STATE MANAGEMENT

The elephant in the room.

What is state

“state” is an object that represents the parts of the app that can change. Each component can maintain its own state.

To understand this lets take a step back and re examine the whole react webpage.

Single-page application

A **single-page application** (**SPA**) is a web application or website that interacts with the user by dynamically rewriting the current web page with new data from the web server, instead of the default method of the browser loading entire new pages. The goal is faster transitions that make the website feel more like a native app.(wikipedia)

If you remember, all the react apps we have been creating, have one single html page; index.html and react app injected at with id=root.

This Means we create fast dynamic web applications by never reloading the html page but just changing the contents of the . This is how almost all modern web applications work. (Facebook, Amazon, Twitter, etc). The address change ie(www.facebook.com/campus) are just simulations to make the webapp more navigable without loading any new html pages.

BUT HOW????

React rerenders the whole app or components whenever it is prompted to do so.

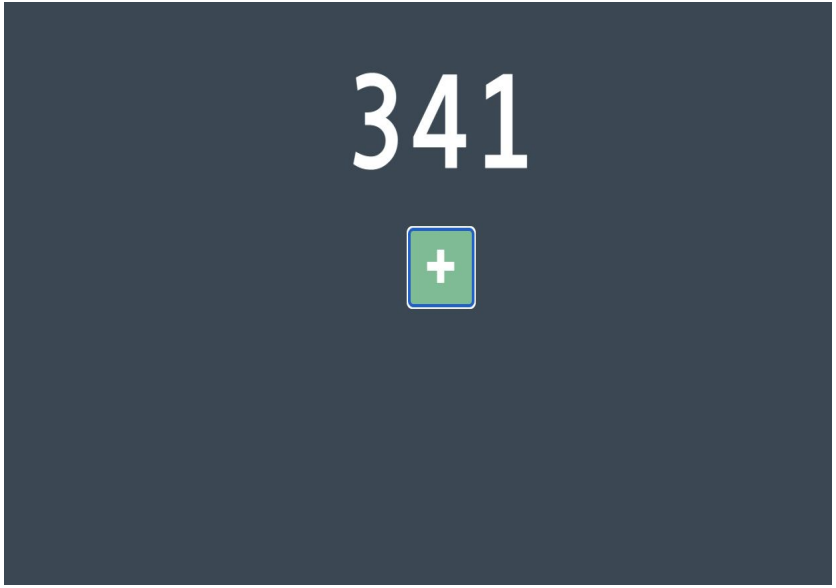
Examples will be provided.

SO NOW WHAT IS STATE????

“state” is still an object that represents the parts of the app that can change. Each component maintains its own state. They help make the site interactive and dynamic. Assuming everything here is a component and that pressing the buttons increases or decreases the count then the only part that changes and which needs to be re rendered is the count currently at 344. Thus the count has a state, which currently is 344!!!!

Rendering components

To the smallest scale, say we have a single component that is part of a huge web app. This for instance is a counter that is increased by one every time the plus button is pressed



STATE MANAGEMENT

Thus we have determined we need to increase the count on pressing the button. Literally this means we will be rendering the count component only, but instead of rendering it with 341, we will first modify that value by adding one to our count then rendering 342.

Again everything else on the screen was not changed, we only rendered 342. The process of doing the above is called state management.

HOW????

Just adding count = count + 1 doesn’t tell react to render that component again so that users see 342. It just increases the count but we dont see it because react hasnt rerendered that component.

Rule 1: Never modify the state directly egcount = count + 1.

So what do we do?????

Hooks!

Using React Declarative programming (That is we are not using classes) we manage state using hooks.

This is just an API that was added to make state management more intuitive.

Use state

This is an example of one of the “hooks” provided.

When declaring a state like for the previous example we code:

const [count setCount] = useState(341)

The line above creates a state called count which is innitialized with the value 341. And it gives us a function that we then can use to modify the state.

Eg setCount(count + 1);

This modifies the count to 342 and instructs react to rerender every instance of count

See example in CODESANDBOX

useEffect

useEffect is another react “hook” that is used to manage the side effects of rendering a component.

It requires deeper understanding of react component lifecycle that I may not be able to teach with the limited time. However, I will use it for network requests in the upcoming assignment so keep an eye out for additional readings in the assignments. Also look at the react documentation and an article that best explains it.