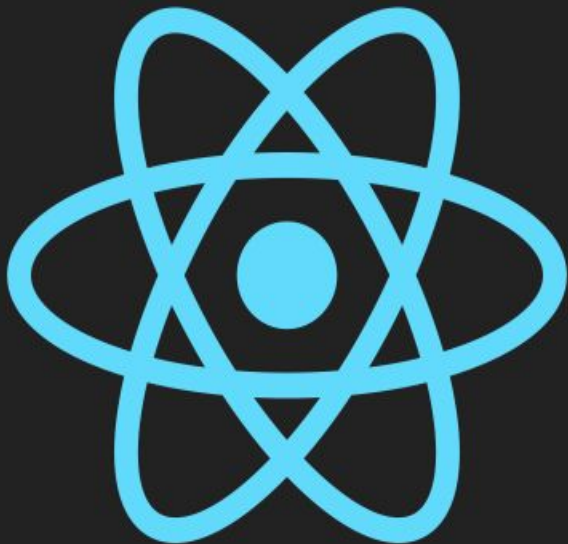



# REACT COMPONENT LIFECYCLE

Evive Health


[WWW.GOEVIVE.COM](http://WWW.GOEVIVE.COM)

# What is React?





**React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It is maintained by Facebook, Instagram and a community of individual developers and corporations.**





evive



*Why React?*



codecademy







- It makes writing Javascript easier
- Components are the future of web development
- React is extremely efficient
- It's awesome for SEO
- It gives you out-of-the-box developer tools
- It Allows developers to create large web-applications that use data and can change over time without reloading the page
- Bonus: Mobile Apps using React Native

# History



React was created by Jordan Walke, a software engineer at Facebook. He was influenced by Angular and XHP, an HTML component framework for PHP. It was first deployed on Facebook's news feed in 2011 and later on Instagram.com in 2012. It was open-sourced at JSConf US in May 2013.

# React.Component

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.

`React.Component` is provided by React.



# Overview

`React.Component` is an abstract base class, so it rarely makes sense to refer to `React.Component` directly.

Instead, you will typically subclass it, and define at least a `render()` method.

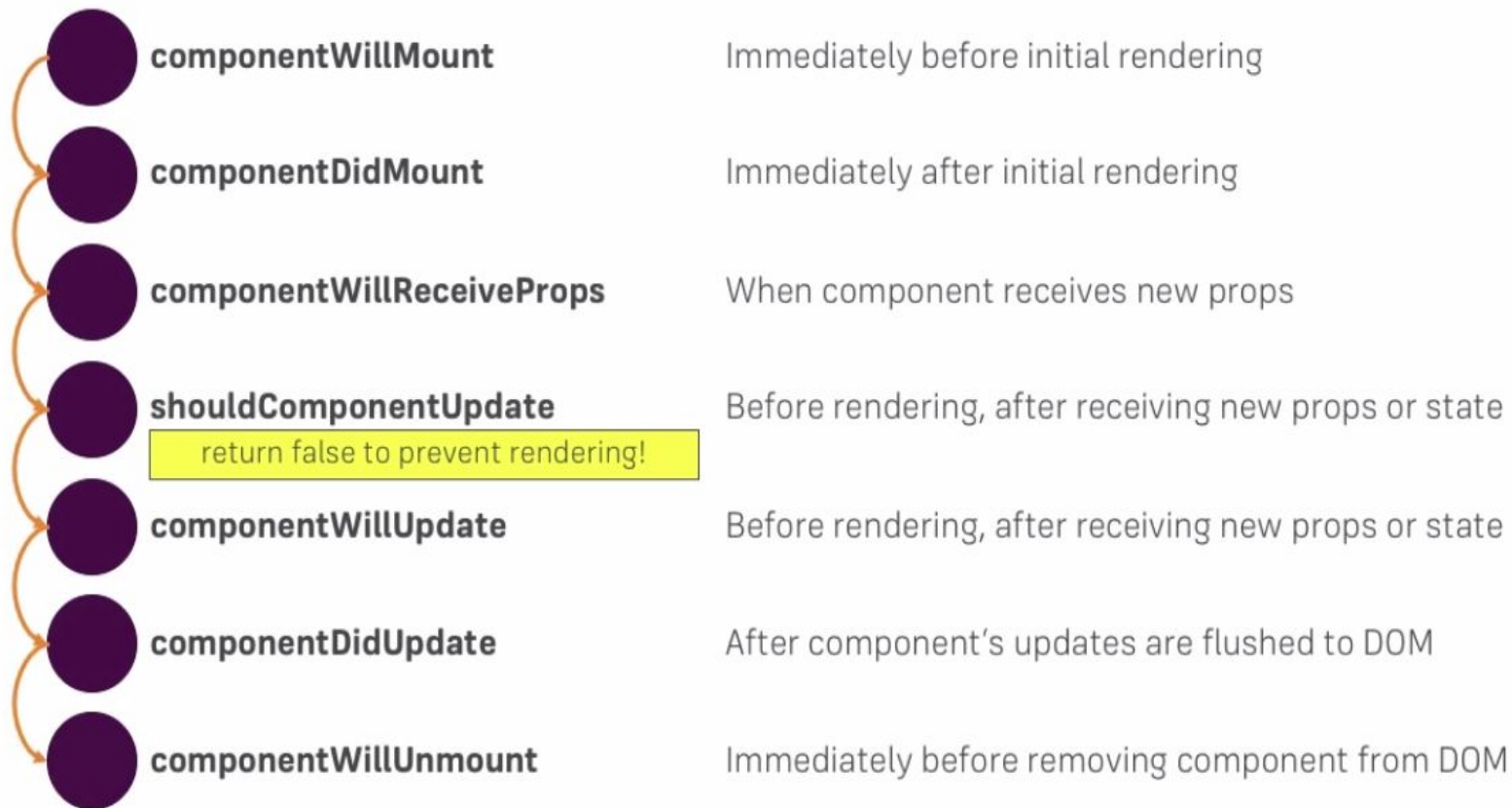
```
class Greeting extends React.Component {  
  render() {  
    return <h1>Hello, {this.props.name}</h1>;  
  }  
}
```



# The Component Lifecycle

Each component has several “lifecycle methods” that you can override to run code at particular times in the process. Methods prefixed with `will` are called right before something happens, and methods prefixed with `did` are called right after something happens.

# Component Lifecycle



# MOUNTING

- constructor()
  - componentWillMount()
  - render()
  - componentDidMount()
-

# UPDATING

- componentWillReceiveProps()
  - shouldComponentUpdate()
  - componentWillUpdate()
  - render()
  - componentDidUpdate()
-

# UN-MOUNTING

- `componentWillUnmount()`



# ERROR-HANDLING

- `componentDidCatch()`





## Constructor()

The constructor for a React component is called before it is mounted. When implementing the constructor for a `React.Component` subclass, you should call `super(props)` before any other statement. Otherwise, `this.props` will be undefined in the constructor, which can lead to bugs. The constructor is the right place to initialize state. If you don't initialize state and you don't bind methods, you don't need to implement a constructor for your React component. It's okay to initialize state based on props. This effectively "forks" the props and sets the state with the initial props. Here's an example of a valid `React`.

### Component subclass constructor:

```
constructor(props) {  
  super(props);  
  this.state = {  
    color: props.initialColor  
  };  
}
```

## **`componentWillMount()`**

`componentWillMount()` is invoked immediately before mounting occurs. It is called before `render()`, therefore setting state synchronously in this method will not trigger a re-rendering. Avoid introducing any side-effects or subscriptions in this method. This is the only lifecycle hook called on server rendering. Generally, we recommend using the `constructor()` instead.

## `componentDidMount()`

`componentDidMount()` is invoked immediately after a component is mounted.

Initialization that requires DOM nodes should go here. If you need to load data from a remote endpoint, this is a good place to instantiate the network request. Setting state in this method will trigger a re-rendering.

## `componentWillReceiveProps()`

`componentWillReceiveProps()` is invoked before a mounted component receives new props. If you need to update the state in response to prop changes (for example, to reset it), you may compare `this.props` and `nextProps` and perform state transitions using `this.setState()` in this method.

Note that React may call this method even if the props have not changed, so make sure to compare the current and next values if you only want to handle changes. This may occur when the parent component causes your component to re-render. React doesn't call `componentWillReceiveProps` with initial props during mounting. It only calls this method if some of component's props may update. Calling `this.setState` generally doesn't trigger `componentWillReceiveProps`.

## `shouldComponentUpdate()`

Use `shouldComponentUpdate()` to let React know if a component's output is not affected by the current change in state or props. The default behavior is to re-render on every state change, and in the vast majority of cases you should rely on the default behavior.

`shouldComponentUpdate()` is invoked before rendering when new props or state are being received. Defaults to `true`. This method is not called for the initial render or when `forceUpdate()` is used. Returning `false` does not prevent child components from re-rendering when *their* state changes. Currently, if `shouldComponentUpdate()` returns `false`, then `componentWillUpdate()`, `render()`, and `componentDidUpdate()` will not be invoked. Note that in the future React may treat `shouldComponentUpdate()` as a hint rather than a strict directive, and returning `false` may still result in a re-rendering of the component. If you determine a specific component is slow after profiling, you may change it to inherit from `React.PureComponent` which implements `shouldComponentUpdate()` with a shallow prop and state comparison. If you are confident you want to write it by hand, you may compare `this.props` with `nextProps` and `this.state` with `nextState` and return `false` to tell React the update can be skipped.

## `componentWillUpdate()`

`componentWillUpdate()` is invoked immediately before rendering when new props or state are being received. Use this as an opportunity to perform preparation before an update occurs. This method is not called for the initial render.

Note that you cannot call `this.setState()` here; nor should you do anything else (eg dispatch a redux action) that would trigger an update to a React component before `componentWillUpdate` returns. Use `componentWillReceiveProps()` if you need to update state in response to props changes.

**Note:** `componentWillUpdate()` will not be invoked if `shouldComponentUpdate()` returns false.



## `componentDidUpdate()`

`componentDidUpdate()` is invoked immediately after updating occurs. This method is not called for the initial render.

## `componentWillUnmount()`

`componentWillUnmount()` is invoked immediately before a component is unmounted and destroyed. Perform any necessary cleanup in this method, such as invalidating timers, canceling network requests, or cleaning up any DOM elements that were created in `componentDidMount`

## `componentDidCatch()`

Error boundaries are React components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI instead of the component tree that crashed. Error boundaries catch errors during rendering, in lifecycle methods, and in constructors of the whole tree below them.

A class component becomes an error boundary if it defines this lifecycle method.

# Find examples on github



<https://github.com/GAVISHSETHI/React-Component-Lifecycle>



