---

1. useState (Local State)

- **Purpose:** Local component state manage करने के लिए
- **Example:** Counter, Toggle, Form Inputs

```
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <h2>{count}</h2>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <button onClick={() => setCount(count - 1)}>Decrement</button>
    </div>
  );
}
```

---

2. Context API (Global State Sharing)

- **Purpose:** Global state share करने के लिए
- **Example:** Theme, User Auth, Language Preference

```
import React, { createContext, useContext, useState } from 'react';

const ThemeContext = createContext();

function App() {
  const [dark, setDark] = useState(false);
  return (
    <ThemeContext.Provider value={{ dark, setDark }}>
      <Toolbar />
    </ThemeContext.Provider>
  );
}

function Toolbar() {
  const { dark, setDark } = useContext(ThemeContext);
  return (
    <div style={{ background: dark ? 'black' : 'white', color: dark ?
'white' : 'black' }}>
      <button onClick={() => setDark(!dark)}>Toggle Theme</button>
    </div>
```

```
  );
}
```

---

3. Redux (Centralized State Management)

3.1 Why Redux?

- Centralized State
- Predictable updates
- Debuggable with Redux DevTools
- Scalable for large apps

3.2 Core Concepts

Concept	Explanation
Store	Central state container
Action	State change instruction
Reducer	Pure function updating state
Dispatch	Function to send action
Selector	Function to read state

3.3 Redux Flow

```
Component ---> dispatch(Action) ---> Reducer ---> Store updated --->
Component updated
```

3.4 Basic Redux Example (Counter)

```
import { createStore } from 'redux';

const initialState = { count: 0 };
const reducer = (state = initialState, action) => {
  switch(action.type) {
    case 'INCREMENT': return { count: state.count + 1 };
    case 'DECREMENT': return { count: state.count - 1 };
    default: return state;
  }
};

const store = createStore(reducer);
```

```
store.dispatch({ type: 'INCREMENT' });
console.log(store.getState());
```

3.5 Redux + React Example

```
import React from 'react';
import { createStore } from 'redux';
import { Provider, useSelector, useDispatch } from 'react-redux';

const initialState = { count: 0 };
const reducer = (state = initialState, action) => {
  switch(action.type){
    case 'INCREMENT': return { count: state.count + 1 };
    case 'DECREMENT': return { count: state.count - 1 };
    default: return state;
  }
};

const store = createStore(reducer);

function Counter() {
  const count = useSelector(state => state.count);
  const dispatch = useDispatch();
  return (
    <div>
      <h2>{count}</h2>
      <button onClick={() => dispatch({ type: 'INCREMENT' })}>+</button>
      <button onClick={() => dispatch({ type: 'DECREMENT' })}>-</button>
    </div>
  );
}

function App() {
  return (
    <Provider store={store}>
      <Counter />
    </Provider>
  );
}
```

3.6 Redux Toolkit Example (Simplified)

```
import { configureStore, createSlice } from '@reduxjs/toolkit';

const counterSlice = createSlice({
  name: 'counter',
  initialState: { count: 0 },
  reducers: {
```

```
    increment: state => { state.count += 1 },
    decrement: state => { state.count -= 1 }
  }
});

export const { increment, decrement } = counterSlice.actions;
const store = configureStore({ reducer: counterSlice.reducer });
export default store;
```

---

4. Redux vs Context vs useState Comparison

Feature	useState	Context API	Redux
Purpose	Local state	Global state	Centralized & predictable state
Scope	Single component	Multiple components	Multiple components, complex apps
Setup	Minimal	Low	Medium–High (RTK reduces)
Predictable	Direct	Provider based	Only via Actions & Reducers
Debugging	Basic	Basic	Advanced (Redux DevTools)
Async	useEffect	Manual	redux-thunk / createAsyncThunk
Performance	Fast	Can re-render consumers	Optimized & controlled
Best Use Case	Small forms, toggles	Theme, auth	Large apps, complex state, API data

---

5. Guidelines

1. **useState:** Small & local state.
2. **Context API:** Medium state, global sharing.
3. **Redux:** Large apps, multiple components, complex async logic.

End of Handbook