

React-এ ডেটা ম্যানেজমেন্ট বলতে কী বুঝায়?

React-এর মূল ভিত্তি হলো component-based UI, যেখানে প্রতিটা component-এর নিজের state থাকতে পারে। তবে যখন তুমি অনেকগুলো component-এর মধ্যে same data শেয়ার করতে চাও (যেমন user info, theme, cart, etc.), তখন তুমি একটি global state management system চাও—এখানেই আসে Redux, Zustand, Context API-এর মতো টুলগুলোর প্রয়োজন।

🔹 ১. Context API (Built-in React Tool)

✅ কবে ব্যবহার করবো:

- ছোট বা মাঝারি অ্যাপে
- যেখানে ২-৩টা component-এ shared state দরকার
- যেমন: dark mode, user authentication, language toggle

🧠 কীভাবে কাজ করে:

- `React.createContext()` দিয়ে একটা context তৈরি করো
- `Provider` দিয়ে state আর action গুলো পুরো অ্যাপ/পাশাপাশি component-এ পাঠাও
- `useContext()` দিয়ে যেকোনো child component সেই data access করতে পারে

🔍 সমস্যাসমূহ:

- অনেক বেশি nested component থাকলে performance issue হতে পারে
- যখন state update হয়, context-এর সব consumer component re-render করে → performance কমে যায়

🟡 9. Context API Example

🧠 CounterContext.jsx

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

const CounterContext = createContext();

export const useCounter = () => useContext(CounterContext);

export const CounterProvider = ({ children }) => {

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

  const increment = () => setCount(prev => prev + 1);

  const decrement = () => setCount(prev => prev - 1);

  return (

    <CounterContext.Provider value={{ count, increment, decrement }}>

      {children}

    </CounterContext.Provider>

  );

};
```

App.jsx

```
import React from 'react';

import { useCounter } from './CounterContext';

export default function App() {

  const { count, increment, decrement } = useCounter();

  return (

    <>

      <h1>Count: {count}</h1>

      <button onClick={increment}>+</button>

      <button onClick={decrement}>-</button>

    </>

  );

}
```

main.jsx

```
import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

import { CounterProvider } from './CounterContext';

ReactDOM.createRoot(document.getElementById('root')).render(

  <CounterProvider>

    <App />


  </CounterProvider>

);
```


ব্যাখ্যা (**Context API**):

Context API মূলত ছোটখাটো global state ম্যানেজ করতে ব্যবহৃত হয়।
createContext দিয়ে কনটেক্সট তৈরি করে, আর useContext দিয়ে তা ব্যবহার করি।


২. Zustand (Lightweight & Simple)

 কবে ব্যবহার করবো:


- lightweight কিন্তু powerful state ম্যানেজমেন্ট চাইলে
- Redux-এর মতো structure দরকার কিন্তু boilerplate চাই না
- বড় অ্যাপে lazy-loading বা dynamic imports ব্যবহার করতে চাইলে

 কীভাবে কাজ করে:

- `create()` ফাংশনের মাধ্যমে একটি store তৈরি করো
- store-এ state ও action define করো
- যেকোনো component-এ store কে custom hook (`useStore()`) দিয়ে access করো

 কীভাবে **efficient**:

- Zustand **reactive** এবং **shallow compare** ব্যবহার করে
- যেই অংশটা state change হয়েছে, শুধু সেটাই re-render করে
- middleware, persistence (`localStorage`), devtools সহজেই যুক্ত করা যায়

 Zustand এর বড় সুবিধা:

- একেবারে boilerplate-free
- কোনো Provider দরকার নেই
- scalable architecture

Zustand Example

✓ Installation:

bash

CopyEdit

```
npm install zustand
```

🧠 store/counterStore.js

```
import { create } from 'zustand';
```

```
const useCounterStore = create((set) => ({  
  count: 0,  
  increment: () => set(state => ({ count: state.count + 1 })),  
  decrement: () => set(state => ({ count: state.count - 1 })),  
}));
```

```
export default useCounterStore;
```

App.jsx

```
import React from 'react';

import useCounterStore from '../store/counterStore';

export default function App() {

  const { count, increment, decrement } = useCounterStore();

  return (

    <>

      <h1>Count: {count}</h1>

      <button onClick={increment}>+</button>

      <button onClick={decrement}>-</button>

    </>

  );


}
```

ব্যাখ্যা (Zustand):


Zustand হলো একটি ছোট, দ্রুত স্টেট ম্যানেজমেন্ট টুল। কোন Provider দরকার হয় না।

`create()` দিয়ে একটা store বানানো হয় আর `useCounterStore()` hook-এর মাধ্যমে তা ব্যবহার করা হয়।

৩. Redux (Industry-grade, Predictable State Manager)

 কবে ব্যবহার করবো:


- বড় বা complex অ্যাপে (dashboard, e-commerce)
- যেখানে state গুলোর উপর strict control দরকার (e.g. undo/redo, debugging)
- অনেক ধরনের data ও nested update handle করতে হয়

 কীভাবে কাজ করে:

- একটি centralized store থাকে যেখানে সব state থাকে
- অ্যাকশন `dispatch()` করে store-এ পরিবর্তন করা হয়
- `reducer()` ফাংশনের মাধ্যমে state update হয় (pure function)
- `useSelector()` দিয়ে UI তে state পাঠানো হয়

 Redux Toolkit:

- Redux Toolkit ব্যবহার করলে boilerplate অনেক কমে যায়
- `createSlice`, `configureStore` ইত্যাদি দিয়ে সহজেই reducer/action/store তৈরি করা যায়

 সমস্যাসমূহ:

- vanilla Redux অনেক বেশি boilerplate heavy
- Redux শিখতে প্রথমে কঠিন মনে হতে পারে

React Redux Example (with Toolkit)

✓ Installation:

```
npm install @reduxjs/toolkit react-redux
```

🔧 store/counterSlice.js

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

const counterSlice = createSlice({
  name: 'counter',
  initialState: { value: 0 },
  reducers: {
    increment: state => { state.value += 1 },
    decrement: state => { state.value -= 1 }
  }
});

export const { increment, decrement } = counterSlice.actions;
export default counterSlice.reducer;
```

store/store.js

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

import counterReducer from './counterSlice';

export const store = configureStore({
  reducer: { counter: counterReducer }
});
```

main.jsx

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import { Provider } from 'react-redux';
import { store } from './store/store';

ReactDOM.createRoot(document.getElementById('root')).render(
  <Provider store={store}>
    <App />
  </Provider>
);
```

App.jsx

```
import React from 'react';

import { useSelector, useDispatch } from 'react-redux';

import { increment, decrement } from './store/counterSlice';

export default function App() {

  const count = useSelector(state => state.counter.value);

  const dispatch = useDispatch();

  return (

    <>

      <h1>Count: {count}</h1>

      <button onClick={() => dispatch(increment())}>➕</button>

      <button onClick={() => dispatch(decrement())}>➖</button>

    </>

  );








}
```

ব্যাখ্যা (Redux):

Redux store-এ state থাকে, আর dispatch দিয়ে আমরা সেই state update করি।
useSelector দিয়ে state কে UI-তে আনছি।
Provider দিয়ে পুরো App কে redux-এর access দিচ্ছি।



তুলনামূলক বিশ্লেষণ:

Feature	Context API	Zustand	Redux (Toolkit)
 Installation	Built-in React	✓ lightweight (5kb)	✗ Extra dependency
 Learning Curve	Easy	Very Easy	Moderate to Hard
 Boilerplate	Low	Very Low	High (Redux Toolkit → Moderate)
 Performance	Low (many re-renders)	High	High (with proper config)
 Devtool Support	Limited	Good (via middleware)	Excellent
 State Sharing	Simple apps	Medium to large apps	Large scale apps
 Middleware Support	Manual	Easy with middleware	Powerful



কবে কোনটা বেছে নেবো?

প্রয়োজন

টুল

ছোট প্রজেক্ট, theme toggle, user info

Context API

Simple কিন্তু powerful state management

Zustand

Complex business logic, devtool, middleware, time-travel debugging দরকার

Redux Toolkit



উদাহরণ: যদি তোমার একটা অ্যাপ থাকে যেখানে—

- Header-এ user login info দেখাবে,
- Sidebar-এ language toggle থাকবে,
- Cart button-এ items number থাকবে,

👉 তবে তুমি হয় **Context API** দিয়ে user info ও theme manage করতে পারো, আর **Zustand/Redux** দিয়ে cart state maintain করতে পারো।



উপসংহার

React-এ ডেটা ম্যানেজমেন্টের মূল লক্ষ্য হলো—

component গুলোর মধ্যে **data efficiently** ও **predictably** শেয়ার ও **update** করা।

যত বড় অ্যাপ হবে, তত ভালো ও maintainable state management structure দরকার হবে।

তাই সময় বুঝে টুল সিলেক্ট করাটাই বুদ্ধিমানের কাজ।