Master MERN(React-2)

React Fundamentals

Components+Hooks+Props

- **What are different types of components in react ?**
  - ➔ There are two types of component in react - function and class based component

- **Difference between both of them ?**

  Syntax Difference
  - ➔ The most obvious one difference is the syntax. A functional component is just a plain JavaScript function which accepts props as an argument and returns a React element.
  - ➔ The most obvious one difference is the syntax. A functional component is just a plain JavaScript function which accepts props as an argument and returns a React element.
  - ➔ Example to show case above difference -

```
var Welcome = function (_React$Component) {
  _inherits(Welcome, _React$Component);

  function Welcome() {
    _classCallCheck(this, Welcome);

    return _possibleConstructorReturn(this, (Welcome.__proto__ || Object.getPrototypeOf(Welcome)).apply(this, arguments));
  }

  _createClass(Welcome, [{
    key: 'render',
    value: function render() {
      return _react2.default.createElement(
        'h1',
        null,
        'Hello, ',
        this.props.name
      );
    }
  }]);

  return Welcome;
}(_react2.default.Component);
```

```
function Welcome(props) {
  return _react2.default.createElement(
    'h1',
    null,
    'Hello, ',
    props.name
  );
}
```

➔ ***State*** :

➢ Because a functional component is just a plain JavaScript function, you cannot use setState() in your component. That's the reason why they also get called functional stateless components. So everytime you see a functional component you can be sure that this particular component doesn't have its own state.

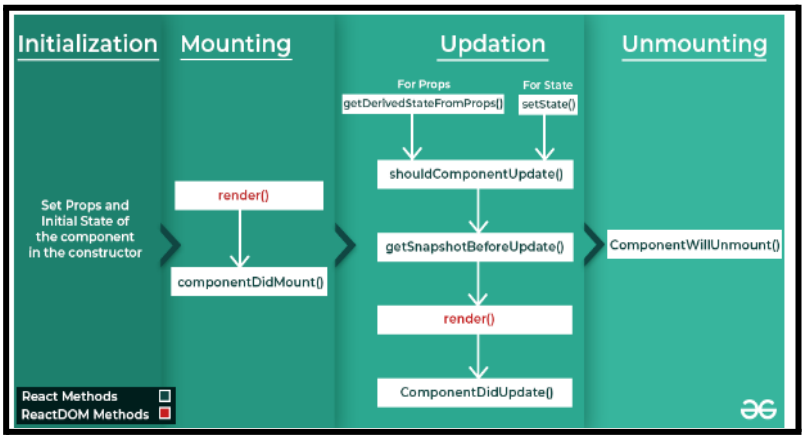
➔ ***LifeCycle Hooks***

➢ Another feature which you cannot use in functional components are lifecycle hooks.The reason is the same as for state, all lifecycle hooks are coming from the React.Component which you extend from in class components. So if you need lifecycle hooks you should probably use a class component.

- **Explain life cycle features of a class based component ?**

➔ Every React Component has a lifecycle of its own, the lifecycle of a component can be defined as the series of methods that are invoked in different stages of the component's existence. A React Component can go through four stages of its life as follows.

- **Initialization:** This is the stage where the component is constructed with the given Props and default state. This is done in the constructor of a Component Class.
- **Mounting:** Mounting is the stage of rendering the JSX returned by the render method itself.
- **Updating:** Updating is the stage when the state of a component is updated and the application is repainted.
- **Unmounting:** As the name suggests Unmounting is the final step of the component lifecycle where the component is removed from the page.



➔ For more details please refer - https://www.geeksforgeeks.org/reactjs-lifecycle-components/

- **What are function based components ?**

  ➔ **ReactJS Functional components are some of the more common components that will come across while working in React. These are simply JavaScript functions. We can create a functional component in React by writing a JavaScript function. These functions may or may not receive data as parameters. In the functional Components, the return value is the JSX code to render to the DOM tree.**

- ➔ When a component re-renders, every function inside the component is recreated, and therefore these functions' references change between renders.
- ➔ useCallback(callback, dependencies) will return a memoized instance of the callback that only changes if one of the dependencies has changed. Instead of recreating the function object on every re-render, we can use the same function object between renders.

- **What are Hooks ?**
  - ➔ Hooks allow function components to have access to state and other React features. Because of this, class components are generally no longer needed. Hooks allow us to "hook" into React features such as state and lifecycle methods.
  - ➔ Hooks can only be called at the top level of a component.

- **Which react method is used for "unmounting" of any class based component inside a function based component ?**
  - ➔ Use the useEffect hook to run a react hook when a component unmounts. The function we return from the useEffect hook gets invoked when the component unmounts and can be used for cleanup purposes.
  - ➔ The cleanup function is often used to remove any previously added event listeners to avoid memory leaks.

- **What is the side effect inside React ?**

  - ➔ A "side effect" is anything that affects something outside the scope of the function being executed. These can be, say, a network request, which has your code communicating with a third party (and thus making the request, causing logs to be recorded, caches to be saved or updated, all sorts of effects that are outside the function.
  - ➔ There are more subtle side effects, too. Changing the value of a closure-scoped variable is a side effect. Pushing a new item onto an array that was passed in as an argument is a side effect.

- **Explain useState and useEffect Hook with an example ?**
  - ➔ Refer to this for an good illustrative example -

    https://sebhastian.com/react-usestate-useeffect-hooks/

- **Explain useCallback Hook with an example ?**
  - ➔ When a component re-renders, every function inside the component is recreated, and therefore these functions' references change between renders.
  - ➔ useCallback(callback, dependencies) will return a memoized instance of the callback that only changes if one of the dependencies has changed. Instead of recreating the function object on every re-render, we can use the same function object between renders.
  - ➔ Imaging there is the EmployeeList component that renders the list of employees. The list could be extensive, maybe hundreds of items. To prevent useless list re-renderings, you wrap it into React.memo().The parent component of EmployeeList provides a handler function to know when an

item is clicked. onItemClick callback is memoized by useCallback(). As long as the term is the same, useCallback() returns the same function object.

➔ When the Dashboard component re-renders, the onItemClick function object remains the same and doesn't break the memoization of EmployeeList.

```
import useSearch from './fetch-items';
function EmployeeList({ term, onItemClick }) {
    const items = useSearch(term);
    const map = item => <div onClick={onItemClick}>{item}</div>;
    return <div>{items.map(map)}</div>;
}
export default React.memo(EmployeeList);
```

```
import { useCallback } from 'react';
export function Dashboard({ term }) {
    const onItemClick = useCallback(event => {
    console.log('You clicked ', event.currentTarget);
    }, [term]);
    return (
   <EmployeeList
        term={term}
        onItemClick={onItemClick}
   />
);
}
```

- **What are Ref() in React ?**

➔ Refs are a function that is used to access the DOM from components. You only need to attach a ref to the element in your application to provide access to it from anywhere within your component without using props and all.

- ➔ We can also use Refs to direct access to React elements and use callbacks with them. We should only use refs when the required interaction cannot be achieved using state and props.
- ➔ We can use refs to do anything that needs to be manipulated in the DOM. Some exemplary cases include focus, test selection, media playback, triggering mandatory animations, or integrating with the third-party DOM library.
- ➔ For more information, please refer -

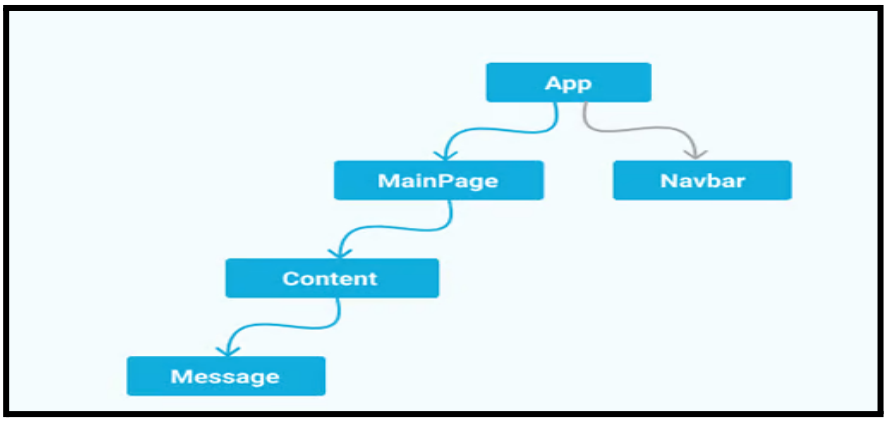
  https://www.knowledgehut.com/blog/web-development/react-functional-components

- **Explain useMemo() hook with an example -**
  - ➔ This can be easily understood with the following example link -

    https://www.w3schools.com/react/react_usememo.asp

- **Can "Props" be passed inside class-based components in React ?**
  - ➔ Props can be passed inside class-based components in React. Props are like HTML attributes and can pass any JavaScript value, including objects, arrays, and functions.
- **What is Prop Drilling ?**
  - ➔ Prop drilling is the unofficial term for passing data through several nested children components, in a bid to deliver this data to a deeply-nested component. The problem with this approach is that most of the components through which this data is passed have no actual need for this data. They are simply used as mediums for transporting this data to its destination component.

➔ This is where the term “drilling” comes in, as these components are forced to take in unrelated data and pass it to the next component, which in turn passes it, and so on, until it reaches its destination.



- **What are “Props” and their features in React ?**
  - ➔ In ReactJS, the props are a type of object where the value of attributes of a tag is stored. The word “props” implies “properties”, and its working functionality is quite similar to HTML attributes.
  - ➔ Basically, these props components are read-only components. In ReactJS, the data can be passed from one component to another component using these props, similar to how the arguments are passed in a function. Inside the component, we can add the attributes called props; however, we cannot change or modify props inside the component as they are immutable.
  - ➔ States are the type of built-in object in ReactJS.
  - ➔ We can create, handle, or manage our data within the component using the state object.Data can be passed by the props but the data cannot be passed by state itself. Using the state, the data is managed internally within the component.
- **How to pass the data from Child to Parent in react** ?
  - ➔ In the parent component, create a callback function. This callback function will retrieve the data from the child component.
  - ➔ Pass the callback function to the child as a prop from the parent component.
  - ➔ The child component calls the parent callback function using props and passes the data to the parent component.

➔ **For in-depth explanation with example, please refer -** https://www.geeksforgeeks.org/how-to-pass-data-from-child-component-to-its-parent-in-reactjs/

➔

1. React Components
2. React Hooks

3. APIs and Data Fetching in Frontend
4. React Redux
5. Project Implementation work in real Life on React