**React Component Lifecycle**

**Objectives**

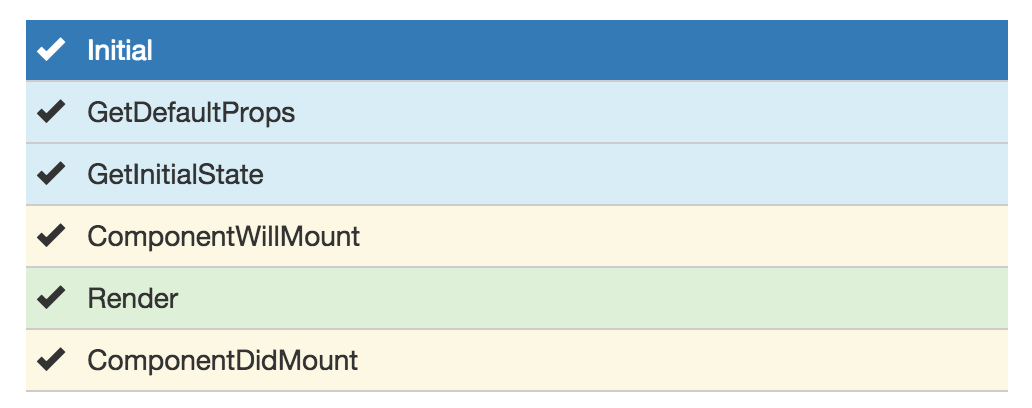
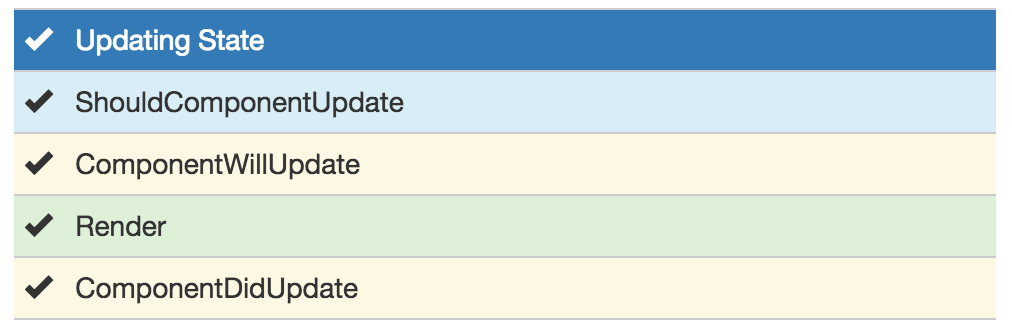
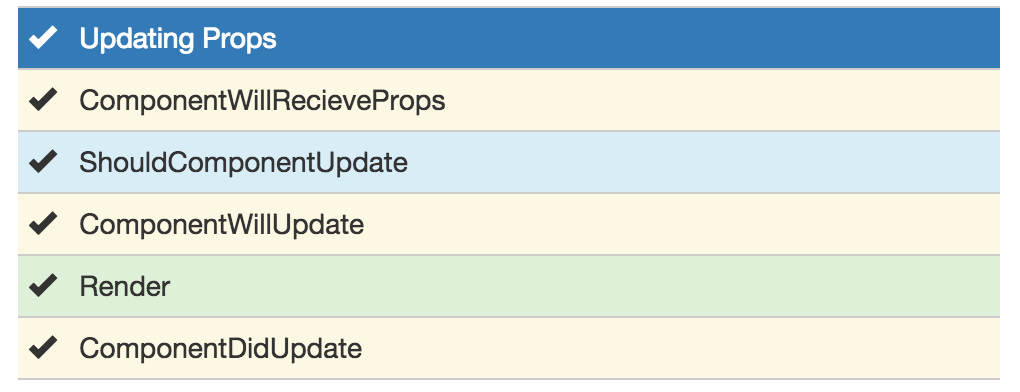
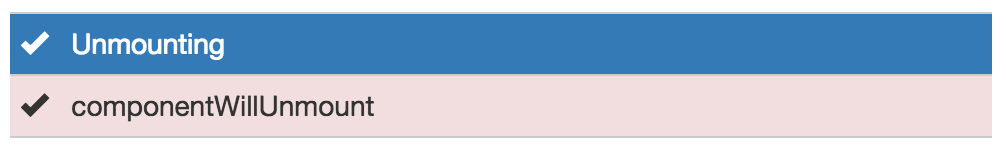
*After this lesson, students will be able to:*

* Have a deeper understanding of mounting, unmounting, and updating React components.
* Understand where to place async calls during the React lifecycle.

**React Component Lifecycle**

When building a React application, EVERYTHING is a component. In React, we usually create a component by extending the React.Component method. When we extend the Component class, we gain the methods that allow us to build a stateful component. So far, we've only been working with the render method, but there are several other functions we have access to when building React apps. These methods get executed every time a component is mounted, updated, or unmounted.

**NOTE** Remember that this only applies to stateful components, these methods are not available in stateless (aka dumb) components.

[](https://imgur.com/BrWR7kt.jpg)[](https://imgur.com/b4aedU0.jpg)[](https://imgur.com/wU8sCmr.jpg)[](https://imgur.com/soRxdUY.jpg)

[*Images from this fantastic blog post by A. Sharif*](https://busypeoples.github.io/post/react-component-lifecycle/)

**Lifecycles**

A React component has 4 separate lifecycles. A lifecycle is a series of methods that prepare the component to be updated in the DOM. These methods are **Mounting**, **Unmounting**, **Updating Props**, and **Updating State**.

**Mounting**

When a component is rendered on screen for the first time, there are 5 methods that are called before, during, and after the component is rendered.

**getDefaultProps()**: The very first method that gets called in the Mounting phase is getDefaultProps. This method looks to the defaultProps object to get initial values. You are able to define defaultProps in any React component. This is really useful when setting default values while waiting for information to come back from the server. You can do this by adding a defaultProps object to the class. It is rare that you will ever need to overwrite this method.

UserComponent.defaultProps = { name: "Stranger", age: "Unknown", location: "Mystery" }

In addition to default props, there is also a .propTypes object available in React. The propTypes object will add a type checker to any prop that you define, and will guarantee that your data comes back in a form you expect. For more info check out [Typechecking with Props](https://facebook.github.io/react/docs/typechecking-with-proptypes.html)

**getInitialState()**: Like get default props, this is just setting up the default state object. This is defined in your component's constructor function. You will never need to overwrite this method.

**componentWillMount()**: componentWillMount is the lifecycle method called immediately after the Component has set it's default state and props and directly before the component is rendered in the DOM. In this method, we can handle configuration, update our state, and prepare for our first render. We can safely refer to our state and props within this method, and we can update our state. It was recently announced that this lifecycle method will be depricated when React 17 is released. We should always use componentDidMount instead.

**render()**: The render() method is the only lifecycle method that is required to be called in every component. We should all be pretty familiar with this one by now. In the render function, you return the JSX that you want to render on the page. Best practices state that the render function should be *pure*. This means that you should not modify the state when rendering. (This is why we make our API calls in componentWillMount)

**componentDidMount()**: This is invoked immediately after a component is mounted and available within the DOM. This allows you to execute code once the component is available and visible on the client's screen. We will be using this method to start API calls with our server. This is most commonly used when connecting React to another library like jQuery or to set up web analytics.

**Updating State and Props**

The mounting lifecycle only takes place when the component is initially being rendered. Any modifications that happen afterward go through the update lifecycle methods. Let's quickly go over these.

**componentWillReceiveProps(nextProps)**: This method will be called whenever React detects the props of a component has been modified. This method allows the developer to have access to both the current props and the new set of props that are coming in. Developers sometimes use this method to setState before the other update methods are executed.

**shouldComponentUpdate(nextProps, nextState)**: Generally, whenever our component receives any new state or props it will update. However, React gives us a method that asks permission. The argument contains the nextState and nextProps, and it returns a boolean value. If shouldComponentUpdate evaluates to false, it will break out of the rest of the update cycle.

**componentWillUpdate(nextProps, nextState)**: Just like componentWillMount but get's called everytime there is an update. You cannot setState here, which should be done in componentWillReceiveProps. This is also being depricated in favor of componentDidUpdate

**render()**: The same render method from mounting also gets called whenever doing an update. See why we want to keep this function pure.

**componentDidUpdate(prevProps, prevState)**: Also similar to componentDidMount(). Allows you to executes functions after the DOM has been updated.

**Unmounting**

**componentWillUnmount()**: There is only one method that gets called whenever you unmount a component.componentWillUnmount is useful for performing cleanup. (i.e. canceling network requests and invalidating timers)

**Summary**

In an ideal world, we wouldn't use lifecycle methods. All our rendering issues would be controlled via state and props.

But it’s not an ideal world, and sometimes you need to exact a little more control over how and when your component is updating. Even when you don't use these methods, it's helpful to demystify what's going on behind the scenes of React.

Use these methods sparingly, and use them with care.

**Further Reading:**

* [React Component Lifecycle by A. Sharif](http://busypeoples.github.io/post/react-component-lifecycle/)
* [Facebook React Docs](https://facebook.github.io/react/docs/react-component.html)
* [State and Lifecycles](https://facebook.github.io/react/docs/state-and-lifecycle.html)

***React: Lifecycles***

***This is one of the more abstract subjects regarding React.js***

***When react goes in and updates it states and updates and renders the DOM; What is actually happening here?***

***Component lifecycle means when we load a react app, every single component has its own set of event listeners to keep track of when things are being rendered into the DOM, updated etc.***

***In ract we have four lifecycles:***

***Mounting the component***

***Updating the component***

***Or Un-mounting the component.***

***Anytime you use a lifecycle it will be one of these three.***

***Throughout mounting and updating, there will always be a render: Render tells the page what JSX should be displayd. Render is absolutely required because react has to know what is meant to be displayed on the screen.***

***With the lifecycles, there are quite a few things that happens.***

***When a lifecycle first gets mounte: getDefaultProps and getInitialState; These two things are grabbing the properties. Getinitalstate, these two methods set all of those values. Componentwillmount is a method you can call to say you haven’t been rendered yet but I want togo ahead and plug in some values here.***

***Render says I can take this JSX abd put all this information into the DOM to be displayed.***

***ComponentDidMount can be usd after a render to connect third-party APIs. Instead of just having a render, we can have an API call using DIdMount.***

***Updating Props:***

***When first loading the page, functions get triggered every single time thee is a change made to state or props. It happens instantaneously, but in that time, these five methods get called. ComponentWillRecieveProps, ShoudComponentUpdate, ComponenWwillUpdate, render, componentDidUpdate.***

***Being familiar with these lifecycles, and the methods associated is so important to understand state and props and the relationship to object and arrays.***

***Un-mounting will removes a component and this is replaced with a different component or at the lowest end a ‘ ‘ or a 0. That is still a replacement whether blank or not.***

***APIs and AJAX***

***There are so many open APIs out there, and leeraging outside third party APIs leads to much better job opprotunities. Finding unique and creative way to leverage data really beefs up my portfolio, and leads to better opportunities.***

***An API is someone else’s server. Another application that is running some data that can be connected to. You can find an API that’s out there to grab random images, gifs, videos, data in general. Data grabbing is awesome and helps create Immersive, impressive sites. When working with APIs its important to LOOK at their data before incorporating the data into your own***

***Reading how th***

***Application Programming Interface:***

***Leveraging reacts API to make components; JQuery provides its own API to grab DOM elements.***

***API applies to everything and is an extremely broad term that mainly applies to third party services. Basically anything that can be utilized on your own server***

***We use a path ( a URL) and add a query parameter on the end to get back JSON data from a massive variety of APIs and anyone can get this info. This data is provided for developers to build out experiences, and push forward the O/S community.***

***APIS all look very diferent from one another. There are very different methods for building APIs.***

***Taking notes on comparisons of APIs:***

***APIs and Keys:***

*Keys for for APIs to know who is making a request from there API. Usually developers must reach out to an API owner through a development website. If Keys provided end up making too many request, the permission can be denied*

*XML FILES; before JSON there was XML. XML is a nightmare to deal with and is not as easy as using JSX JavaScript.*

*Most modern APIS all have moved on to JSON , JSON is basically a string that can be parsed but Ruby or JSX, Try to avoid XML whenever possible.*

*NOT ALL COMPANIES HAVE OPEN APIS:*

*There are several good places to go ; Marvel has an awesome API, HomeDepot does not! HD doesn’t want the public to have keys to the kingdom.*

*ProgrammableWeb is a great one.*

*Interesting/Useful APIs:*

[*https://enterprise.storyblocks.com/api/*](https://enterprise.storyblocks.com/api/)

*The value of utilizing these APIs mad available cannot be overstated. Third party APIs enable developers to create beautiful and Complex applications that are limitless in functionality.*

*When you don’t have an API key, you will be unable to use an API. This is to make sure someone doesn’t build a competetitor site using there OWN tech. Besides getting a key, only hacking in wokrs.*

*API Keys NEVER EVER EVER EVER EVEREVEVRVEVREVREVRVRVRVEVRVRVE should be stored and displayed on github. When using APIs, before even Pushing once, store the API key into your .ENV file and list ENV in your . gitignore,*

*Building a very simple app to post a movie title and poster.*

*Install axios after an npm init and start. When we make an API call using JS…*

*The old school way (don’t use this):*

*XHRHTTP REQUEST this is the old school way of grabbing APIs in JS.*

*There is a new way: FetchAPI.*

*We are going to use something easier and cleaner: Axios. This allows us to make get request, post requests, patches, updates and deletes.*

*Its going to look very similar to the way we have done routing before.*

*AJAX Request:*

const keyword = document.getElementById("#keyword").value; const url = `https://www.omdbapi.com/?t=${keyword}&apikey=d31f1a94`; axios.get(url).then(response => { console.log(response); });

Three important things to notice about axios here...

1. The method called is .get(). We can also use .post(), .put(), .delete(), and more.
2. The .get() method only takes one argument. The url of the api we want to hit.
3. We can tell axios what to do with the data by calling .then().

### Promises

After making an API call, we need to let our program know what to do with the data when it comes back. We can do this by doing something with the its return value, which comes in the form of a **promise**. We can use **promise methods** to tell the axios request what to do if the request is successful or not. We will use two promise methods when working with Axios...

#### .then()

.then() is called whenever the previous method returns information. In the case of AJAX calls, this typically means that .then() will be called once the information has been requested, processed, and sent back to our program. We use .then() to tell our program what to do with this JSON data.

axios.get(url).then(response => { var data = response.data document.window.append(data); });

*EXAMPLES OF AXIOS API CALLS:*

[*https://github.com/axios/axios\\\*](https://github.com/axios/axios\\\)

*Whenever we make s call using async, using this makes KS wait a certain amount of time before .*

*Async function usingAsync() => {*

*Try{*

*Const response = await axios.get(ur;)*

*Console.log(response Date)*

*Catch (err) {  
console.log(err)*

*}*

*This is the ‘”aSync” method of call bak=mm=m =====--*

*We are going to start using Axios INSIDE of our react component*