext( )

AVOID PROPS DRILLING USING CONTEXT API

```
// create & export context
export const ThemeContext = createContext(null);

// wrap parent component with context provider
return (
    <ThemeContext.Provider value={{theme: 'dark'}}>
        <App />
    </ThemeContext.Provider>
)


// use context inside any child component
const { theme } = useContext(ThemeContext);
```

SWIPE

LEARN with SUMIT

Manage State with useReducer( )

INITIALIZE A LOCAL STATE AND CREATE REDUCER

```
const initialState = {
 value : 0
}
const reducer = (state, action) => {
    switch (action.type) {
        case 'increment':
            return { ...state, value: state.value + 1 };
        case 'set_to':
            return { ...state, value: action.value };
        default:
            throw new Error('Unhandled action');
    }
};
```

CREATE LOCAL STATE AND DISPATCH ACTIONS

```
const [state, dispatch] = useReducer(reducer, initialState)
...
<button onClick={() => { dispatch({ type: 'increment' })}} />
<button onClick={() => { dispatch({ type: 'set_to',value: 42 })}} />
```

SWIPE

LEARN with SUMIT

Create your own Custom Hook

CUSTOM HOOKS MUST START WITH use

```
const useApiResult = (param) => {
    const [result, setResult] = useState(null);
    useEffect(() => {
        // Your Task
    }, [param]);
    return { result };
};

// To use it in a component:
const { result } = useApiResult('some-param');
```

SWIPE