*reactJs*

**==>** If you want a function to be executed **after the state change occurs**, pass it in as a **callback**.

this.setState({value: event.target.value}, function () {

console.log(this.state.value); // this line will print after state is set

});

if **setState**() is called within **componentWillMount**() , the state update will be synchronous, and the Component will only render once

***Some of the major advantages of React*** :

* It increases the application’s performance
* It can be conveniently used on the client as well as server side
* Because of JSX, code’s readability increases
* React is easy to integrate with other frameworks like Meteor, Angular, etc
* Using React, writing UI test cases become extremely easy

***React and MVC/FLUX***

React has their own architecture called Flux ➡️. The problem with a MVC structure is it’s bidirectional communication

Flux is made up of 4 key elements:

* Actions
* Stores
* The Dispatcher
* Views

**Folder Structure in react**

structure applications by separating **stateful containers** from my **stateless components**. This way, when my application components start to get too large, I can take components that are used in multiple places and modularize them in an component library.

In addition, my **containers** and **components** are **grouped by route**. For example, in my containers/ folder, I will have a subfolder called home/ for all of the containers that live on the home page. The same is true for my components/ folder.

***Functional vs Pure components***

**functional** components are also commonly referred to as **stateless components**.

They dont have state and lifecycle methods

PURE COMPONENTS ARE NOT FUCTIONAL

Pure components are created by extending react.PureComponent class .i.e.:

**PureComponent** implements a **shallow** comparison (of **old & new State & props**) for us by default in shouldComponentUpdate() , saving us time and reducing the complexity of our components.

**U may use RECOMPOSE library’s HOC of pure to convert a functional component to something like React.PureComponent**

***HOC (High order Component)***

A higher-order component is a function that takes a component as an argument and returns a component.



**It is just a function that takes a component as input and returns the same component with new(changed / additional) props**

A HOC doesn’t modify the input component, nor does it use inheritance to copy its behavior. Rather, a HOC composes the original component by wrapping it in a container component. **A HOC is a pure function** with zero side-effects.

A higher-order component in React is a pattern used to share common functionality between components without repeating code.

***things to be kept in mind (while working wid HOC):***

* A HOC should be a pure function with no side-effects. It should not make any modifications and just compose the original component by wrapping it in another component.
* Do not use HOC’s in the render method of a component. Access the HOC outside the component definition.
* Static methods must be copied over to still have access to them. A simple way to do this is the hoist-non-react-statics package.
* Refs are not passed through.



***Cons of Rendering React on the Server***

* SSR can improve performance if your application is small. But it can also degrade performance if it is heavy.
* It increases response time (and it can be worse if the server is busy).
* It increases response size, which means the page takes longer to load.
* It increases the complexity of the application.

***Advantages of Server Side Rendering?***

**In Client-side rendering,** your browser downloads a minimal HTML page. It renders the JavaScript and fills the content into it.

**Server-side rendering**, on the other hand, renders the React components on the server. The output is HTML content.

**1- SEO**

Unfortunately, Search engine crawlers do not yet understand/render JavaScript

This means they see a blank page, no matter how helpful your site is

**2. Improve performance 🚀**

In SSR, the application performance depends on the server’s resources and user’s network speed. This makes it very useful for ***content-heavy sites***

**UNI\_DIRECTIONAL DATA FLOW IN REACT**



View calls action which setState that updates view

***Limitations of React are listed below***:

* React is just a library, not a full-blown framework
* It can be little difficult for the novice programmers to understand
* Coding gets complex as it uses inline templating and JSX

***wat is Virtual DOM***

A virtual DOM is a lightweight JavaScript object which originally is just the copy of the real DOM. It is a **node tree** that lists the elements, their attributes and content as **Objects and their properties**. React’s render function creates a node tree out of the React components. It then updates this tree in response to the mutations in the data model which is caused by various actions done by the user or by the system.

At any given time, ReactJS maintains two virtual DOM, one with the updated state Virtual DOM and other with the previous state Virtual DOM.

ReactJS using diff algorithm compares both the Virtual DOM to find the minimum number of steps to update the Real DOM

This Virtual DOM works in three simple steps.

* Whenever any underlying data changes, the entire UI is re-rendered in Virtual DOM (if a component is updated all its children are rerendered in vDom)
* Then the difference between the previous DOM representation and the new one is calculated
* Once the calculations are done, the real DOM will be updated with only the things that have actually changed.

**Each Virtual Dom node contains :**

* nodeName (e.g. div, p h1)
* attributes(e.g. class)
* children(child nodes)



***8. Why can’t browsers read JSX?***

Browsers can only read JavaScript objects but JSX in not a regular JavaScript object. Thus to enable a browser to read JSX, first, we need to transform JSX file into a JavaScript object using JSX transformers like **Babel** and then pass it to the browser.



***12. Explain the purpose of render() in React.***

It returns a single React element which is the representation of the native DOM component. in virtual DOM this render function creates the node representing the component

If more than one HTML element needs to be rendered, then they must be grouped together inside one enclosing tag such as <form>, <group>,<div> etc. This function must be kept **pure**

**14. What is Props?**

Props is the shorthand for Properties in React. They are read-only components which must be kept **immutable**. They are always passed down from the parent to the child components

**15. What is a state in React and how is it used?**

States are the heart of React components. States are the source of data and must be kept as simple as possible. Basically, states are the objects which determine components rendering and behavior. They are **mutable** unlike the props and create dynamic and interactive components.



***What are selectors in redux ?***

*a function that knows how to extract a specific piece of data from the store.*



Selectors are functions that take Redux state as an argument and return some data to pass to the component.

**Why should it be used?**

Your data object shape keeps varying as your application grows, so rather than making changes in all the related component.It is much recommended/easier to change the data at one place.



***three principles that Redux follows?***

**Single source of truth:** The state of the entire application is stored in an object/ state tree within a single store. The single state tree makes it easier to keep track of changes over time and debug or inspect the application.

**State is read-only:** The only way to change the state is to trigger an action. An action is a plain JS object describing the change. Just like state is the minimal representation of data, the action is the minimal representation of the change to that data.

**Changes are made with pure functions:** In order to specify how the state tree is transformed by actions, you need pure functions. Pure functions are those whose return value depends solely on the values of their arguments.

***42. Explain the role of Reducer.***

Reducers are pure functions which specify how the application’s state changes in response to an ACTION. Reducers work by taking in the **previous state** and **action**, and takes a **clone** and then it returns a **new state.**

***Component LifeCycle Methods***

* **componentWillMount() -** *Invoked once, both on the client and server, immediately before the initial rendering occurs*. If you call setState within this method, render() will see the updated state and will be executed only once despite the state change.DO NOT MAKE AJAX CALLS HERE
* **UNSAFE\_componentWillReceiveProps(object nextProps)** *it is only called when the props have changed and when this is not an initial rendering*. Calling this.setState() within this function will not trigger an additional render. One common mistake is for code executed during this lifecycle method to assume that props have changed.
* recently ‘componentWillReceiveProps’is replaced with ‘**getDerivedStateFromProps(nextProps, prevState)**’ this method can NOT access this. it either returns an object to update state or returns null to depict no change.
* **componentWillUnmount() -** Invoked immediately before a component is unmounted from the DOM. Perform any necessary cleanup in this method, such as invalidating timers or cleaning up any DOM elements that were created in componentDidMount.
* **componentWillUpdate(object nextProps, object nextState)** - Invoked immediately before rendering when new props or state are being received. not on initial render.This method is not called for the initial render.
* **componentDidMount()** - *Invoked once, only on the client (not on the server), immediately after the initial rendering occurs*. At this point in the lifecycle, you can access any refs to your children (e.g., to access the underlying DOM representation). The componentDidMount() method of child components is invoked before that of the parent component.

DO NOT CALL THIS.SETSTATE HERE). Make Ajax calls here **(The initial ajax call should be made here)**

* **componentDidUpdate(object prevProps, object prevState)** - Invoked immediately after the component’s updates are flushed to the DOM. This method is not called for the initial render. Use this as an opportunity to operate on the DOM when the component has been updated.(DO NOT CALL THIS.SETSTATE HERE). Make Ajax calls here
* **shouldComponentUpdate** is always called before the render method and enables to define if a re-rendering is needed or can be skipped. Obviously this method is never called on initial rendering. A **boolean** value must be **returned**.
* **componentDidCatch()** componentDidCatch(error, info)
* This lifecycle is invoked after an error has been thrown by a descendant component. It should be used for things like logging errors

**Error Boundry in react**

Error boundaries are React components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI instead of the component tree that crashed. Error boundaries catch errors during rendering, in lifecycle methods, and in constructors of the whole tree below them.

A class component becomes an error boundary if it defines **componentDidCatch** method

**Q Ref vs Keys**

refs manipulate **actual**DOM as opposed to **virtual**DOM

**UseEffect**

**Example as componentWillUnmount**



useEffect as componentdidmount and didUpdate both





error boundaries only catch errors in the components below them in the tree. An error boundary can’t catch an error within itself.

***KEYS in react***

Keys in React are used to identify unique VDOM Elements with their corresponding data driving the UI; having them **helps React optimize rendering by recycling existing DOM elements**. Let’s look at an example to portray this.

We have two <TwitterUser> Components being rendered to a page, drawn in decreasing order of followers:



Let’s say that B gets updated with 105 Twitter followers, so the app re-renders, and switches the ordering of A and B:



Without keys, React would primarily re-render both <TwitterUser> Elements in the DOM. It would re-use DOM elements, but React won’t re-order DOM Elements on the screen.

**With keys, React would actually re-order the DOM elements**, instead of rendering a lot of nested DOM changes. This can serve as a huge performance enhancement, especially if the DOM and VDOM/React Elements being used are costly to render.

***What are functional components***

A syntax to write components using functions rather than classes. State and lifecycle methods are not availble in functional components however Reacts Hooks provide us a way accessing both of em in functional components

function Date(props){

let {msg="The date is:"} = props

let now = new Date()

return <div>

<span>{msg}</span>

<time>{now.toLocaleDateString()}</time>

</div>

}

This function that returns a React Element can be used whereever we see fit:

**wat is propTypes object**

in Component.propTypes object we tell react about wat shud be the type of props and whether they are required (compulsory) or not

**18. When you will use the class component over a functional component?**

When your component carrying a state or lifecycle then we will use the Class component.

**y is setState async in react?**

**setState**() actions are asynchronous and are batched for performance gains. setState() does not immediately mutate this.state but creates a pending state transition.

This is because setState alters the state and causes rerendering. This can be an expensive operation and making it synchronous might leave the browser unresponsive. Thus the setState calls are asynchronous as well as batched for better UI experience and performance.

***<Unknown> in react-devtools.***

When a component is exported via an anonymous function



It shows up as <Unknown> in react-devtools.

**React Hooks**

hooks allow us to use STATE or LIFECYCLE METHODS in functional components . e.g: useState, useEffect etc. same hook may be used multiple times in same react component.

Hooks are a way to reuse stateful logic, not state itself. In fact, each call to a Hook has a completely isolated state — so you can even use the same custom Hook twice in one component.

**useEffect**, adds the ability to perform side effects from a function component. It serves the same purpose as componentDidMount, componentDidUpdate, and componentWillUnmount in React classes, but unified into a single API.

**Custom Hooks** are more of a convention than a feature. If a function’s name starts with ”**use**” and it **calls** other **Hooks**, we say it is a custom Hook.

**MapStateToProps**

**Using connect creates a new component. That’s why it’s called a High Order Component.**

This new component calls our mapStateToProps function and our MyComponent that we passed to connect.

This new ConnectedComponent has access to the Redux Store, that’s why it can use getState to get the state and pass it when calling our mapStateToProps function.

// it maps the stores state values to props of component

***useEffect without dependency array ?***

This means useEffect runs on every render and thus the event handlers will unnecessarily get detached and reattached on each render

***useEffect with empty dependency array ?***

it would run once only

***What is thunk***

Redux-Thunk is the most popular middleware used to handle asynchronous actions in Redux.  
in normal redux flow actions are synchronous and we use THUNK to make them async

***What is HOOK in react***

Hooks are functions that let you “hook into” React state and lifecycle features from function components

***How to lazy load components in react***

CSS  
 universal selector  
inline vs block element

diff b/w nosql and sql