**HOOK**



Hooks are a new addition in React 16.8. They let you use state and other React features without writing a class. Hooks are functions that let you “hook into” React state and lifecycle features from function components. Hooks don’t work inside classes — they let you use React without classes. (We do not recommend rewriting your existing components overnight but you can start using Hooks in the new ones if you’d like.)

React provides a few built-in Hooks like useState. You can also create your own Hooks to reuse stateful behavior between different components. We’ll look at the built-in Hooks first.

**Effect Hook**

You’ve likely performed data fetching, subscriptions, or manually changing the DOM from React components before. We call these operations “side effects” (or “effects” for short) because they can affect other components and can’t be done during rendering.

The Effect Hook, useEffect, adds the ability to perform side effects from a function component. It serves the same purpose as componentDidMount, componentDidUpdate, and componentWillUnmount in React classes, but unified into a single API

When you call useEffect, you’re telling React to run your “effect” function after flushing changes to the DOM. Effects are declared inside the component so they have access to its props and state. By default, React runs the effects after every render — including the first render. Effects may also optionally specify how to “clean up” after them by returning a function. Hooks let you organize side effects in a component by what pieces are related (such as adding and removing a subscription), rather than forcing a split based on lifecycle methods.

**Rules of Hooks**

Hooks are JavaScript functions, but they impose two additional rules:

* Only call Hooks **at the top level**. Don’t call Hooks inside loops, conditions, or nested functions.
* Only call Hooks **from React function components**. Don’t call Hooks from regular JavaScript functions.