React Life-Cycle

Each component in React has a lifecycle which you can monitor and manipulate during its three main phases.

The three phases are: **Mounting**, **Updating**, and **Unmounting**

1. **componentWillMount is executed before rendering, on both the server and the client side.**
2. **componentDidMount is executed after the first render only on the client side. This is where AJAX requests and DOM or state updates should occur. This method is also used for integration with other JavaScript frameworks and any functions with delayed execution such as setTimeout or setInterval. We are using it to update the state so we can trigger the other lifecycle methods.**
3. **componentWillReceiveProps is invoked as soon as the props are updated before another render is called. We triggered it from setNewNumber when we updated the state.**
4. **shouldComponentUpdate should return true or false value. This will determine if the component will be updated or not. This is set to true by default. If you are sure that the component doesn't need to render after state or props are updated, you can return false value.**
5. **componentWillUpdate is called just before rendering.**
6. **componentDidUpdate is called just after rendering.**
7. **componentWillUnmount is called after the component is unmounted from the dom. We are unmounting our component in main.js.**

A screenshot of a cell phone

Description automatically generated

**Mounting -** Mounting means putting elements into the DOM.

React has four built-in methods that gets called, in this order, when mounting a component:

1. constructor()

* The constructor() method is called before anything else, when the component is initiated, and it is the natural place to set up the initial state and other initial values.
* The constructor() method is called with the props, as arguments, and you should always start by calling the super(props) before anything else, this will initiate the parent's constructor method and allows the component to inherit methods from its parent (React.Component)

1. getDerivedStateFromProps()

* The getDerivedStateFromProps() method is called right before rendering the element(s) in the DOM.
* This is the natural place to set the state object based on the initial props.
* It takes state as an argument, and returns an object with changes to the state.
* The example below starts with the favorite color being "red", but the getDerivedStateFromProps() method updates the favorite color based on the favcol attribute

1. render()

* The render() method is required, and is the method that actual outputs HTML to the DOM.

1. componentDidMount()

* The componentDidMount() method is called after the component is rendered

**Updating**

The next phase in the lifecycle is when a component is *updated*.

A component is updated whenever there is a change in the component's state or props.

React has five built-in methods that gets called, in this order, when a component is updated:

1. getDerivedStateFromProps()
2. shouldComponentUpdate()

* In the shouldComponentUpdate() method you can return a Boolean value that specifies whether React should continue with the rendering or not.
* The default value is true.
* The example below shows what happens when the shouldComponentUpdate() method returns false:

1. render()
2. getSnapshotBeforeUpdate()

* In the getSnapshotBeforeUpdate() method you have access to the props and state before the update, meaning that even after the update, you can check what the values were before the update.
* If the getSnapshotBeforeUpdate() method is present, you should also include the componentDidUpdate() method, otherwise you will get an error.

1. componentDidUpdate()
2. The componentDidUpdate method is called after the component is updated in the DOM.

## Unmounting

When a component is removed from the DOM, or unmounting as React likes to call it.

React has only one built-in method that gets called when a component is unmounted:

1. componentWillUnmount()
2. The componentWillUnmount method is called when the component is about to be removed from the DOM.

**Redux**: Redux is a state management system. Therefore, we will need:

1. a place to save the state
2. a method to get the state
3. a method to change the state

A close up of a sign

Description automatically generated

1. store is the place we save the state

import { createStore } from "redux";

import { reducer } from "./reduxModule";

const store = createStore(reducer);

1. getState is the method to get the state

const state = store.getState();

1. action & reducer is the method to change the mapStateToProps

const INCREMENT = "redux/increment";

const initialState = {

counter: 0,

};

export const reducer = (state = initialState, action) => {

switch (action.type) {

case INCREMENT:

return {

counter: state.counter + action.amount

};

default:

return state;

}

};

export const incrementAction = (amount = 1) => {

return {

type: INCREMENT,

amount,

};

};