Frontend Assignment

Name: Mahavirsinh P. Dodiya

Course: Front End Development

Assignment- React JS

(MODULE: List and Hooks)

Q) Explain Life cycle in Class Component and functional component with Hooks.

Ans) Component lifecycle in React and its methods are an essential part of developing applications in React. While today the approach is being increasingly superseded by React hooks, it's worth it to take a closer look at how it works, study the relationship between class components and functions, and gain a deeper understanding of DOM manipulation with React. Let's practice the class-based methods and review their relevance in the light of useEffect's emergence.

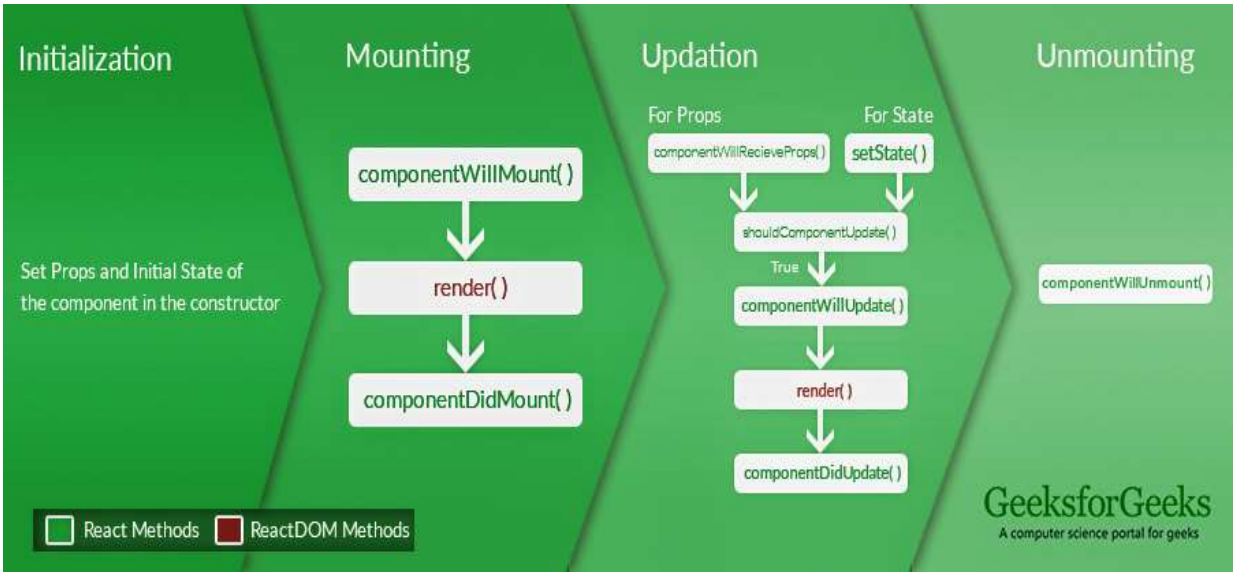
React lifecycle method explained

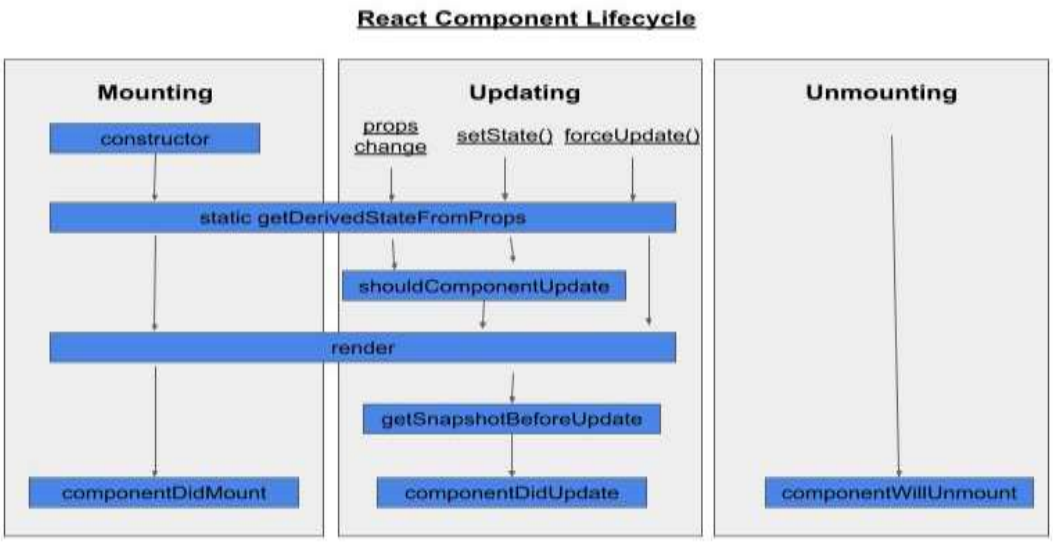
First, let's take a look at how it's been done traditionally. In order to do that, we're going to take a closer look at React components.

As you probably know, each React component instance has a lifecycle. The component's lifecycle consists of three phases:

- **Mounting lifecycle methods**, that is inserting elements into the DOM.
- **Updating**, which involves methods for updating components in the DOM.
- **Unmounting**, that is removing a component from the DOM.

Each phase has its own methods, which make it easier to perform typical operations on the components. With class-based components, React developers directly extend from the *React.Component* in order to access the methods.





Class components vs functional components

As you probably know, an alternative way of taking advantage of lifecycle methods is to use hooks. With the release of React 16.8 back in March 2019, it is now possible to create functional components that are **not** stateless and **can** use lifecycle methods.

It's all thanks to the useState and useEffect hooks – special functions that hook into React features that allow to set the initial state and use lifecycle events in functional components. Currently, it is possible to emulate the performance of

almost any supported lifecycle method by skilfully applying these two hooks in your pure JavaScript functions.

Are hook-enhanced functional components superior to class based ones? Before we get to that, let's go over the lifecycle phases using the traditional approach.

Mounting in the React component lifecycle

As we mentioned, during the mounting phase of the lifecycle, the class component is inserted into the DOM. A good example would be ***componentDidMount()*** – a lifecycle method that runs after the component is

mounted and rendered to the DOM. It is great when you want to do an interval function or an asynchronous request.

Example:

```
componentDidMount() {
  fetch(url).then(res => {
    // Handle response in the way you want.
    // Most often with editing state values.
  })
}
```

Updating in the React component lifecycle

The **componentDidUpdate()** render method is called right after the updating happens. This one is called always except for the initial render. That's a good place to interact with a non-reactive environment. It's a good idea to make http requests here.

You can call **setState()** in this method to enqueues changes to the component state. but it is very important to wrap that in some condition to avoid an infinite loop (doesn't matter if state has the same values or not). If there is no condition, the process goes as follows:

1. You call setState() in the componentDidUpdate() method.
2. The component is updated.
3. componentDidUpdate() is invoked.
4. setState() is called again …

```
componentDidUpdate(prevProps, prevState) {
  // Always compare props or state
  if (this.props.counter !== prevProps.counter) {
```

```
    this.postCounter(this.props.counter);
  }
}
```

Unmounting in the React component lifecycle

componentWillUnmount() is invoked just before the component is removed from the DOM. You should use that to remove event listeners, clear intervals and cancel requests. In other words: all the needed cleanup.

```
componentWillUnmount() {
        document.removeEventListener("click", this.someFunction);
}
```

You shouldn't use setState in that method because the component won't be rerendered anymore.

React component lifecycle with hooks

You can take advantage of the useEffect hook to achieve the same results as with the componentDidMount, componentDidUpdate and componentWillUnmount methods. useEffect accepts two parameters. The first one is a callback which runs **after render**, much like in componentDidMount. The second parameter is the effect dependency array. If you want to run it on mount and unmount only, pass an empty array [].

To clean up, return the callback in useEffect:

```
useEffect(
  () => {
    document.addEventListener(“click”, someFunc);
```

```
    return () => {
      document.removeEventListener("click", someFunc);
    };
  },
  []
);
```

If you want it to behave like componentDidUpdate, put some dependencies into the array or don't pass the second argument at all.

You can also use useState instead of this.state in class components. Instead of:

```
this.state = {
        counter: 0,
        usersList: [],
}
```

You can do that:

```
const [counter, setCounter] = useState(0);
const [usersList, setUsersList] = useState([]);
```

As you can, it is possible to use hooks to achieve similar or the same end results.

Class components vs hooks (is there a clear victor?)

- As you can see, both class components and hooks have their pros and cons. Does it mean that the choice is mostly a matter of personal preference? For the most part, that's the case. But there are some things worth noting:
- If you are more used to functional programming, you will definitely enjoy using hooks.
- Developers can use functional components without having to convert them into class components.
- Despite the fact that hooks are really popular nowadays, there is nothing wrong with using class components. Everything that you can do with hooks can also be done with class components.
- There are also other related topics worth exploring to make your choice even more informed, including network requests, choosing the only required method, child component issues and other React elements.
- Class components aren't deprecated and are not going to be anytime soon so use them if that style suits you more.