**React Js-**

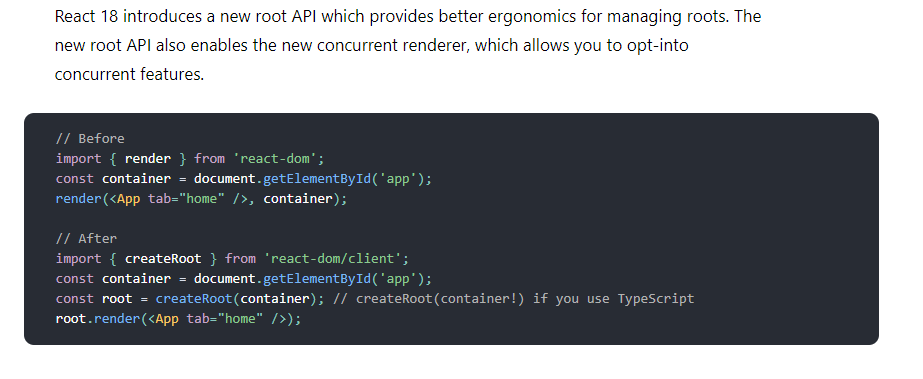
**JSX—**



This funny tag syntax is neither a string nor HTML.

Color-rgb(40,44,52)

It is called JSX, and it is a syntax extension to JavaScript. We recommend using it with React to describe what the UI should look like. JSX may remind you of a template language, but it comes with the full power of JavaScript.



**Virtual Dom ---**

A copy of dom is created alongside the actual dom. It is used to identify changes.

**State ---**

Trigger re render only when

1.when the state changes

2.when the props changes

**UseEffect** 🡪 <https://www.memberstack.com/blog/uselayouteffect-vs-useeffect>

**UseRef** 🡪 <https://www.memberstack.com/blog/react-refs>

**UseContext** 🡪 <https://www.memberstack.com/blog/react-context>

**UseReducer** 🡪 <https://www.memberstack.com/blog/react-usereducer>

**UseCallback** 🡪 <https://www.memberstack.com/blog/using-usecallback>

**UseMemo** 🡪 <https://www.memberstack.com/blog/react-usememo-hook>

**React Fragments** 🡪 <https://www.memberstack.com/blog/react-fragment>

For state changing we use useState

Const [ state , function for changing state] = useState()

EX- const [counter ,setcounter] = useState(0)

useEffect(function, dependency array{

// by default runs after every re render

if any time our dependency array changes , function is called

}

useref – if the value changes it again does not rerendered it.{

//preserves previous value

//does not trigger re render

//target dom nodes/element

}

/\*\*

Const products = useRef([ ])

Products -> {current : [ ]}

Products.current.push(“shirt”);

Products -> {“shirt”}

Note : would not trigger a re render

Const [products ,setProducts ] = useState([ ])

Products -> [ ]

setProducts([“shirt”])

products - > [“shirt”]

Trigger a re render

\*\*/

Optimization in react

UseMemo(function ,dependency array)

UseCallback(function , dependency array)

1 render ->1011 memory location

2 render ->1011 memory location

3 render ->1011 memory location

**UseReducer 🡪 https://www.memberstack.com/blog/react-usereducer**

**UseContext 🡪**

A solution to fix the problem of prop drilling

We create a context via createContext command and then we provide it to the highest hierarchy of the component so that any child component can use the pros which have been passed in provide.

**React Router🡪**

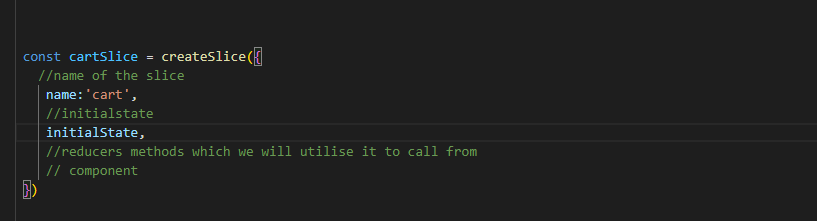
**Axios🡪**

* import axios
* axios.get(url) , axios.post(url) , axios.patch/put(url) ,axios.delete(url)
* default get axios(url)
* returns a promise
* response data located in data object
* error in error.response
* ####Headers
* Second argument axios.get(url , { })
* Third argument in requests with data
* Axis.put( url , { data } ,{ } )
* ####Post request
* Send data to the sever
* Axios.post(url,{data})
* More options (auth header) – axios.post (url,{data},{})
* ####Global default
* Axios.defaults.header.common[‘Accept’]=’application/json’
* Axios.defaults.baseURL=’http://api.examle.com’
* Axios.defaults.header.common[‘Authorization’]=AUTH\_TOKEN
* Axios.defaults.header.post[‘content-type’]=’application/x-www-form-’

**Redux-State Management tool**

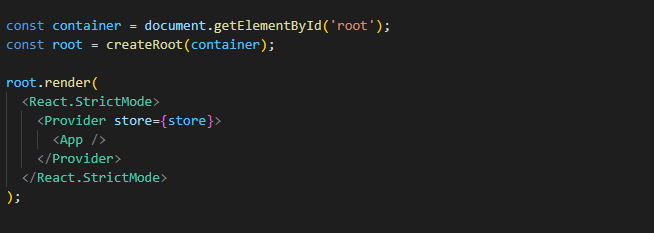
**Redux**🡪 <https://www.memberstack.com/blog/react-redux>

1.Creating a SLICE



2.STORE – Redux helps to put all the application states in a global variable called the store**. Redux will offer us a central store to hold all the React application states. It has an API so any component at any level of the app can access and update the state at any time.**

3.Connect the store to your app



4.Access redux inside your component using { UseSelector , UseDispatch }