<https://reactjs.org/docs>

# Babel



# [**Why does React require Babel and Webpack to work?**](https://stackoverflow.com/questions/43175140/why-does-react-require-babel-and-webpack-to-work)

React doesn't "need" babel or webpack but the library is built on the concept of using ES6 javascript syntax and JSX (essentially HTML in JS).

React however can be used without [ES6](https://facebook.github.io/react/docs/react-without-es6.html) and [JSX](https://facebook.github.io/react/docs/react-without-jsx.html) which would remove the initial need for Babel but you would lose the potential benefits of ES6 and JSX.

## What is Webpack?

How does Webpack compare to Grunt, Gulp, Browserify, Brunch, etc?

Webpack is a **build tool that puts all of your assets, including Javascript, images, fonts, and CSS, in a dependency graph.** Webpack lets you use require() in your source code to point to local files, like images, and decide how they're processed in your final Javascript bundle, like replacing the path with a URL pointing to a CDN.

## Should I Use Webpack?

If you're building a complex Front End™ application with many **non-code static assets** such as CSS, images, fonts, etc, then **yes, Webpack will give you great benefits.**

If your application is fairly small, and you don't have many static assets and you only need to build one Javascript file to serve to the client, then **Webpack might be more overhead than you need.**

# JSX

# JS is standard javascript, JSX is an HTML-like syntax that you can use with React to (theoretically) make it easier and more intuitive to create React components. [As the docs say](https://facebook.github.io/react/docs/jsx-in-depth.html#why-jsx), the only purpose is to make it easier to create React components... there's not much else there. Without JSX, creating large, nested HTML documents using JS syntax would be a large pain in the rear; JSX simply makes that process easier.



**JSX In Depth**

Fundamentally, JSX just provides syntactic sugar for the React.createElement(component, props, ...children) function. The JSX code:

<MyButton color="blue" shadowSize={2}>

Click Me

</MyButton>

compiles into:

React.createElement(

MyButton,

{color: 'blue', shadowSize: 2},

'Click Me'

)

You can also use the self-closing form of the tag if there are no children. So:

<div className="sidebar" />

compiles into:

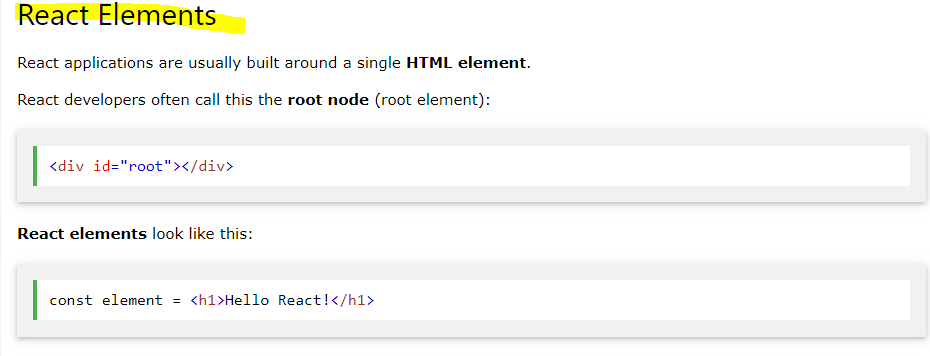
React.createElement(

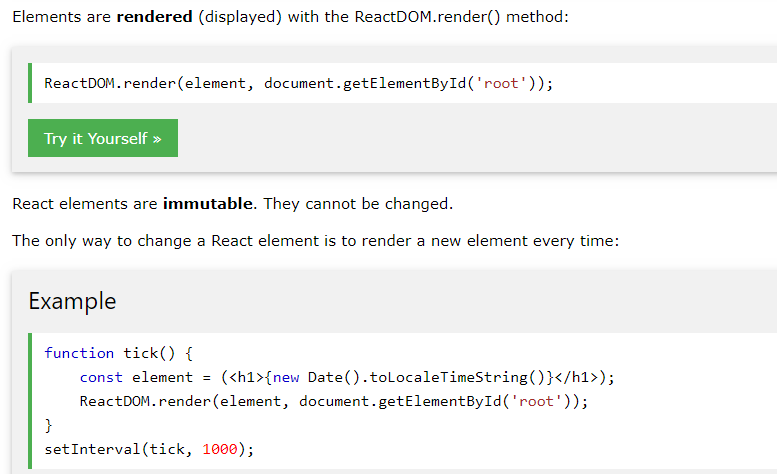
'div',

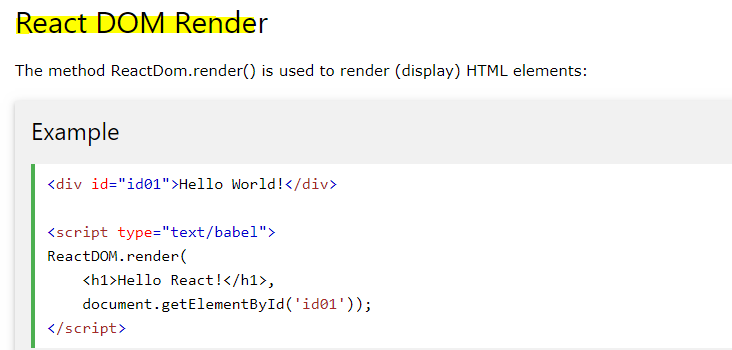
{className: 'sidebar'},

null

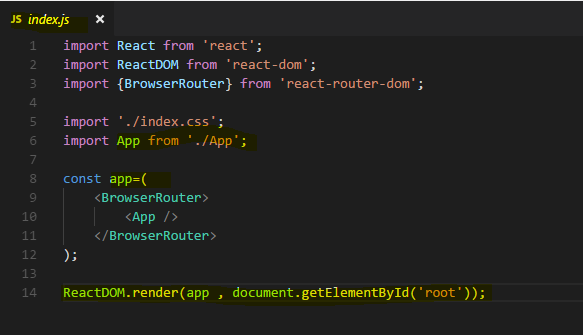
)







**For example entry point of react application which is served by server**

**And here’s the root component ‘App’ getting render to id=root.** 

**This index.js file is the entry point for bundler/builder(like webpack) for making a ‘bundle.js’ file out of whole react application and attaching the bundle.js file to index.html**

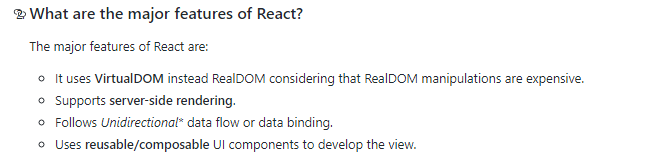
# What Is React?

**React** is a JavaScript library for building user interfaces. It lets you compose complex UIs from small and isolated pieces of code called “components”.

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex UIs from small and isolated pieces of code called “components”.

[React](https://facebook.github.io/react/) is a JavaScript library for building user interfaces. It is the view layer for web applications. It lets you compose complex UIs from small and isolated pieces of code called “components”.

At the heart of all React applications are **components**. A component is a self-contained module that renders some output.



## Try React

React has been designed from the start for gradual adoption, and **you can use as little or as much React as you need.** Whether you want to get a taste of React, add some interactivity to a simple HTML page, or start a complex React-powered app, the links in this section will help you get started.

If you’re interested in playing around with React, you can use an online code playground. Try a Hello World template on [CodePen](https://reactjs.org/redirect-to-codepen/hello-world" \t "_blank) or [CodeSandbox](https://codesandbox.io/s/new).

### (Without JSX)Add React to a Website

Use as little or as much React as you need.

React has been designed from the start for gradual adoption, and **you can use as little or as much React as you need**. Perhaps you only want to add some “sprinkles of interactivity” to an existing page. React components are a great way to do that.

The majority of websites aren’t, and don’t need to be, single-page apps. With **a few lines of code and no build tooling**, try React in a small part of your website. You can then either gradually expand its presence, or keep it contained to a few dynamic widgets.

we will show how to add a React component to an existing HTML page.

There is no complicated tools or install requirements.

Step 1: Add a DOM Container to the HTML

First, open the HTML page you want to edit. Add an empty <div> tag to mark the spot where you want to display something with React. For example:

<!-- ... existing HTML ... -->

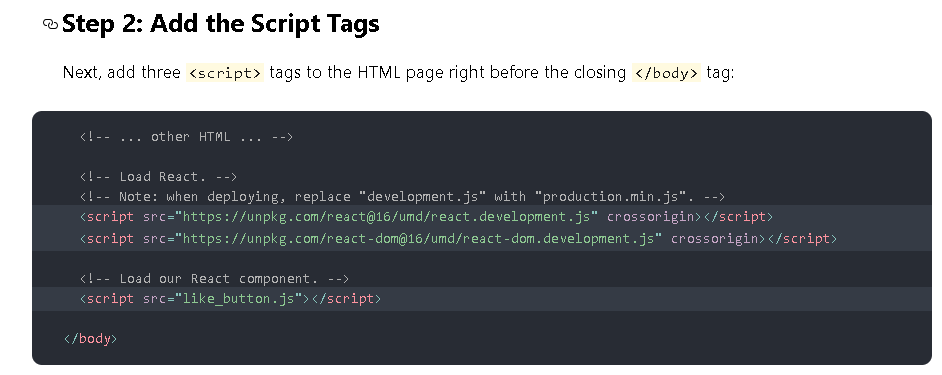
<div id="like\_button\_container"></div>

<!-- ... existing HTML ... -->

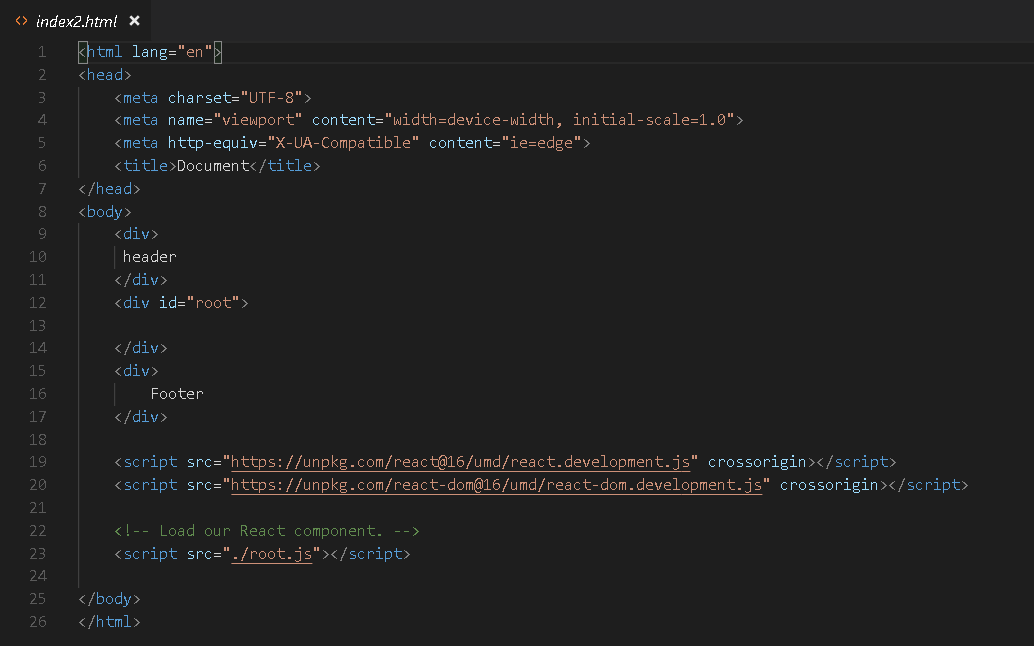
We gave this <div> a unique id HTML attribute. This will allow us to find it from the JavaScript code later and display a React component inside of it.

**Tip**

You can place a “container” <div> like this **anywhere** inside the <body> tag. You may have as many independent DOM containers on one page as you need. They are usually empty — React will replace any existing content inside DOM containers.

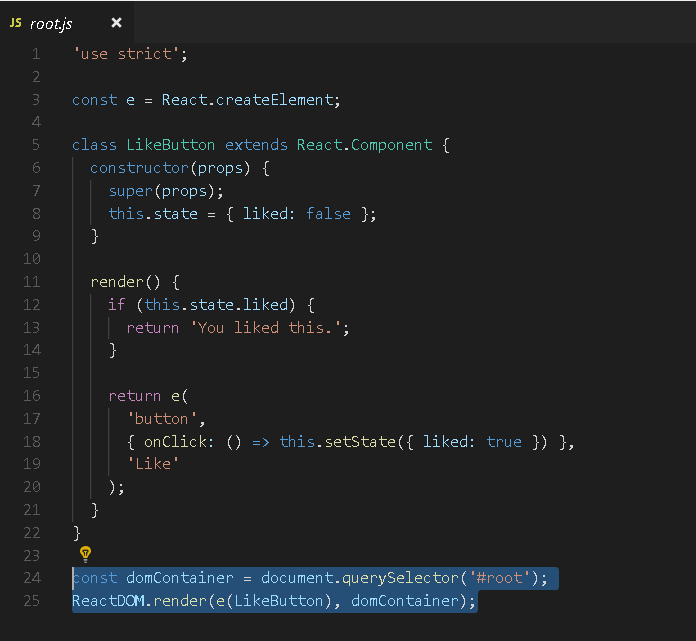


**The first two tags load React. The third one will load your component code.**



**Step 3: Create a React Component**

Create a file called like\_button.js next to your HTML page.

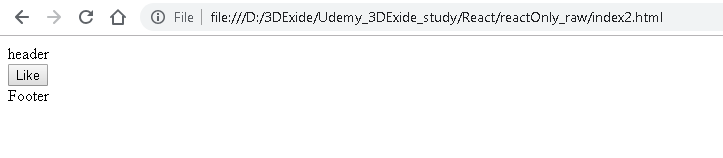


These two highlighted lines of code find the <div> we added to our HTML in the first step, and then display our “Like” button React component inside of it.

**That’s It!**

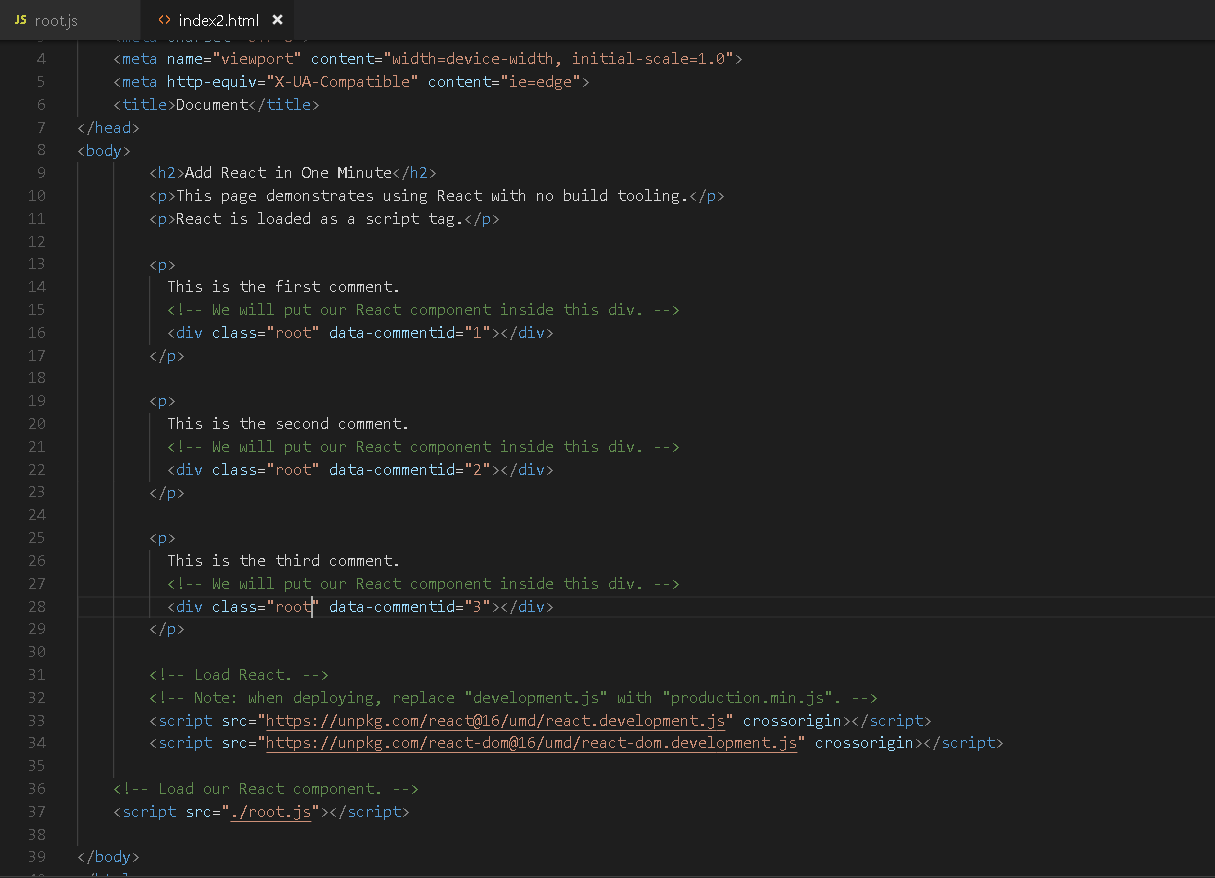
There is no step four. **You have just added the first React component to your website.**

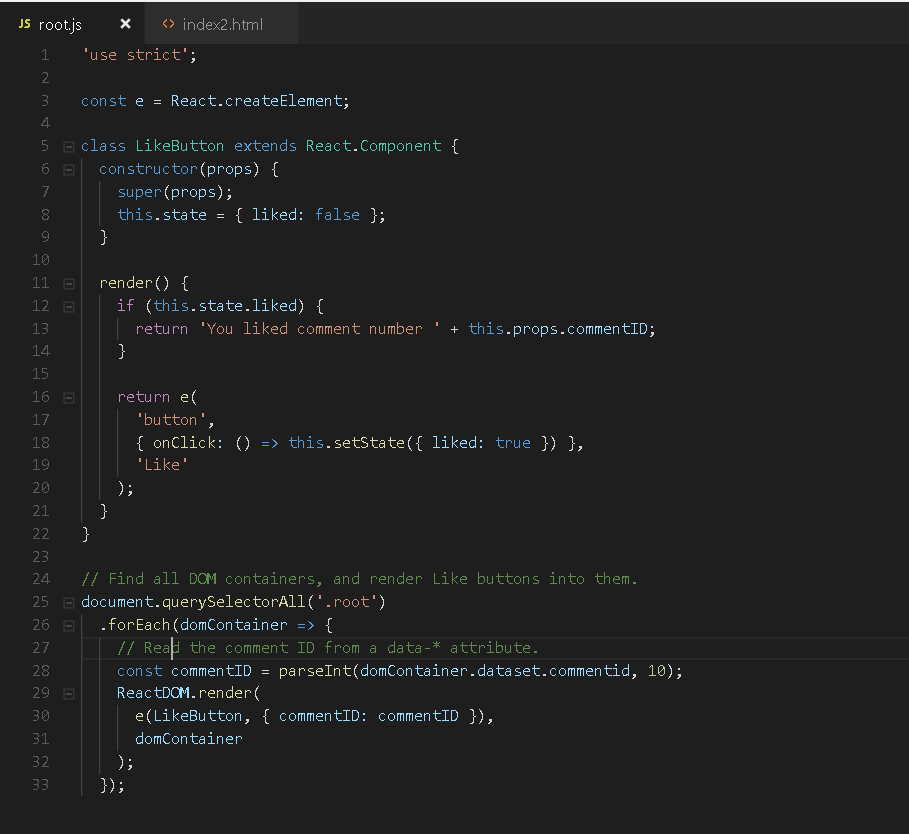
**Output –**

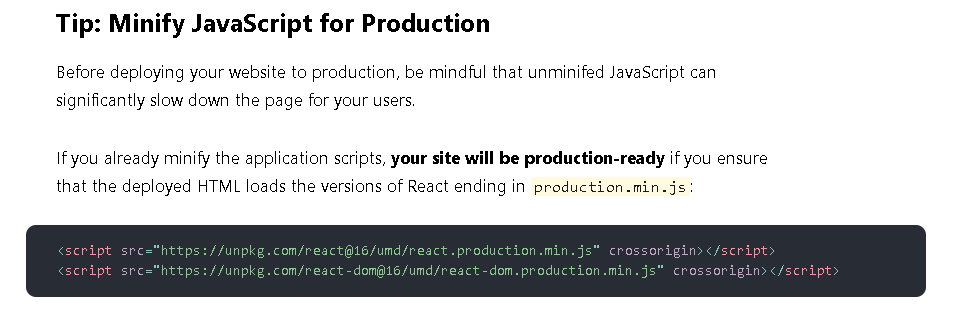


**Tip: Reuse a Component**

Commonly, you might want to display React components in multiple places on the HTML page. Here is an example that displays the “Like” button three times and passes some data to it:





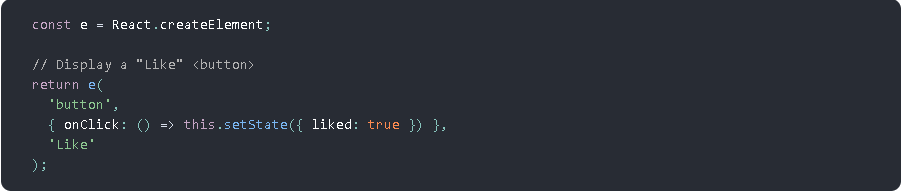


If you don’t have a minification step for your scripts, [here’s one way to set it up](https://gist.github.com/gaearon/42a2ffa41b8319948f9be4076286e1f3).

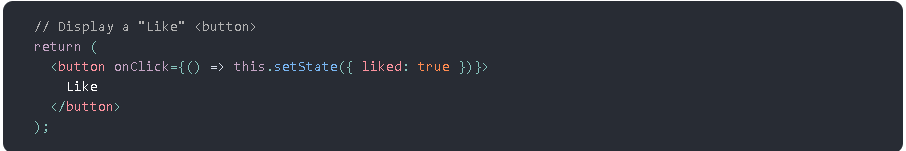


### Try react with JSX

In the examples above, we only relied on features that are natively supported by the browsers. This is why we used a JavaScript function call to tell React what to display



**However, React also offers an option to use**[JSX](https://reactjs.org/docs/introducing-jsx.html)**instead**



These two code snippets are equivalent. While **JSX is**[completely optional](https://reactjs.org/docs/react-without-jsx.html), many people find it helpful for writing UI code — both with React and with other libraries.

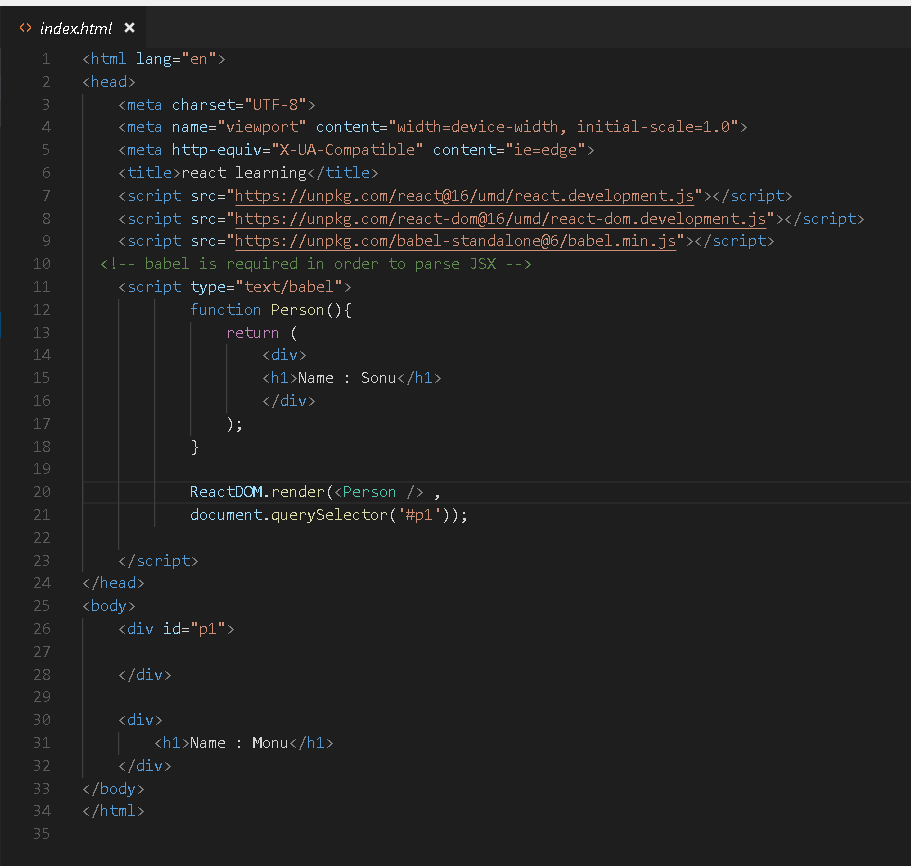
**Quickly Try JSX**

**The quickest way to try JSX in your project is to add this <script> tag to your page:**

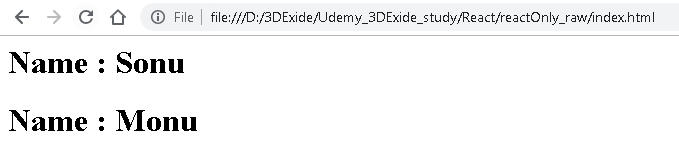
<script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>

**Now you can use JSX in any <script> tag by adding type="text/babel" attribute to it.**

**Note :** here babel is used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments.



**Output –**



This approach is fine for learning and creating simple demos. However, it makes your website slow and **isn’t suitable for production**. When you’re ready to move forward, remove this new <script> tag and the type="text/babel" attributes you’ve added. Instead, in the next section you will set up a JSX preprocessor to convert all your <script> tags automatically.

Need to try -doubt

**Add JSX to a Project**

Adding JSX to a project doesn’t require complicated tools like a bundler or a development server. Essentially, adding JSX **is a lot like adding a CSS preprocessor.** The only requirement is to have [Node.js](https://nodejs.org/) installed on your computer.

Go to your project folder in the terminal, and paste these two commands:

1. **Step 1:** Run npm init -y (if it fails, [here’s a fix](https://gist.github.com/gaearon/246f6380610e262f8a648e3e51cad40d))
2. **Step 2:** Run npm install babel-cli@6 babel-preset-react-app@3

**Tip**

We’re **using npm here only to install the JSX preprocessor;** you won’t need it for anything else. Both React and the application code can stay as <script> tags with no changes.

Congratulations! You just added a **production-ready JSX setup** to your project.

**Run JSX Preprocessor**

Create a folder called src and run this terminal command:

npx babel --watch src --out-dir . --presets react-app/prod

**Note**

npx is not a typo — it’s a [package runner tool that comes with npm 5.2+](https://medium.com/@maybekatz/introducing-npx-an-npm-package-runner-55f7d4bd282b).

If you see an error message saying “You have mistakenly installed the babel package”, you might have missed [the previous step](https://reactjs.org/docs/add-react-to-a-website.html#add-jsx-to-a-project). Perform it in the same folder, and then try again.

Don’t wait for it to finish — this command starts an automated watcher for JSX.

If you now create a file called src/like\_button.js with this [JSX starter code](https://cdn.rawgit.com/gaearon/c8e112dc74ac44aac4f673f2c39d19d1/raw/09b951c86c1bf1116af741fa4664511f2f179f0a/like_button.js), the watcher will create a preprocessed like\_button.js with the plain JavaScript code suitable for the browser. When you edit the source file with JSX, the transform will re-run automatically.

As a bonus, this also lets you use modern JavaScript syntax features like classes without worrying about breaking older browsers. The tool we just used is called Babel, and you can learn more about it from [its documentation](http://babeljs.io/docs/en/babel-cli/).

If you notice that you’re getting comfortable with build tools and want them to do more for you, [the next section](https://reactjs.org/docs/create-a-new-react-app.html) describes some of the most popular and approachable toolchains. If not — those script tags will do just fine!

### Create a New React App

Use an integrated toolchain for the best user and developer experience.

This page describes a few popular React toolchains which help with tasks like:

* Scaling to many files and components.
* Using third-party libraries from npm.
* Detecting common mistakes early.
* Live-editing CSS and JS in development.
* Optimizing the output for production.

The toolchains recommended on this page **don’t require configuration to get started**.

#### You Might Not Need a Toolchain

If you don’t experience the problems described above or don’t feel comfortable using JavaScript tools yet, consider [adding React as a plain <script> tag on an HTML page](https://reactjs.org/docs/add-react-to-a-website.html), optionally [with JSX](https://reactjs.org/docs/add-react-to-a-website.html#optional-try-react-with-jsx).

This is also **the easiest way to integrate React into an existing website.** You can always add a larger toolchain if you find it helpful!

#### **Recommended Toolchains**

The React team primarily recommends these solutions:

* If you’re **learning React** or **creating a new**[single-page](https://reactjs.org/docs/glossary.html#single-page-application)**app,** use [Create React App](https://reactjs.org/docs/create-a-new-react-app.html#create-react-app).
* If you’re building a **server-rendered website with Node.js,** try [Next.js](https://reactjs.org/docs/create-a-new-react-app.html#nextjs).
* If you’re building a **static content-oriented website,** try [Gatsby](https://reactjs.org/docs/create-a-new-react-app.html#gatsby).
* If you’re building a **component library** or **integrating with an existing codebase**, try [More Flexible Toolchains](https://reactjs.org/docs/create-a-new-react-app.html#more-flexible-toolchains).

##### **Create React App**

[Create React App](http://github.com/facebookincubator/create-react-app) is a comfortable environment for **learning React**, and is the best way to start building **a new**[single-page](https://reactjs.org/docs/glossary.html#single-page-application)**application** in React.

It sets up your development environment so that you can use the latest JavaScript features, provides a nice developer experience, and optimizes your app for production. You’ll need to have Node >= 6 and npm >= 5.2 on your machine. To create a project, run:

npx create-react-app my-app

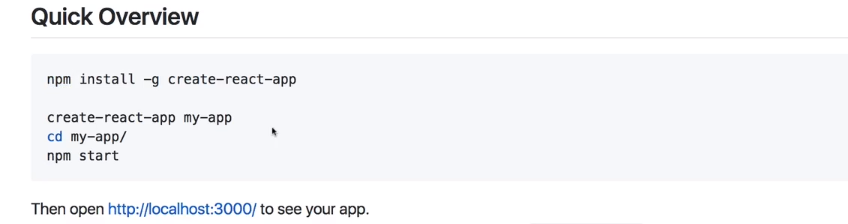
cd my-app

npm start

**Note**

npx on the first line is not a typo — it’s a [package runner tool that comes with npm 5.2+](https://medium.com/@maybekatz/introducing-npx-an-npm-package-runner-55f7d4bd282b).

**another way –**



Create React App doesn’t handle backend logic or databases; it just creates a frontend build pipeline, so you can use it with any backend you want. Under the hood, it uses [Babel](http://babeljs.io/) and [webpack](https://webpack.js.org/), but you don’t need to know anything about them.

When you’re ready to deploy to production, running npm run build will create an optimized build of your app in the build folder. You can learn more about Create React App [from its README](https://github.com/facebookincubator/create-react-app#create-react-app-) and the [User Guide](https://github.com/facebookincubator/create-react-app/blob/master/packages/react-scripts/template/README.md#table-of-contents).

##### **Next.js**

[Next.js](https://nextjs.org/) is a popular and lightweight framework for **static and server‑rendered applications**built with React. It includes **styling and routing solutions** out of the box, and assumes that you’re using [Node.js](https://nodejs.org/) as the server environment.

Learn Next.js from [its official guide](https://nextjs.org/learn/).

##### **Gatsby**

[Gatsby](https://www.gatsbyjs.org/) is the best way to create **static websites** with React. It lets you use React components, but outputs pre-rendered HTML and CSS to guarantee the fastest load time.

Learn Gatsby from [its official guide](https://www.gatsbyjs.org/docs/) and a [gallery of starter kits](https://www.gatsbyjs.org/docs/gatsby-starters/).

##### **More Flexible Toolchains**

The following toolchains offer more flexiblity and choice. We recommend them to more experienced users:

* [Neutrino](https://neutrinojs.org/) combines the power of [webpack](https://webpack.js.org/) with the simplicity of presets, and includes a preset for [React apps](https://neutrinojs.org/packages/react/) and [React components](https://neutrinojs.org/packages/react-components/).
* [nwb](https://github.com/insin/nwb) is particularly great for [publishing React components for npm](https://github.com/insin/nwb/blob/master/docs/guides/ReactComponents.md#developing-react-components-and-libraries-with-nwb). It [can be used](https://github.com/insin/nwb/blob/master/docs/guides/ReactApps.md#developing-react-apps-with-nwb) for creating React apps, too.
* [Parcel](https://parceljs.org/) is a fast, zero configuration web application bundler that [works with React](https://parceljs.org/recipes.html#react).
* [Razzle](https://github.com/jaredpalmer/razzle) is a server-rendering framework that doesn’t require any configuration, but offers more flexibility than Next.js.

#### **Creating a Toolchain from Scratch**

A JavaScript build toolchain typically consists of:

* A **package manager**, such as [Yarn](https://yarnpkg.com/) or [npm](https://www.npmjs.com/). It lets you take advantage of a vast ecosystem of third-party packages, and easily install or update them.
* A **bundler**, such as [webpack](https://webpack.js.org/) or [Parcel](https://parceljs.org/). It lets you write modular code and bundle it together into small packages to optimize load time.
* A **compiler** such as [Babel](http://babeljs.io/). It lets you write modern JavaScript code that still works in older browsers.

If you prefer to set up your own JavaScript toolchain from scratch, [check out this guide](https://blog.usejournal.com/creating-a-react-app-from-scratch-f3c693b84658) that re-creates some of the Create React App functionality.

Don’t forget to ensure your custom toolchain [is correctly set up for production](https://reactjs.org/docs/optimizing-performance.html#use-the-production-build).

# CDN Links

Both React and ReactDOM are available over a CDN.

<script crossorigin src="https://unpkg.com/react@16/umd/react.development.js"></script>

<script crossorigin src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>

The versions above are only meant for development, and are not suitable for production. Minified and optimized production versions of React are available at:

<script crossorigin src="https://unpkg.com/react@16/umd/react.production.min.js"></script>

<script crossorigin src="https://unpkg.com/react-dom@16/umd/react-dom.production.min.js"></script>

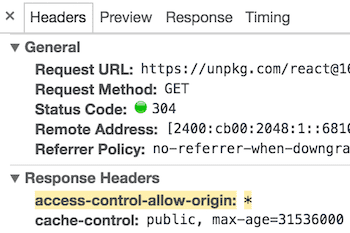
To load a specific version of react and react-dom, replace 16 with the version number.

### Why the crossorigin Attribute?

If you serve React from a CDN, we recommend to keep the [crossorigin](https://developer.mozilla.org/en-US/docs/Web/HTML/CORS_settings_attributes) attribute set:

<script crossorigin src="..."></script>

We also recommend to verify that the CDN you are using sets the Access-Control-Allow-Origin: \* HTTP header:

[](https://reactjs.org/static/cdn-cors-header-89baed0a6540f29e954065ce04661048-dd807.png)

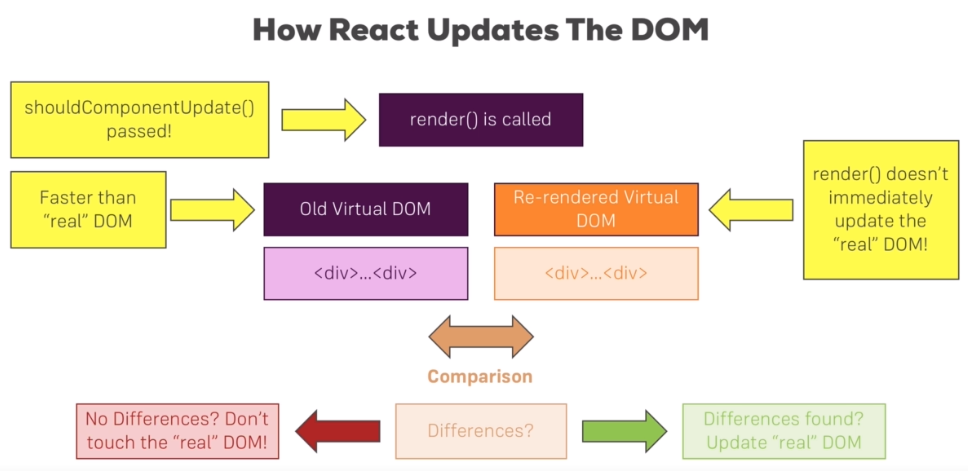
This enables a better [error handling experience](https://reactjs.org/blog/2017/07/26/error-handling-in-react-16.html) in React 16 and later.

# What is Virtual DOM? How reacts Updates the DOM ? What is reconciliation?

The Virtual DOM (VDOM) is an in-memory representation of Real DOM. The representation of a UI is kept in memory and synced with the "real" DOM. It's a step that happens between the render function being called and the displaying of elements on the screen. This entire process is called reconciliation.

The virtual DOM (VDOM) is a programming concept where an ideal, or “virtual”, representation of a UI is kept in memory and synced with the “real” DOM by a library such as ReactDOM.

## **How reacts Updates the DOM**





it compares virtual DOMs

It has an old virtual DOM and a re-rendered or a future virtual DOM. React takes this virtual DOM approach because it's faster than the real DOM.

Now a virtual DOM simply is a DOM representation in Javascript.

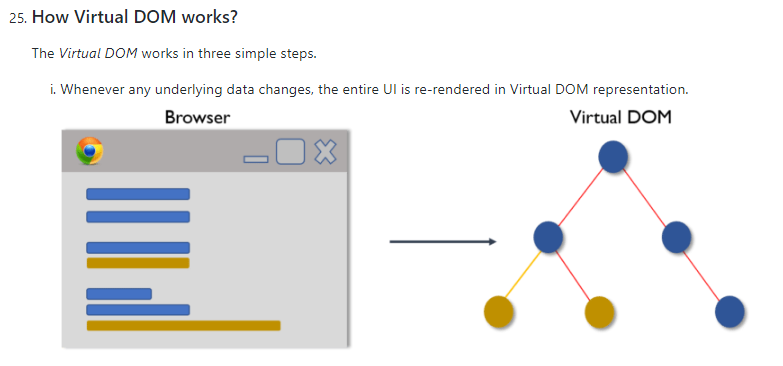
You can of course represent all HTML and therefore DOM elements and objects in pure Javascript, so without rendering anything to the browser and this is what happens here and

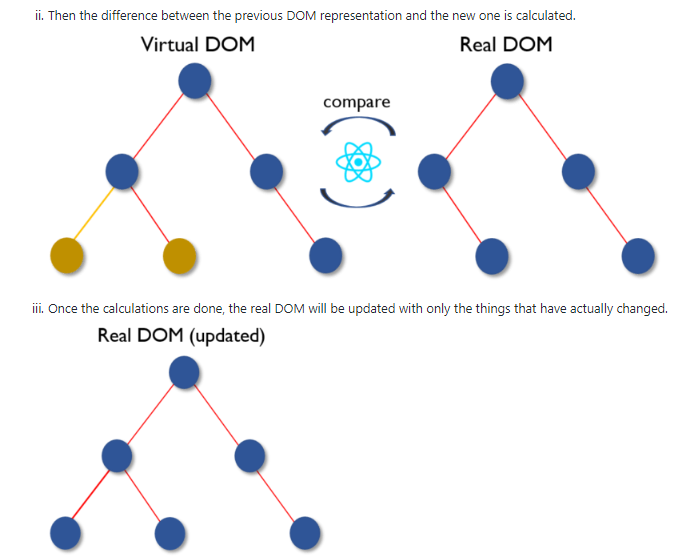
**React basically keeps two copies of the DOM. It has the old virtual DOM and then the re-rendered one, the re-rendered one is the one which gets created when the render method is called.**

**React makes a comparison. It compares the old virtual DOM to the new one and it checks if there are any differences. If it can detect differences, it reaches out to the real DOM and updates it and**

**even then, it doesn't re-render the real DOM entirely, it only changes it in the places where differences were detected,**

**for example if a button text changed, it will only update that text and not re-render the whole button, leave alone the whole DOM.**

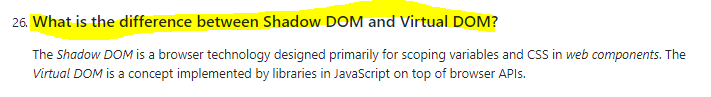




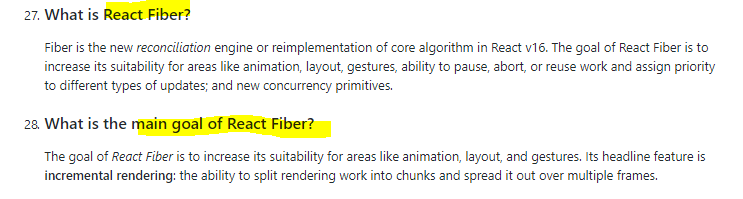
## **What is reconciliation?**

When a component's props or state change, React decides whether an actual DOM update is necessary by comparing the newly returned element with the previously rendered one. When they are not equal, React will update the DOM. This process is called reconciliation.

virtual DOM like the real DOM is too a simple js object.React always maintains two virtual DOMs one with the initial state and the other with the updated state of UI. And further the real DOM is updated based on the changes in the two. For this comparison React uses the **diffing algorithm** instead of state of the art algorithms .



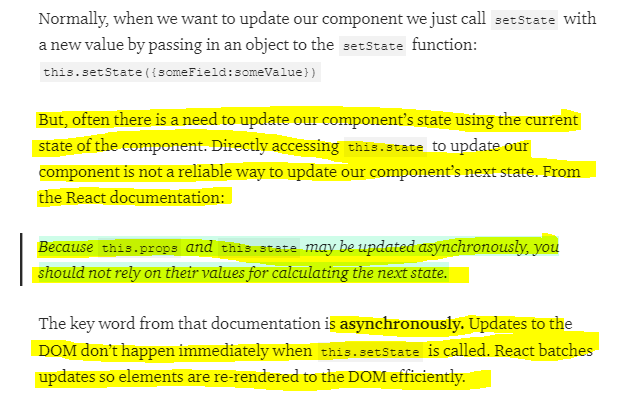
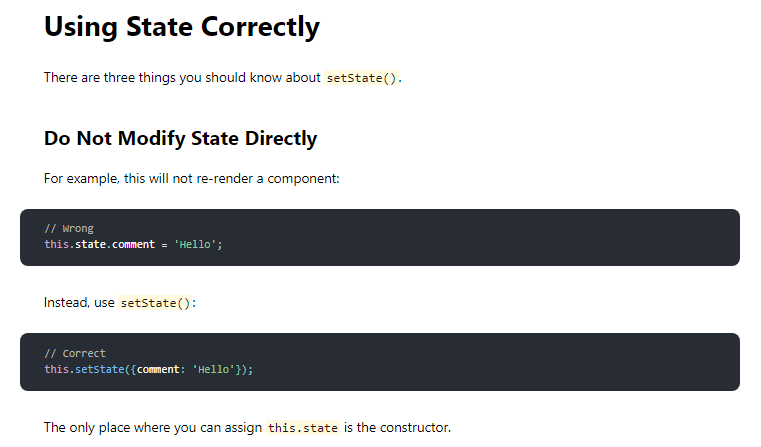
## **React Fibre**

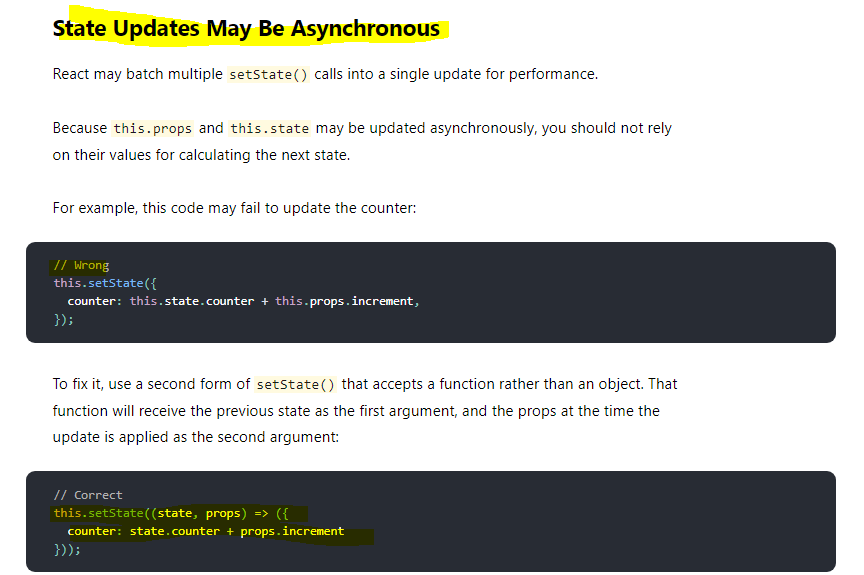


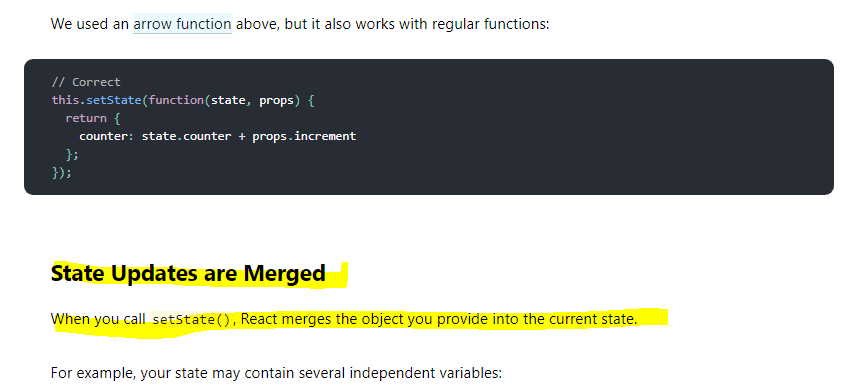
# **setState - Setting the State**

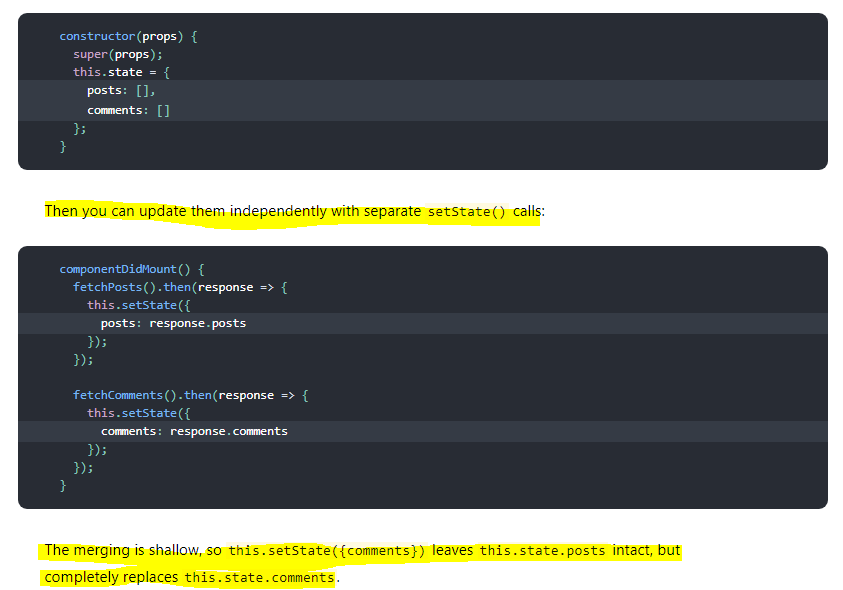
Always use the setState() method to change the state object, it will ensure that the component knows its been updated and calls the render() method (and all the other lifecycle methods).

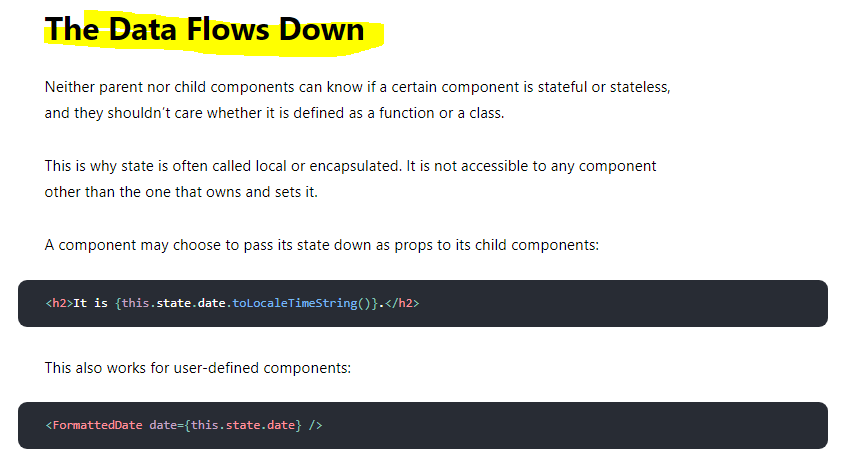


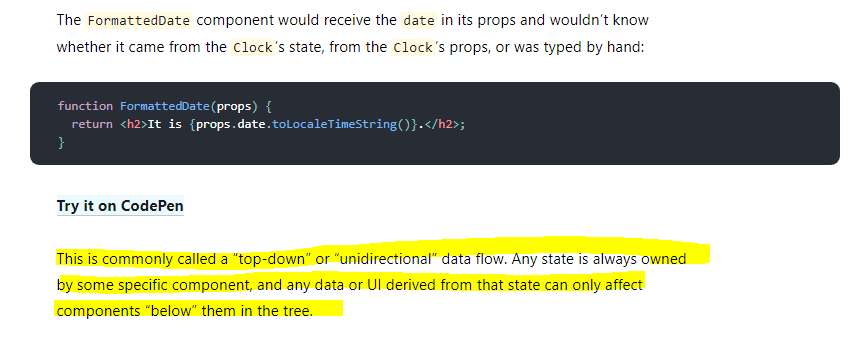
**Setting state correctly** 

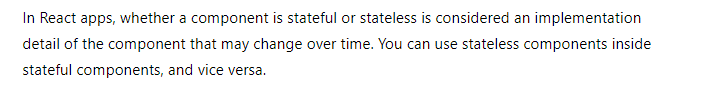


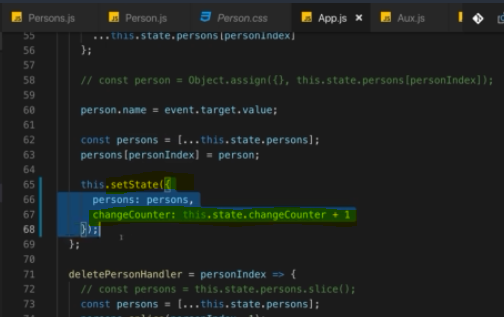












We use setState to update their state.

Now the good news is we're using set state correctly here, bad news is you can use it incorrectly but I will show you how and how to avoid it.

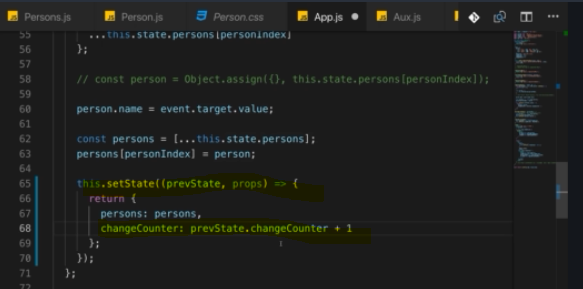
now of course this above code always depends on the old state, right?

Behind the scenes, set state does not immediately triggeran update of the state of this component in a re-render cycle, instead it's basically scheduled by React and React will then perform the state update and the re-render cycle when it has the available resources to do that.

You call set state synchronously here but it's not guaranteed to execute and finish immediately and therefore, this state when used for a state update is not guaranteed to be the latest state or the previous state on which you depend, it could be an older state.

**Set state does not only take a Javascript object, it also works when you pass in a function, so you can use either syntax.**

optional syntax where you actually receive two arguments and I'm using an anonymous arrow function here, where the first argument is your old state and I'll name it prev state for previous state and the second argument are your current props in case you need those. Then you have your function body and in that function body, you should return this new state object, like this. Now of course there, you are now not referring to this state but you can refer to previous state change counter and here, React guarantees you that this will be the actual previous state as you would expect it for this set state update.

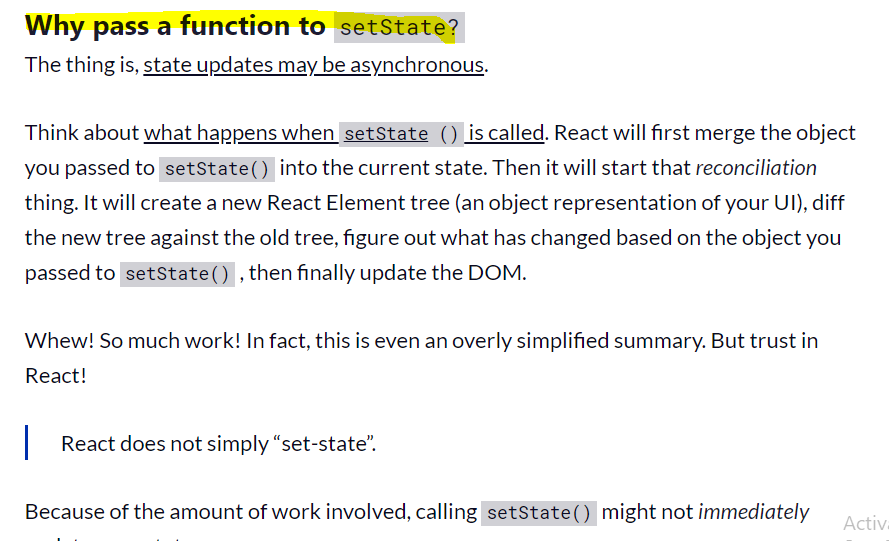


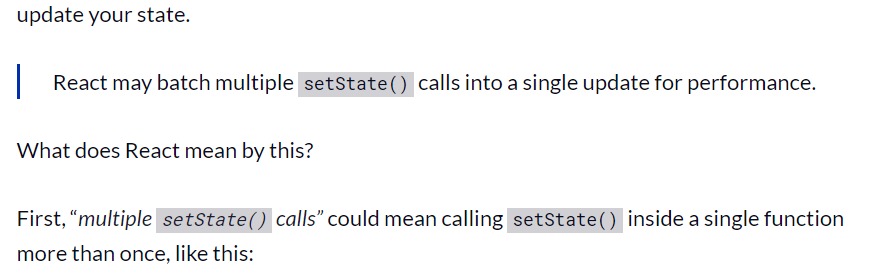
**this is the better way and the recommended way of updating the state when you're depending on old state. Super important to keep that in mind, it's an important pattern, not an optional solution but really the best practice for state updates that depend on the old state.**

**If you know you’re going to use setState to update your component and you know you’re going to need the current state or the current props of your component to calculate the next state, passing in a function as the first parameter of this.setState instead of an object is the recommended solution.**

## **Another very good explanation**

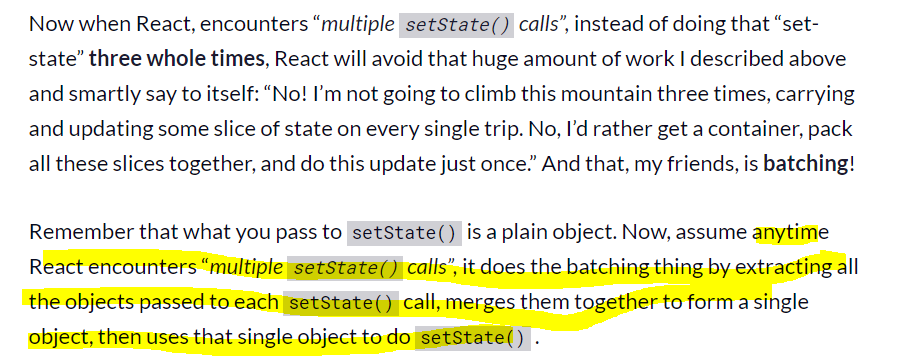
<https://www.freecodecamp.org/news/functional-setstate-is-the-future-of-react-374f30401b6b/>



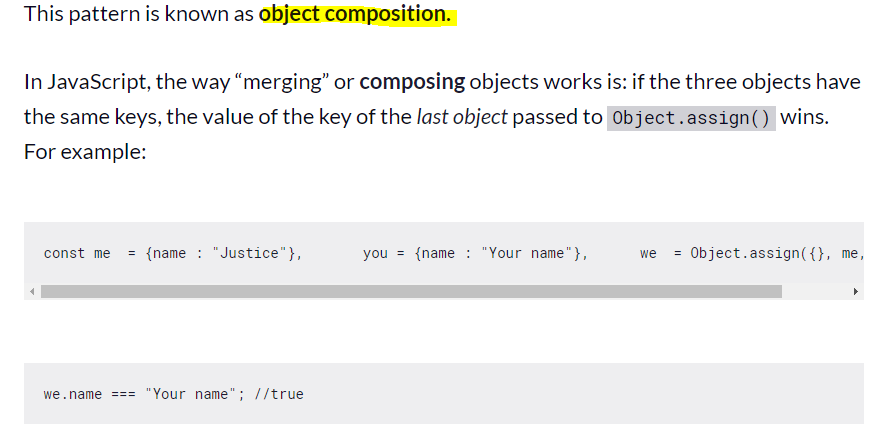


// multiple setState()

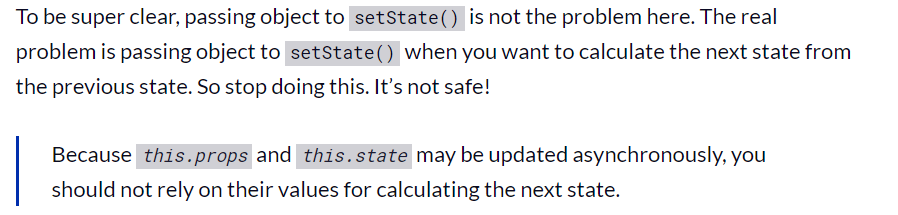
callsincreaseScoreBy3 () { this.setState({score : this.state.score + 1}); this.setState({score : this.state.score + 1}); this.setState({score : this.state.score + 1});}



const singleObject = Object.assign( {}, objectFromSetState1, objectFromSetState2, objectFromSetState3);

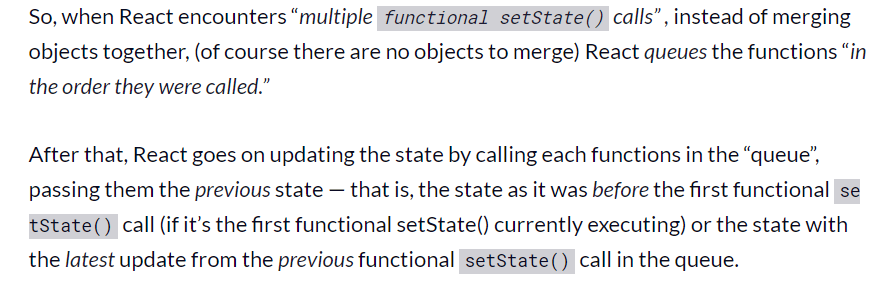


Thus, if you call setState() with an object multiple times — passing an object each time — React will **merge**. Or in other words, it will **compose** a new object out of the multiple objects we passed it. And if any of the objects contains the same key, the value of the key of the last object with same key is stored.



**When you do functional setState…**

**Updates will be queued and later executed in the order they were called.**



**// multiple functional setState call**

**increaseScoreBy3 () { this.setState( (state) => ({score : state.score + 1}) ), this.setState( (state) => ({score : state.score + 1}) ), this.setState( (state) => ({score : state.score + 1}) ) }}**

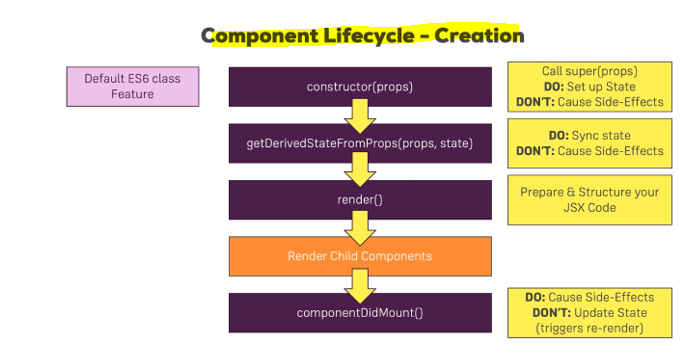
# **Component Lifecycle hooks/methods(only available in Class-based components)**

**it's only available in class-based components. For functional component handling lifecycle of component we have react hooks.**

We've got these methods which we can add to any class-based component and React will execute them for us and they will run at different points of time and we can do different things with them.

for example

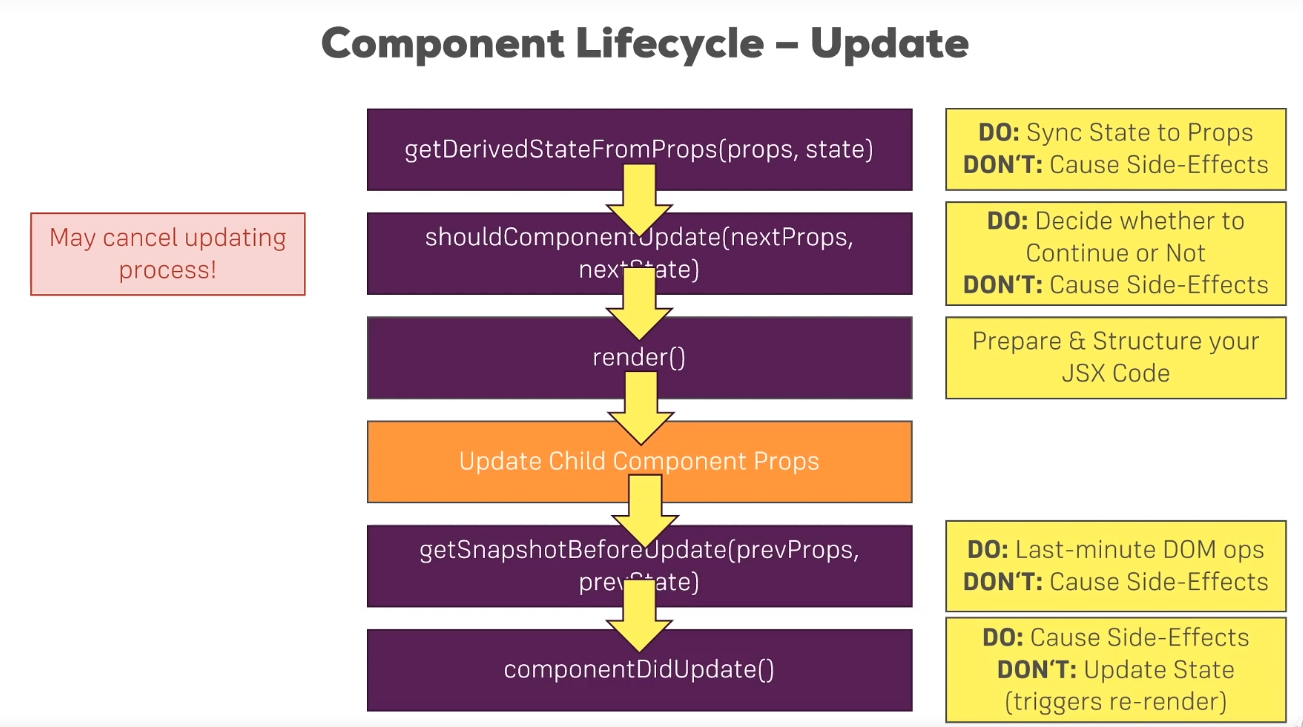
fetch data from the web or to do some cleanup work before a component is removed from the DOM

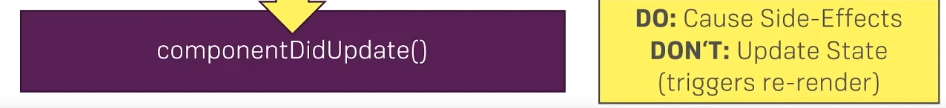




**here, you can cause side effects. That is a typical hook you would use for making an HTTP request to get new data from the web.**

**What you shouldn't do in here is update the state, so don't call set state in here unless it's in, let's say the then block of a promise after you sent an HTTP request but don't call set state in here synchronously. So** **you can definitely set up some code that executes in the future which then updates the state, for example when the response from the server is back but don't do it right away when componentDidMount runs that you immediately call set state because that will trigger a re-render cycle and that is bad for performance.**



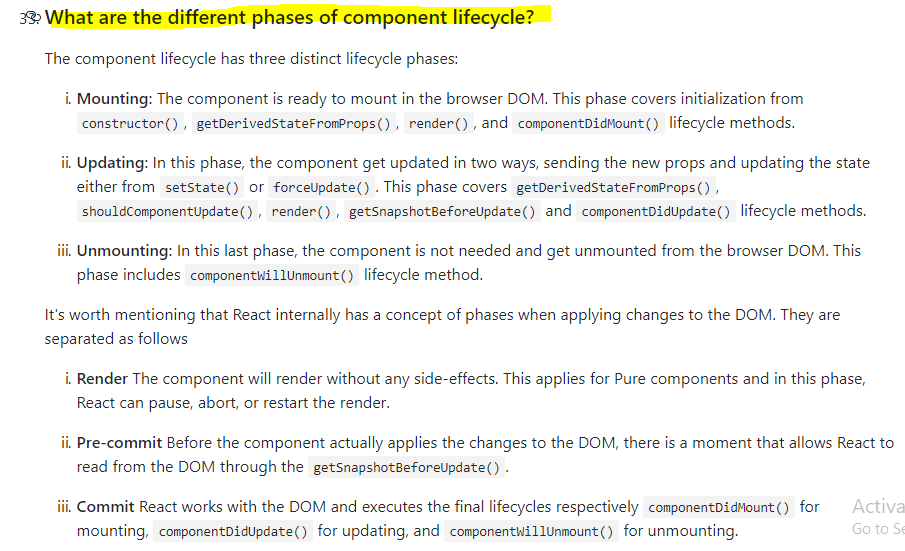


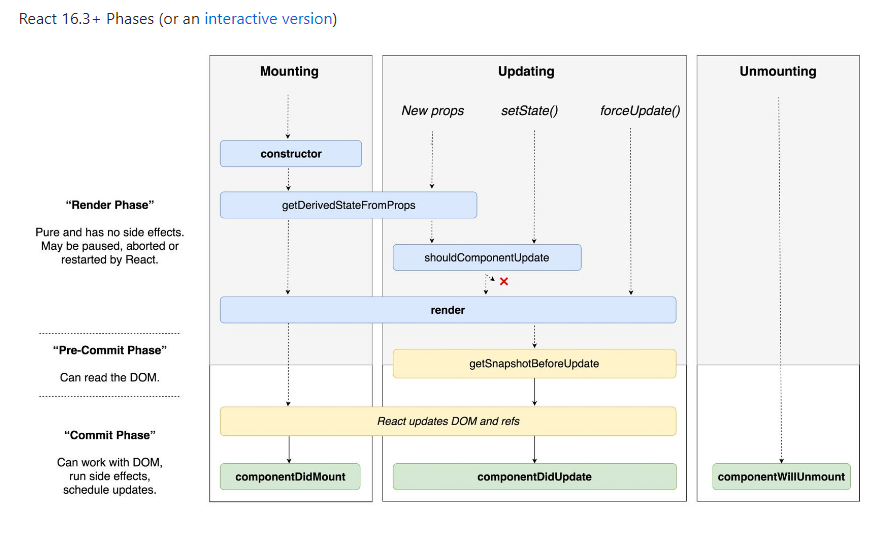
once we're done with the update, componentDidUpdate is called. A lifecycle hook that signals that you are now done with the updating, that the render method has been executed

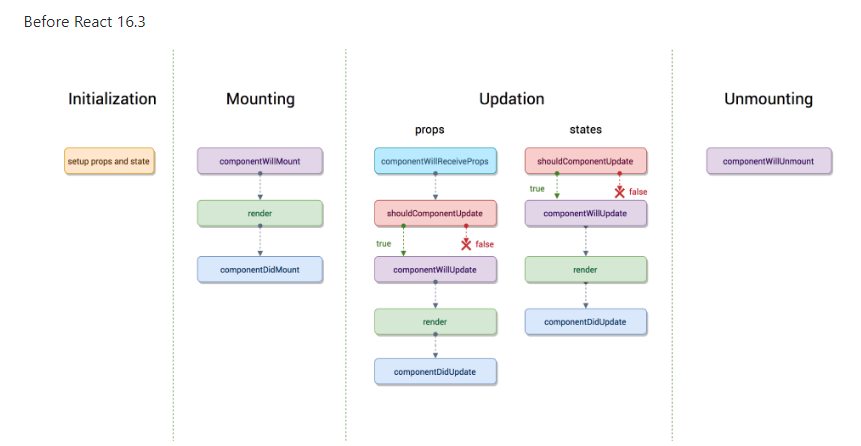
and here you can now cause side effects, so here you could now make an HTTP request, though you'll have to watch out to not enter an infinite loop here

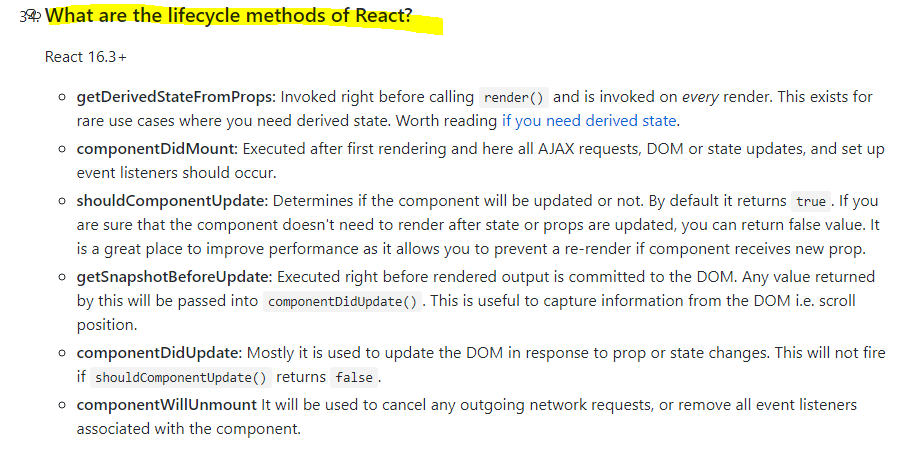
if you make an HTTP request and you get back a response and you then update your component and then this cycle starts .this is a typical problem you'll be facing. Now what you shouldn't do here outside of the, let's say then block of a promise of an HTTP request is updating the state with set state.

It's fine to do it as a result of some async task you're kicking off here but you should not call it synchronously in componentDidUpdate because that will simply lead to an unnecessary re-render cycle.





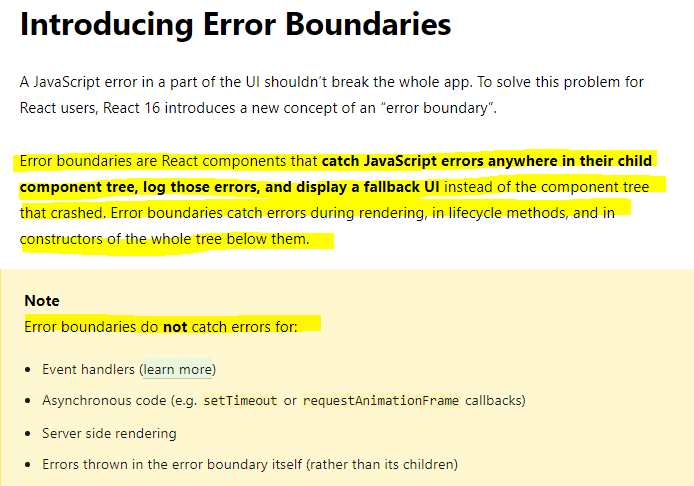






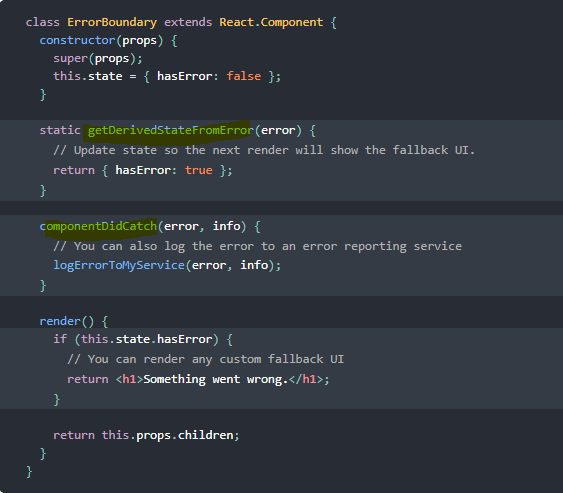
# **Error Boundaries**

In the past, JavaScript errors inside components used to corrupt React’s internal state and cause it to [emit](https://github.com/facebook/react/issues/4026) [cryptic](https://github.com/facebook/react/issues/6895) [errors](https://github.com/facebook/react/issues/8579) on next renders. These errors were always caused by an earlier error in the application code, but React did not provide a way to handle them gracefully in components, and could not recover from them.



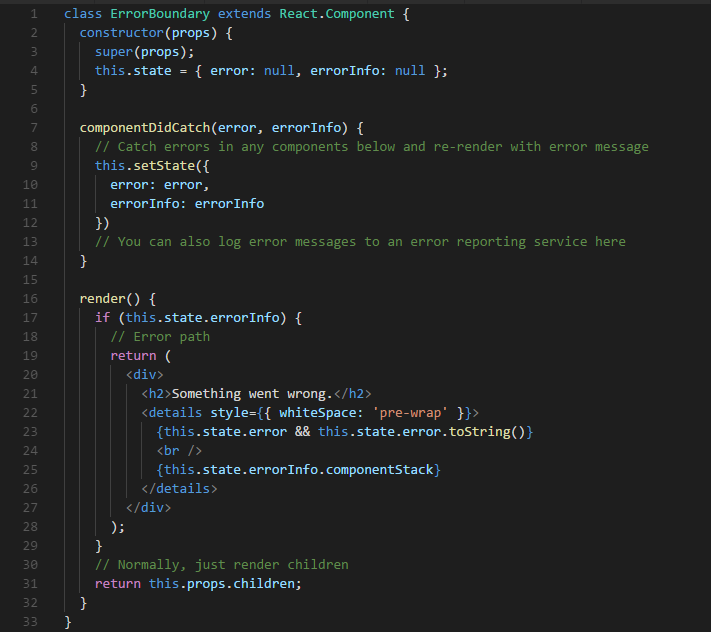
**A class component becomes an error boundary if it defines either (or both) of the lifecycle methods**[**static getDerivedStateFromError()**](https://reactjs.org/docs/react-component.html#static-getderivedstatefromerror)

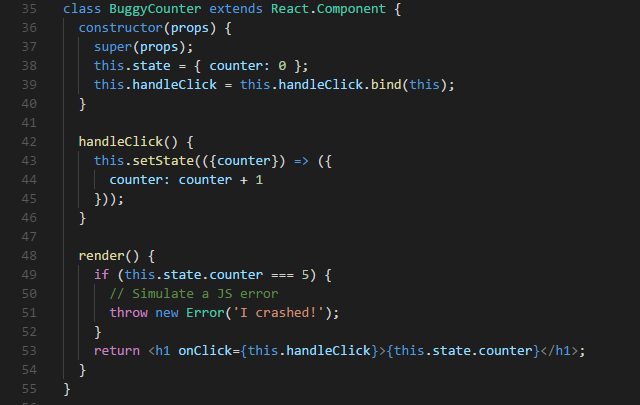
**or [componentDidCatch()](https://reactjs.org/docs/react-component.html" \l "componentdidcatch). Use static getDerivedStateFromError() to render a fallback UI after an error has been thrown. Use componentDidCatch() to log error information.**

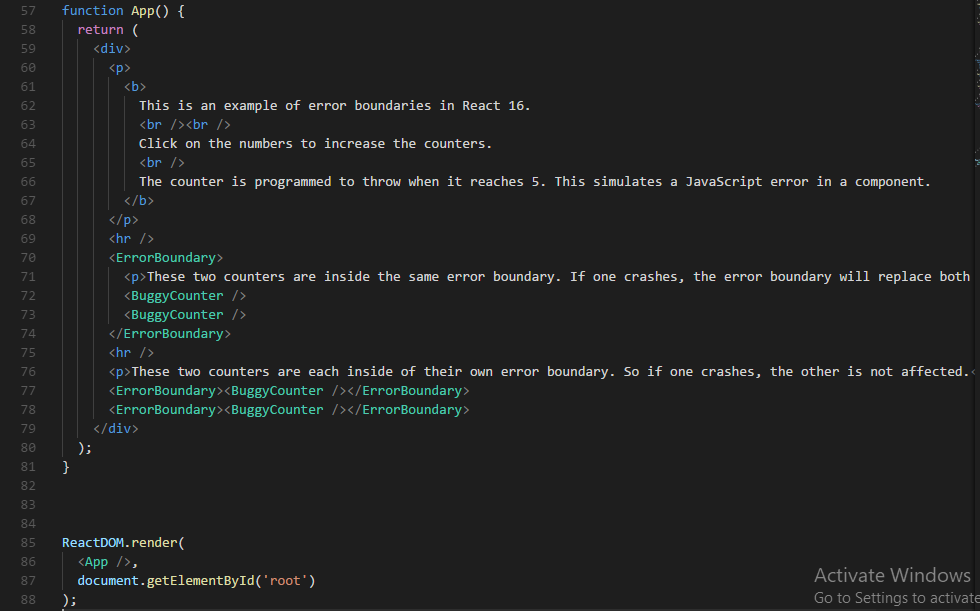


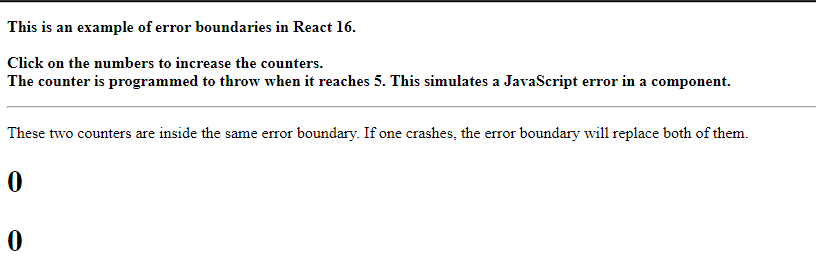


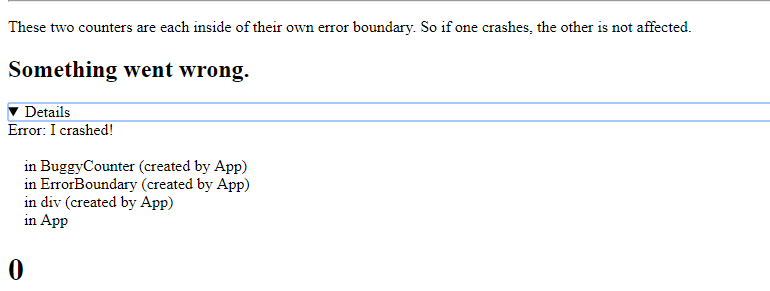
## **Example**

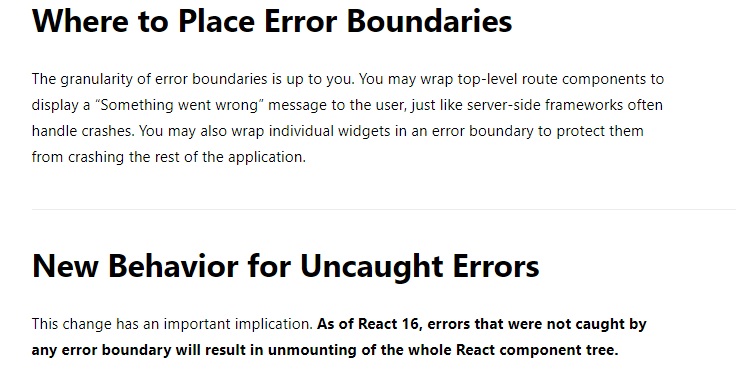


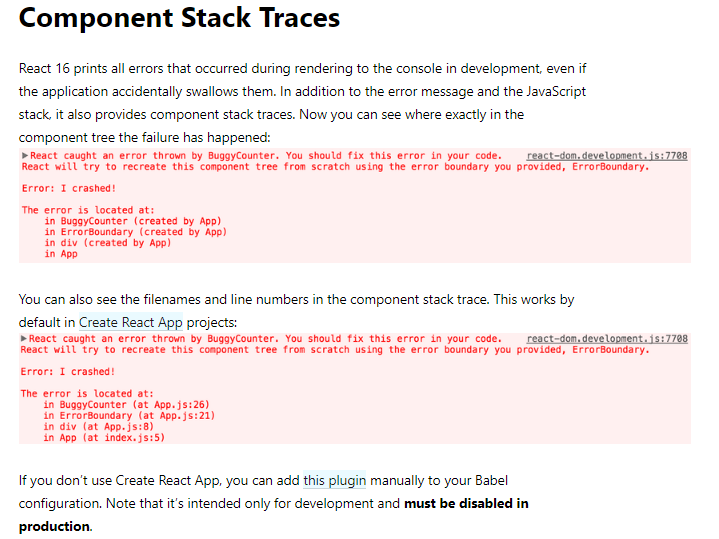


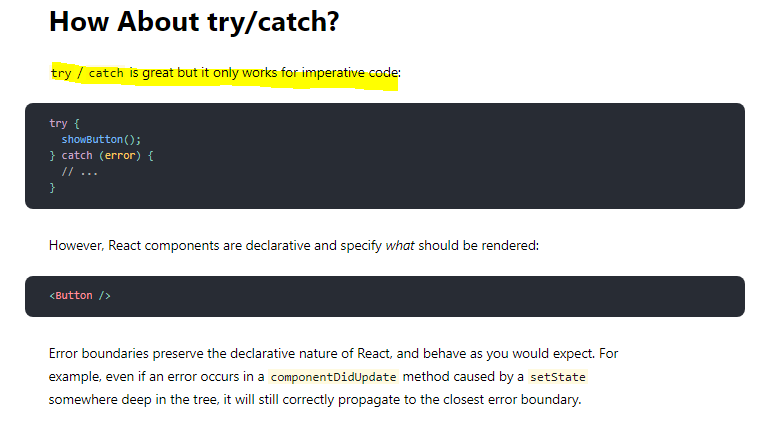


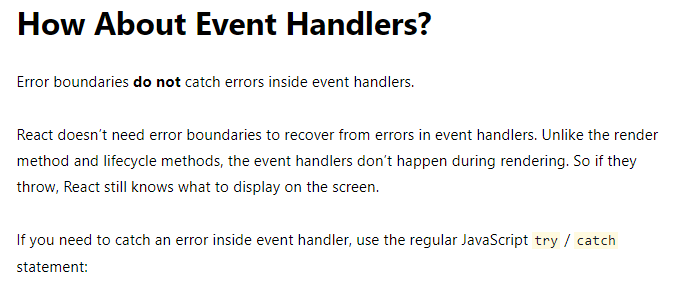
***Output –*** 













# Context

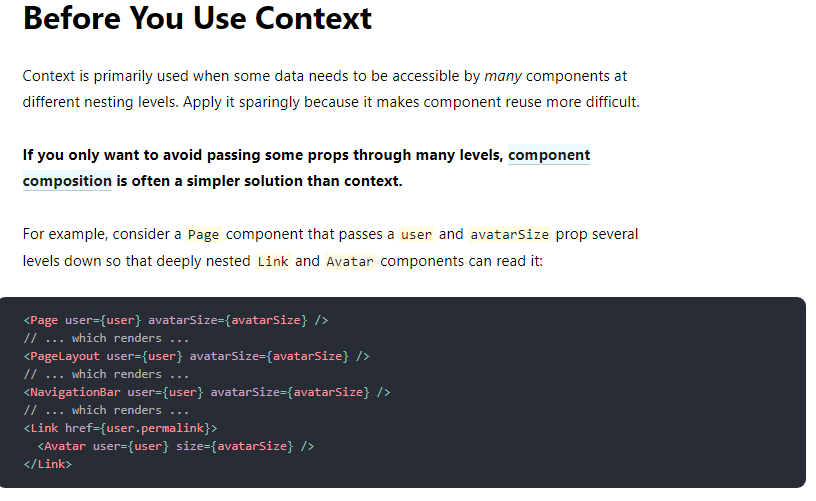
**Context was introduced by React and it helps us handle cases like this, where you need certain data, certain state in multiple components and you don't want to pass that state across multiple layers of components just to get it from component A at the top to component D at the very bottom when the components B, C in between don't really care about it.**

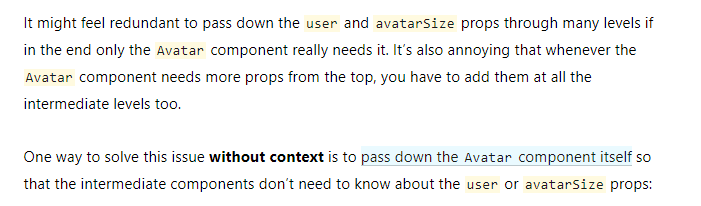
**Context provides a way to pass data through the component tree without having to pass props down manually at every level.**

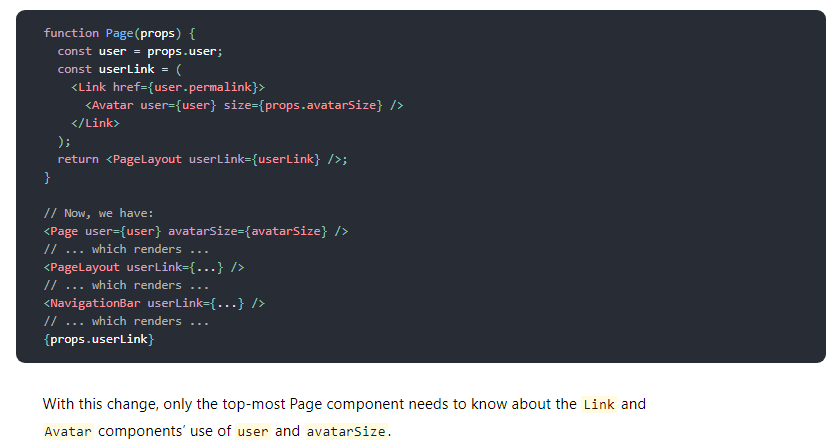
In a typical React application, data is passed top-down (parent to child) via props, but this can be cumbersome for certain types of props (e.g. locale preference, UI theme) that are required by many components within an application. Context provides a way to share values like these between components without having to explicitly pass a prop through every level of the tree.

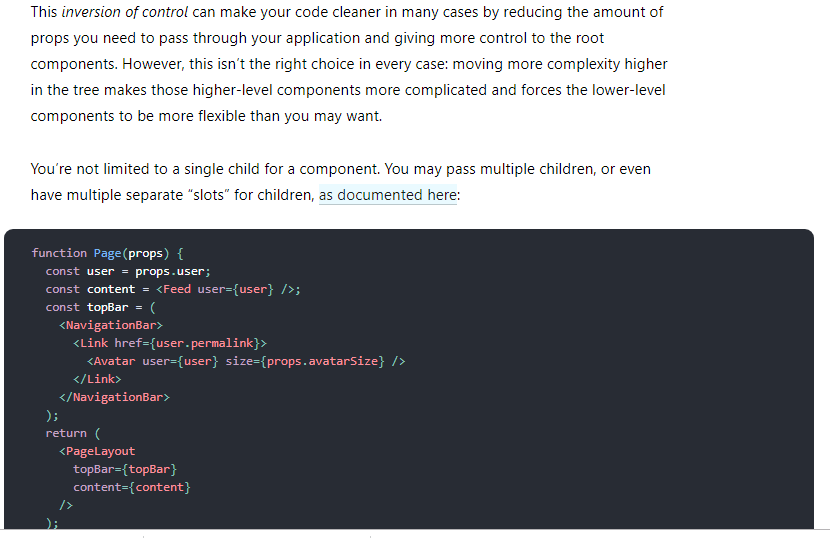
**When to Use Context**

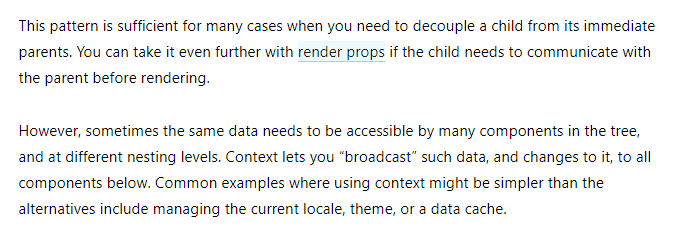
Context is designed to share data that can be considered “global” for a tree of React components, such as the current authenticated user, theme, or preferred language.

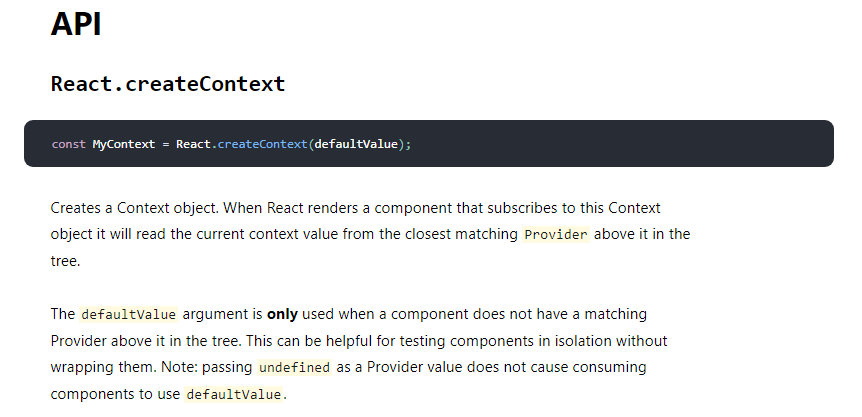


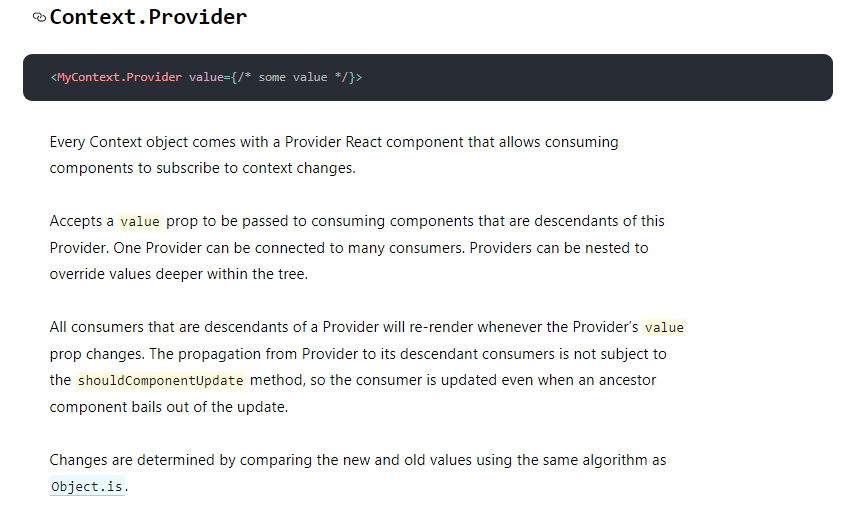


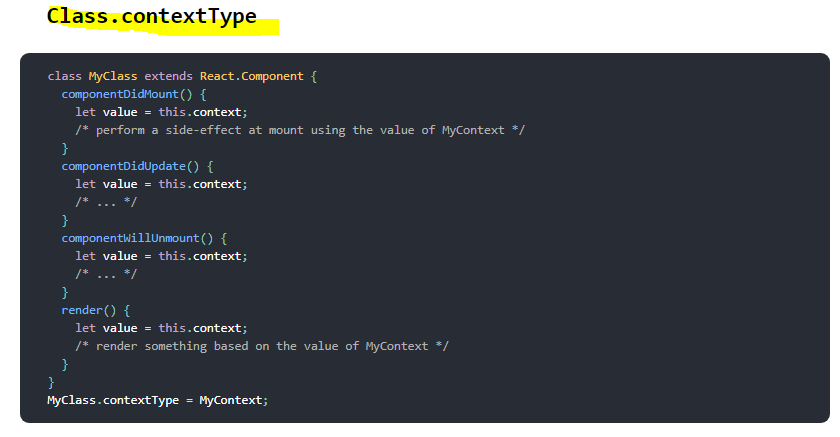


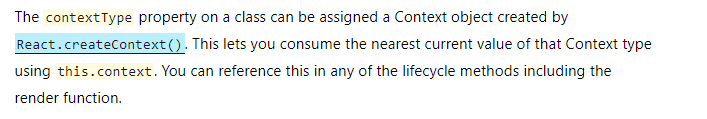


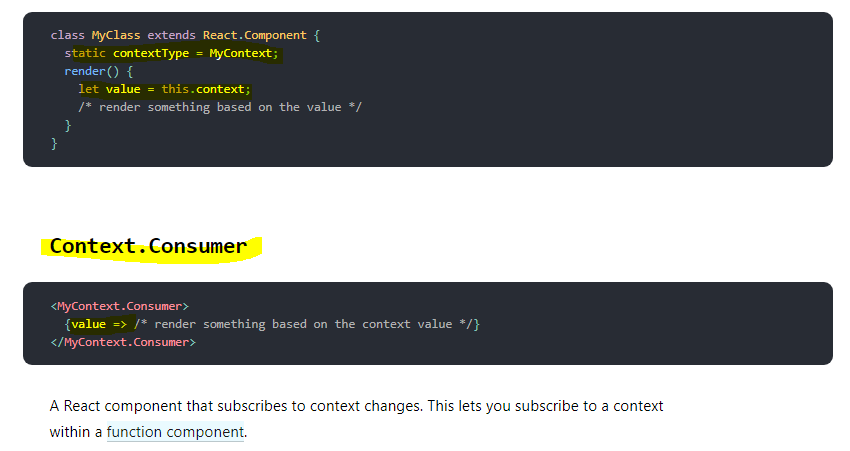


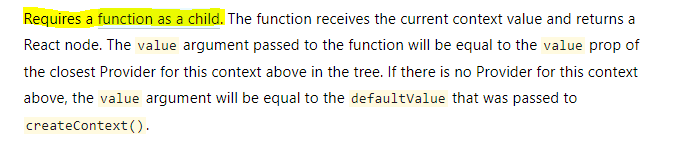


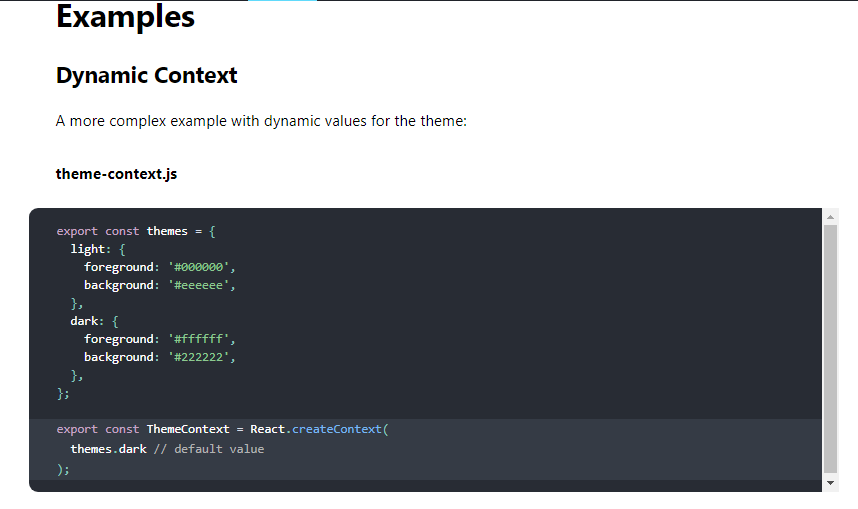






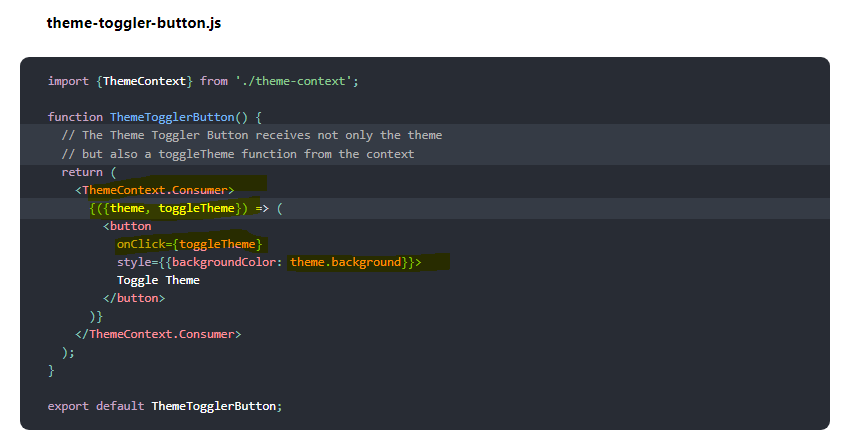


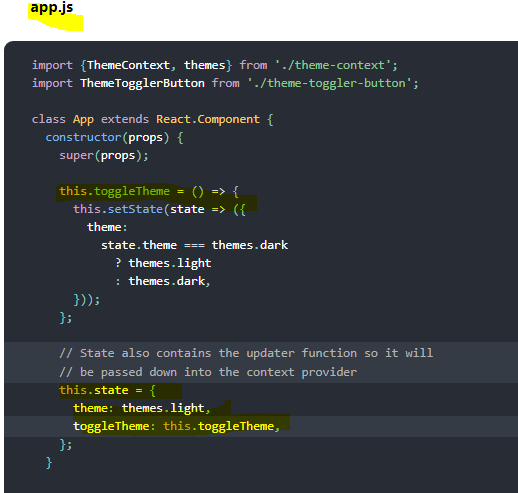


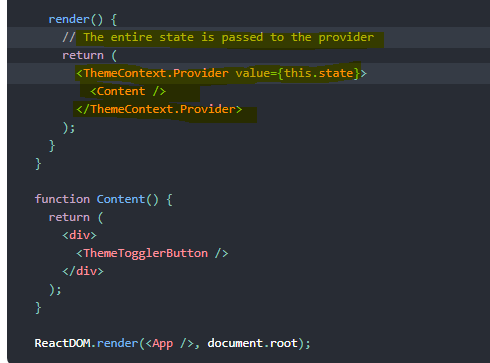






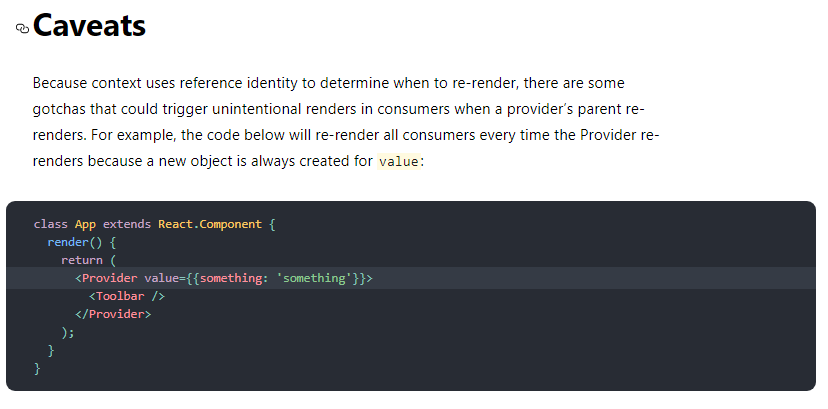






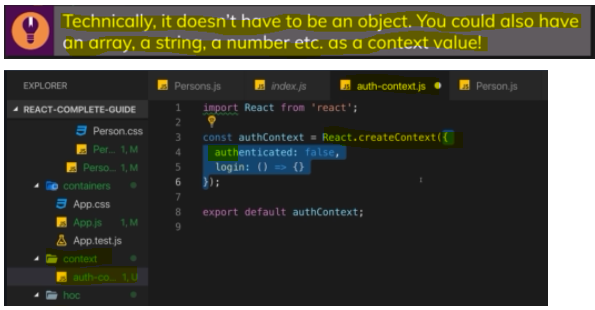


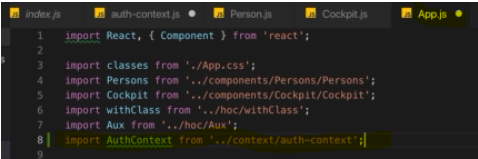




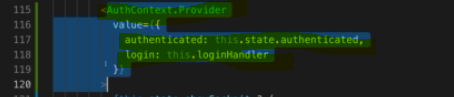


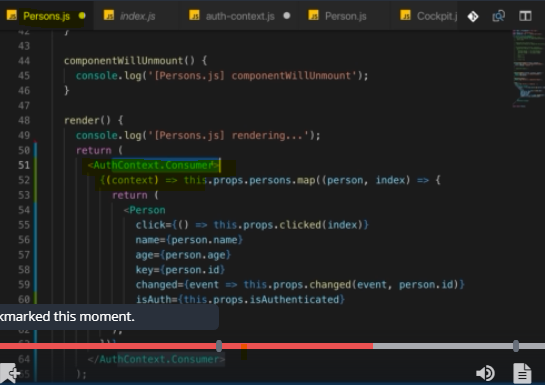
## **Example –**

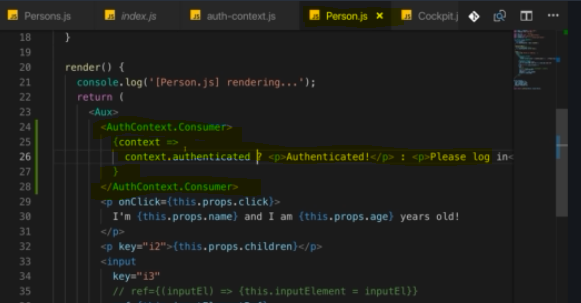


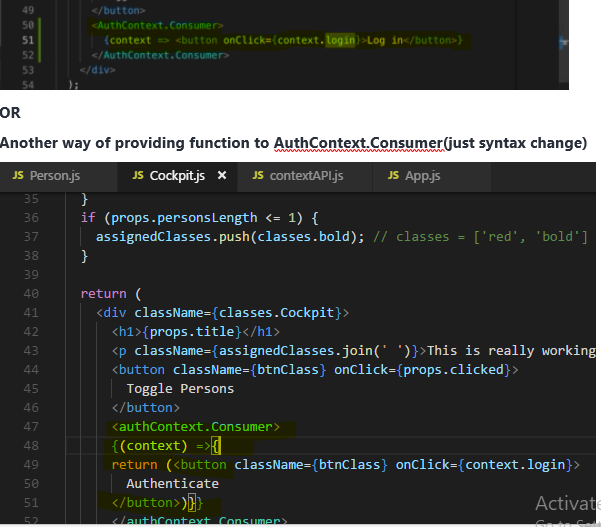


**React will re-render when state or props change. So only changing something in a context object would not cause a re-render cycle and therefore this is not enough. Hence I still manage my authentication status in the state of this component  but I then also store the current state in that authenticated prop of the object I am passing as a value to the authContext and since this effectively is a prop of the authContext provider, this will update whenever this state updates.**









## **contextType and useContext() -Better way of Using ContextAPI( for class-based and functional based component)**

**For class-based component**

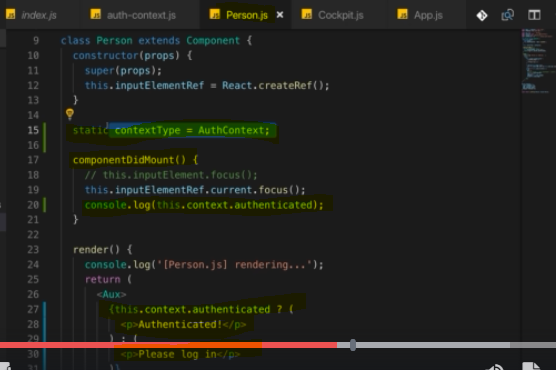
Previous ContextAPI approach code a bit verbose and can be tricky to wrap your head around when you see it the first time, **it also only gives you access to the context in your JSX code here, where you render authContext consumer.**

Thankfully, React 16.6 added another way of using context.

You can add a special static property named **contextType**.

**This allows us to get access to our context even in places like componentDidMount where we previously couldn't.**

**And you can of course also use this context in your render function.**

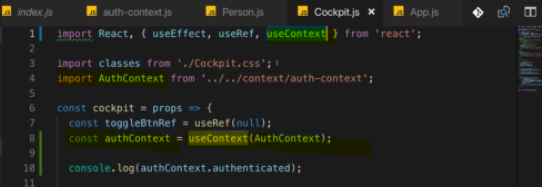


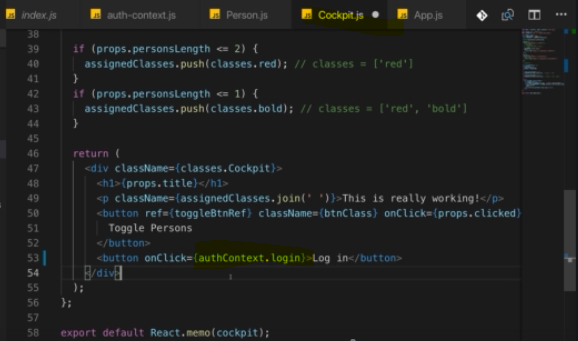
**Now this allows React to automatically connect this component here, this class-based component to your context behind the scenes and it gives you a new property in this component, the this.context property.**

**For Functional Component**

Thankfully, React has you covered with hooks.

**You can import the useContext hook and this now allows you to get access to your context anywhere in your functional component function body**





**So use context is to functional components what static context type is to class-based components you could say.**

# **Prop-types**

you're building a library that you want to distribute to other people, let's say via npm or you're working in a bigger team, then you might have scenarios where people start using your component incorrectly because they don't know that age should be a number, maybe they try to pass in a string. you can do this by importing or by installing an extra package,

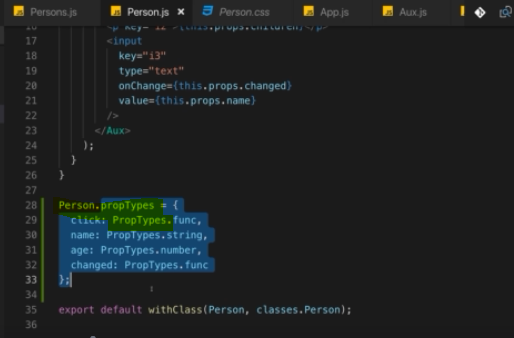


That is provided by the React community or by the official React team actually and it's not built into React core but it is part of React so to say, it's just an extra installation away because it's optional to use.

Now lets use it, first import it in your component



Then use it like this.



on your component you can add an extra property and now this works in both functional and class-based components.

