State Management

🔹 What is “State”?

State মানে হলো →
👉 আপনার অ্যাপের **current data / condition** যা সময়ের সাথে **change হতে পারে**।

Examples:

- User logged in নাকি logged out
- Button click হলে counter বাড়ে
- Form input এর value
- API থেকে আসা data
- Dark mode on / off

👉 এগুলো সবই **State**।

🔹 Then what is State Management?

State Management মানে হলো:

State কোথায় থাকবে, কীভাবে update হবে, আর কোন component গুলো সেই state ব্যবহার করবে — এগুলোকে clean ও predictableভাবে handle করা।

Simple কথায়:

- State **store করা**
- State **update করা**
- State **share করা** (multiple components এর মাঝে)

🔹 Problem without State Management

ধরি একটা app 👇

App

├── Navbar (user info লাগে)

├── Sidebar (user role লাগে)

└── Profile (user details লাগে)

Problem:

- User data শুধু Profile component এ থাকলে
- Navbar + Sidebar কে data দিতে হলে
  👉 **props drilling শুরু হয়**

App → Navbar

App → Sidebar

App → Profile

😖 Code messy
😖 Hard to maintain
😖 Bug-prone

🔹 Solution: State Management

State কে **central জায়গায় রাখো**
সব component প্রয়োজন অনুযায়ী access করবে

Global State (Store)

├── user

├── theme

├── cart

Components:

- শুধু **read / update** করবে
- Direct data flow থাকবে

🔹 Types of State

1. Local State

একটা component এর ভেতরে সীমাবদ্ধ

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

✓ Simple
❌ Share করা যায় না

2. Global State

পুরো app জুড়ে use হয়

Examples:

- Auth user
- Shopping cart
- Theme

👉 **এখানে State Management tool দরকার**

🔹 Popular State Management Tools (React / Next.js)

🔹 Built-in

- useState
- useReducer
- Context API

◆ External Libraries

Tool	Use Case
Redux / Redux Toolkit	Large apps, predictable flow
Zustand	Lightweight, simple
Recoil	Facebook experimental
Jotai	Atomic state
MobX	Reactive style

👉 **FAANG-level apps mostly use Redux Toolkit / Zustand**

◆ Example: Simple State Management (Context API)

```
const UserContext = createContext();


function App() {
  const [user, setUser] = useState(null);


  return (
    <UserContext.Provider value={{ user, setUser }}>
      <Navbar />
```

```
      <Profile />
    </UserContext.Provider>
  );
}
```

Now:

- Navbar → user info
- Profile → update user
  without props drilling ✅

🔹 Why State Management is IMPORTANT (Interview 💡)

FAANG interview এ জিজ্ঞেস করে:

- Why global state?
- Props drilling problem?
- Redux vs Context?
- When NOT to use Redux?

👉 State Management = **Scalability + Maintainability**

🔹 Real-life Analogy 🏪

Think of **State** like:

- Shop inventory

State Management like:

- Central warehouse system
  Everyone (cashier, manager, website) sees same data

◆ Summary (TL;DR)

✓ State = app এর current data
✓ State Management = state handle করার system
✓ Small app → local state
✓ Large app → global state
✓ Clean code + scalable architecture

সহজ কথায় বলতে গেলে, **State Management** হলো একটি অ্যাপ্লিকেশনের বিভিন্ন ডেটা বা তথ্যকে সুশৃঙ্খলভাবে নিয়ন্ত্রণ করার পদ্ধতি। যখন একটি অ্যাপে অনেক ডেটা থাকে এবং সেই ডেটাগুলো বিভিন্ন স্ক্রিন বা কম্পোনেন্টের মধ্যে শেয়ার করতে হয়, তখন স্টেট ম্যানেজমেন্টের প্রয়োজন পড়ে।

নিচে বিস্তারিত আলোচনা করা হলো:

১. State (স্টেট) কী?

একটি অ্যাপ্লিকেশনের নির্দিষ্ট কোনো মুহূর্তের অবস্থা বা ডেটাই হলো **State**।

- **উদাহরণ:** একটি ফেসবুক অ্যাপে আপনি যখন একটি ছবিতে লাইক দেন, তখন লাইকের সংখ্যা ১ বেড়ে যায়। এই যে "লাইক সংখ্যা" বা "লাইক দেওয়া হয়েছে কি না" এই তথ্যটিই হলো ওই অ্যাপের একটি স্টেট।
- **সহজ উদাহরণ:** একটি পানির বোতলের কথা ভাবুন। বোতলটি পূর্ণ আছে নাকি খালি—এটি হলো তার বর্তমান 'স্টেট'।

২. State Management কেন প্রয়োজন?

ছোট অ্যাপে সাধারণত ডাটা আদান-প্রদান করা সহজ। কিন্তু অ্যাপ যত বড় হয়, ডেটা হ্যান্ডেল করা তত কঠিন হয়ে পড়ে। কেন এটি দরকার তার প্রধান কারণগুলো হলো:

- **Data Sharing:** অ্যাপের এক পেজ থেকে অন্য পেজে ডেটা পাঠানোর জন্য। যেমন: লগইন করার পর আপনার নাম প্রোফাইল পেজ এবং হোম পেজ উভয় জায়গায় দেখানোর জন্য।
- **UI Update:** ডেটা পরিবর্তন হওয়ার সাথে সাথে স্ক্রিনে যেন অটোমেটিক সেই পরিবর্তন দেখা যায় (Re-render)।
- **Prop Drilling এড়ানো:** যখন আপনাকে একটি ডেটা ৫-৬ লেয়ার নিচের কোনো কম্পোনেন্টে পাঠাতে হয়, তখন সব মাঝখানের কম্পোনেন্ট দিয়ে ডেটা পাস করা খুব ঝামেলার। একে বলে **Prop Drilling**। স্টেট ম্যানেজমেন্ট এটি সমাধান করে।
- **Consistency:** পুরো অ্যাপে সব জায়গায় যেন একই ডেটা থাকে তা নিশ্চিত করা।

৩. স্টেটের প্রকারভেদ

সাধারণত স্টেটকে দুই ভাগে ভাগ করা যায়:

1. **Local State (লোকাল স্টেট):** যা শুধুমাত্র একটি নির্দিষ্ট উইজেট বা কম্পোনেন্টের মধ্যে সীমাবদ্ধ। (যেমন: একটি সার্চ বক্সের টেক্সট)।
2. **Global State (গ্লোবাল স্টেট):** যা পুরো অ্যাপের যেকোনো জায়গা থেকে অ্যাক্সেস করা যায়। (যেমন: ইউজারের প্রোফাইল ইনফরমেশন বা শপিং কার্ট)।

৪. জনপ্রিয় State Management টুলস

আপনি কোন ফ্রেমওয়ার্ক ব্যবহার করছেন তার ওপর ভিত্তি করে টুলসগুলো ভিন্ন হয়:

ফ্রেমওয়ার্ক (Framework)	**স্টেট ম্যানেজমেন্ট টুলস**
React	Context API, Redux Toolkit, Zustand, Recoil
Flutter	Provider, Riverpod, Bloc, GetX
Vue	Vuex, Pinia
Angular	NgRx, RxJS (BehaviorSubject)

৫. একটি বাস্তব উদাহরণ (E-commerce App)

ধরা যাক আপনি একটি ই-কমার্স অ্যাপ বানাচ্ছেন।

- **স্টেট:** ইউজারের 'Cart' বা ঝুড়িতে কয়টি পণ্য আছে।
- **ম্যানেজমেন্ট:** আপনি যখন "Product Detail" পেজ থেকে একটি পণ্য 'Add to Cart' বাটনে ক্লিক করবেন, তখন গ্লোবাল স্টেটে ১টি আইটেম যোগ হবে। সাথে সাথে অ্যাপের উপরে থাকা 'Cart Icon'-এ সংখ্যাটি বেড়ে যাবে। এখানে সব কাজ হয়েছে একটি সেন্ট্রাল সিস্টেমের মাধ্যমে, যা-ই হলো **State Management**।

উপসংহার

স্টেট ম্যানেজমেন্ট হলো আপনার অ্যাপের ব্রেইন বা মগজ, যা সব তথ্য মনে রাখে এবং সময়মতো সঠিক জায়গায় পৌঁছে দেয়। এটি ছাড়া বড় বা প্রোফেশনাল অ্যাপ তৈরি করা প্রায় অসম্ভব।