set-state-in-effect

Validates against calling setState synchronously in an effect, which can lead to re-renders that degrade performance.

Rule Details

Setting state immediately inside an effect forces React to restart the entire render cycle. When you update state in an effect, React must re-render your component, apply changes to the DOM, and then run effects again. This creates an extra render pass that could have been avoided by transforming data directly during render or deriving state from props. Transform data at the top level of your component instead. This code will naturally re-run when props or state change without triggering additional render cycles.

Synchronous `setState` calls in effects trigger immediate re-renders before the browser can paint, causing performance issues and visual jank. React has to render twice: once to apply the state update, then again after effects run. This double rendering is wasteful when the same result could be achieved with a single render.

In many cases, you may also not need an effect at all. Please see You Might Not Need an Effect for more information.

Common Violations

This rule catches several patterns where synchronous setState is used unnecessarily:

- Setting loading state synchronously
- Deriving state from props in effects
- Transforming data in effects instead of render

Invalid

Examples of incorrect code for this rule:

```
// ❌ Synchronous setState in effect
function Component({data}) {
  const [items, setItems] = useState([]);

  useEffect(() => {
    setItems(data); // Extra render, use initial state instead
  }, [data]);
}

// ❌ Setting loading state synchronously
function Component() {
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    setLoading(true); // Synchronous, causes extra render
    fetchData().then(() => setLoading(false));
  }, []);
}

// ❌ Transforming data in effect
function Component({rawData}) {
  const [processed, setProcessed] = useState([]);

  useEffect(() => {
    setProcessed(rawData.map(transform)); // Should derive in render
  }, [rawData]);
}

// ❌ Deriving state from props
function Component({selectedId, items}) {
  const [selected, setSelected] = useState(null);

  useEffect(() => {
    setSelected(items.find(i => i.id === selectedId));
  }, [selectedId, items]);
}
```

Valid

Examples of correct code for this rule:

```
// ✅ setState in an effect is fine if the value comes from a ref
function Tooltip() {
  const ref = useRef(null);
  const [tooltipHeight, setTooltipHeight] = useState(0);

  useLayoutEffect(() => {
    const { height } = ref.current.getBoundingClientRect();
    setTooltipHeight(height);
  }, []);
}

// ✅ Calculate during render
function Component({selectedId, items}) {
  const selected = items.find(i => i.id === selectedId);
  return <div>{selected?.name}</div>;
}
```

When something can be calculated from the existing props or state, don't put it in state. Instead, calculate it during rendering. This makes your code faster, simpler, and less error-prone. Learn more in You Might Not Need an Effect.

이전
refs

다음
set-state-in-render

React 학습하기

빠르게 시작하기

설치하기

API 참고서

React APIs

React DOM APIs