React Component Lifecycle

# Introduction

This document will discuss the Component Lifecylce in React, what LifeCycle methods are available and how they could fit into an application that you build.

# The Component Lifecycle

The lifecycle of a component begins when it is first rendered in the DOM, it exists as long as it is represented in the DOM until it is removed, each Component has its own lifecycle.

Aside from Render, there are 3 types of Lifecycle methods

1. **Mounting** – The component is being added to the DOM
2. **Updating** – The component is receiving new Props or State, these are called when the component is being re-rendered
3. **Unmounting** – The component is being removed from the DOM

Methods are prefixed with **will** or **did,** indicating if they are executed **before** the event happens, or **after** the event happened.

# Render

* Required in all components, it should examine props and state and return a single element, this element however can have other elements or Components nested inside it.
* Can return null or false indicating there is nothing to render
* **Should not change components state**
* Returns the same result every time it is called
* Does not interact with the browser

render() {

return (

<div>

<Hello />

</div>

);

}

# Mounting Methods

## Constructor

* Constructor for a React component
* Should have a called to **super(props)** before any other code.
* Correct place to initialise state

constructor() {

super();

this.state = {

header: "Default Header",

content: "Default Content"

};

}

## componentWillMount()

* Invoked immediately before mounting occurs.
* Called before render(), therefore changes in state do not trigger re-rendering
* Generally recommended to use a constructor instead, however you may still see this method used as it still exists for backwards compatibility.

## componentDidMount()

* The Component now exists in the DOM.
* Ideal for making requests to the Backend, for example using Axios.
* Calling a setState() method here will cause the Component to re-render

# Updating Methods

## componentWillReceiveProps(nextProps)

* Called before a mounted component receives new props.
* The *nextProps* that we pass into the function are the new props.
* Useful for comparing new props with old to decide if the state should be changed.

## shouldComponentUpdate(nextProps, nextState)

* Purely for efficiency – rendering can be expensive
* Will return a Boolean value that indicates if the next steps in the lifecycle should be executed.
* If false, the Component will return to its running state.

## componentWillUpdate

* Invoked immediately before rendering when new props or state are being received.
* As it is an update method, it is not called during the component’s initial render
* Cannot use **this.setState()** in this method
  + Updating state in response to a prop change should be done in the **componentWillReceiveProps()** method instead
* Similarly to **componentWillMount(),** it is advised to not use this method.

## componentDidUpdate

* Invoked immediately after an update occurs
* Opportunity to operate on the DOM after a component update

# Unmounting Methods

## componentWillUnmount

* Called immediately before a component is unmounted and destroyed
* Opportunity to perform necessary cleanup, e.g. invalidating timers, cleaning up DOM elements created in componentDidMount()

# Example 1

Below I have added an example component that utilises most of the lifecycle methods, it is rather simple but does show of the previously discussed lifecycle methods in action.

import React, { Component } from "react";

export default class Clock extends Component {

constructor(props) {

super(props);

this.state = {

date: new Date(),

text: "",

boolForShould: true

};

console.log("Constructor sets the time as :" + this.state.date);

}

componentWillMount() {

console.log("componentWillMount sets the time as :" + this.state.date);

}

componentDidMount() {

console.log("componentDidMount sets the time as :" + this.state.date);

this.timerID = setInterval(() => this.tick(), 1000);

}

componentWillUpdate() {

console.log("componentWillUpdate sets the time as :" + this.state.date);

}

componentWillUnmount() {

console.log("componentWillUnmount sets the time as :" + this.state.date);

clearInterval(this.timerID);

}

shouldComponentUpdate() {

console.log("shouldComponentUpdate sets the time as :" + this.state.date);

return this.state.boolForShould;

}

componentDidUpdate() {

console.log("componentDidUpdate sets the time as :" + this.state.date);

}

tick() {

this.setState({

date: new Date()

});

}

stateHandle = () => {

console.log("stateHandle sets the time as :" + this.state.date);

this.setState({

text: "Updated"

});

};

stateHandle2 = () => {

console.log("stateHandle2 sets the time as :" + this.state.date);

clearInterval(this.timerID);

this.setState({

text: "Updated and timer has stopped"

});

};

shouldHandle = () => {

console.log("shouldHandle sets the time as :" + this.state.date);

this.setState({

boolForShould: !this.state.boolForShould

});

};

render() {

return (

<div>

<h1>{this.state.text}</h1>

<h2>It is {this.state.date.toLocaleTimeString()}</h2>

<button onClick={this.stateHandle}>Changes State</button>

<button onClick={this.stateHandle2}>

Changes State and stops timer

</button>

<button onClick={this.shouldHandle}>

Changes shouldComponentUpdate

</button>

</div>

);

}

}

# Example 2

The previous example, through thorough, was quite contrived, the component below endeavours to show a more realistic use of some of the lifecycle methods involving Axios request calls.

import React, { Component } from "react";

import axios from "axios";

class AutoFilm extends Component {

constructor(props) {

super(props);

this.state = {

data: "",

apikey: "",

filmTitle: ""

};

}

componentDidMount() {

this.makeRequest();

}

makeRequest = () => {

axios

.get("http://www.omdbapi.com/?apikey=cf6d6c63&t=hop")

.then(response => {

console.log(response.data);

this.setState({

data: response.data

});

});

};

render() {

return (

<div>

<h2>AutoFilm.js</h2>

<h4>{this.state.data.Title}</h4>

<h4>{this.state.data.Year}</h4>

<h4>{this.state.data.Rated}</h4>

<h4>{this.state.data.Genre}</h4>

<h4>{this.state.data.Plot}</h4>

<img src={this.state.data.Poster} />

</div>

);

}

}

# Conclusion

In conclusion it can be hard to master working with React Lifecycles, however you need to appreciate that not everything is done in the render method or constructor. There are other methods that can be called at different points in the lifecycle that may be more practical or relevant, a good developer will know when to use these methods.