2. Hooks

Monday, 15 December 2025 6:25 PM

One-Line Summary

Hook	Remember As
useState	Memory box - stores and updates values
useEffect	Alarm - "when X happens, do Y"
useCallback	Saved recipe - remembers how to do something
Custom Hook	Backpack - bundle of related tools

Imagine a Video Game Character

Your game character needs to:
1. Remember things (health points, coins collected, current level)
2. Do things when something happens (when game starts, load saved progress)
3. Use special powers (abilities that can be reused)

---

1. **useState = Character's Memory** 🧠

```
const [coins, setCoins] = useState(0);
```

Your character remembers they have 0 coins.
When they collect a coin: setCoins(coins + 1) → Now they remember they have 1 coin!

Simple: It's a box that stores a value and updates the screen when it changes.

---
2. **useEffect = "When This Happens, Do That"** ⚡

```
useEffect(() => {
  loadSavedGame();  // Load saved game when game starts
}, []);
```

Simple: It's an alarm that triggers when something happens.
- Game starts? → Load saved progress
- Player enters new level? → Play level music

---
3. **Custom Hooks = Backpack of Tools** 🎒

```
const { loading, transactions, fetchData } = useCancelPaymentTransactions();
```

Instead of carrying tools separately, you put them in a backpack.

- useFPTITracker = Analytics backpack (tracks what user does)
- useCancelPaymentTransactions = API backpack (fetches data)
- useCancelPaymentFilters = Filter backpack (handles filtering)

Backend Analogy:

React Hook	**Backend Equivalent**
useState	Instance variable in a class
useEffect	@PostConstruct / lifecycle method
useCallback	Memoized/cached method
Custom hooks (e.g., useFPTITracker)	Service class / Utility class

Example - Custom Hook Pattern:

```
// useCancelPaymentTransactions.ts - This is a "custom hook"
// Think of it like a Service class in Java/Spring

export const useCancelPaymentTransactions = (config) => {
  // Internal state (like private fields)
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  // Methods (like public service methods)
  const loadTransactions = useCallback(async () => {
    setLoading(true);
    const result = await api.fetchTransactions();
    setLoading(false);
    return result;
  }, []);

  // Return public API (like exposing service methods)
  return {
    loading,
    error,
    loadTransactions,
  };
};

Usage in component:
const { loading, loadTransactions } = useCancelPaymentTransactions(config);
```

- useState → stores data across renders
- useEffect → reacts to renders / changes
- useEffect does **not** store data
- Custom hooks → reusable combinations of hooks
- Custom hooks do **not** share state
- Everything is re-executed, but React preserves hook state