**React – Basics**

**Install node.js and react-app**

**npx create-react-app my-app** : **npx** – not installed globally like older version, it is xplicitly used for creating a react app

Install

* node.js ,
* vscode,
* gitbash,
* react and
* yarn;

**Create-react-app :**

npx create-react-app my-app;

cd my-app;

yarn start || npm start

**React Project set up :**

**yarn run build || npm run build** creates a deployable code inside build folder

**Babel and Webpack**

**Babel** takes our source folder(with weird syntax) and makes it understand for all the browsers

**Webpack** is a module bundler : bundle all our files and optimize it

**import React from 'react' :** the package that allows us to **write jsx , build components and build virtual dom**

**import ReactDOM from 'react-dom':** the little robot that interacts **with the DOM**

**Class Components:** here **App is a component -> app component**

Classes that returns html (react)

**import React,{Component} from 'react';** // import component

**state : js object with properties** that can be accessed at any point inside the class

Create a class

**Class App extends Component{**

**render(){**

**return(**

**<div>**

**</div>**

**)**

**}**

**}**

**export default App;**

**<p>**

**{this.state.name} // {} – jsx – tells html anything btw{} is js and render js**

**</p>**

**<button onClick = { () => this.setState( { name : "welcome MSK" } ) } > hi </button>**

**setState() -> change the value of state**

**access this.state**

**class App extends Component {**

**constructor(){**

**super();**

**this.state={ // setting properites**

**name:"msk",**

**}**

**}**

**render(){}**

**}**

***Note:*** *acutally html here is not actual html , it is JSX ; JSX mimics html*

*<div* ***className****=’app’> //not class*

*<button* ***onClick****={}>btn</button> //camel case because its JSX (javascript)*

**Dynamic content & using** **map():**

**var newarray = myarray.map(function);**

map() iterates over an array and creates a new array that has had each item passed through a function:

this.state={

**monster:[**

**{**

**name:'frankenstein'**

**},**

**{**

**name:'Dracula'**

**},**

**]**

}

Inside render () -> **this.state.monster.map(monster => <h1>{monster.name}</h1>)**

**Fetching content: LifeCycleMethods** = methods that called at diff stages when component is rendered

**componentDidMount();** //Life cycle method. When components mounts it calls the block of code

**fetch()** //returns a promise

**componentDidMount(){**

**fetch('https://jsonplaceholder.typicode.com/users')**

**.then(resp => resp.json())**

**.then(users => this.setState({monster:users}));**

**}**

**Card-list component:**

**Card-list.style.css**

.cardlist{

width:85vw;

margin:0 auto;

display:grid;

grid-template-columns: 1fr 1fr 1fr 1fr ;

grid-gap:20px;

}

**Card-list.component.js**

import React from 'react';

import './card-list.style.css';

**export const CardList = (props)=> {**

**return(**

**<div className="cardlist">**

**{props.children}**

**</div>**

**);**

}

**App.js**

<div className="App">

<CardList >

{

**this.state.monster.map(monster =>**

**<h1 key={monster.id}>**

**{monster.name}**

**</h1>**

**)**

**}**

**</CardList>**

</div>

**Key** – *it should be provided to let react identify individual h1s here*

**Props**

**<CardList name="ssk" id={this.num} > // get this using props.name and props.id**

<h1> hi</h1> **// get this using props.children**

**</CardList>**

**Card-Component:** create a card component with display flex and transition and transform

**Card.js**

export const Card = (props) =>{

return(

<div className="cardcontainer">

**<img alt="Moster" src={`https://robohash.org/${props.monster.id}?set=set2`}/>**

<h2>{props.monster.name}</h2>

<p>{props.monster.email}</p>

</div>

);}

**Breaking into components:** as small as possible

**State vs props** : states are sent as properties to other components. It can be placed anywhere

**Search-Field:**

**To avoid asynch miss-match use 2nd arg() of set state**

**onChange**={ e =>{//synthetic event not actual html it is react-dom event

this.setState({searchField:e.target.value**},()=>{**

console.log(this.state)

**}**); **// second prop - > call back function**

}} // set state calls are bad practice

**On input change state is set**

onChange={ e =>{

this.setState({searchField:e.target.value});

console.log(this.state)

}}

**this.setState is asynchronous . Executes after console.log**

**React-Events:** react -> synthetic events

**Filtering state:**

const {monster ,searchField} = this.state; // de-structuring objects into constants

const filteredMonsters = **monster.filter**(monster => { //filter function is used to filter data from array

return monster.name.toLowerCase().**includes**(searchField.toLowerCase());

}); //includes () to check whether the array has the same data

<CardList monsters={filteredMonsters}/> // pass filtered data to cardlist component

**SearchBox**:

**App.js**

<SearchBox **placeholder={'search monsters'}** **handleChange={e=>this.setState({searchField:e.target.value}**)} />

**SearchBox.js**

export const SearchBox = **({placeholder,handleChange}**) =>{ //destructuring props

return(

<input className='search' type="search" placeholder= {placeholder} **onChange= {handleChange**} />)

}

**Where to put state:** *it should be above the node that shares (and not to nodes that doesn’t need)*

**Class methods and arrow functions:**

**Arrow functions (lexical scoping)**

**handleChange = (e) =>** {

this.setState({searchField:e.target.value});

} **// this keyword is withing the context of the place where this function is defined (i.e in the APP class)**

handleChange(e){

this.setState({searchField:e.target.value});

} **// this refers to window and not App class**

**Inside constructor():**/ so bind externally and bind returns function

**this.handleChange = this.handleChange.bind(this);**

**Git hub page** : refere git-hub document

**React and React-DOM:** you can create your own index.html file with the help of react libraries from online src

**Virtual DOM + unidirectional data flow:**

**Actual DOM and virtual DOM ->changes in Virtual DoM is compared with actual DOM -> renders particular part alone**

Note : to see which part is currently being updatedbrowser 🡪developer tools 🡪 more tools 🡪 rendering 🡪enable pain flashing

**Unidirectional dataflow** - > flows from **state 🡪 views 🡪 actions (changes) 🡪 state** (loop like structure in one direction)

**Asynchronous setState: state updates are asynchronous**

**setState - >** changesdoes not occur immediately (asynchronous 🡪passes the control outside and there is a delay )

When multiples setStates are there , react tries to combine as a batch and delay may occur

**Alternate class syntax**: //not in js

class App extends Component {

**state={**

**num :10;**

**}**

}//state can be used like this directly bcos of create react app and babel compiler

**<App increment={1}/>**

this.setState**((prevState,prevProps)=>{ return {num:prevState.num+ this.prevProps.increment}**

},

**Or if you want to use in constructor**

class App extends Component {

**constructor(props){**

**super(props)**

this.state={num + this.props.increment)

**this.props=props;**

}}

**React rule :** *to use data after update*

**if you want to use this.state or this.props directly inside setState to calculate something , use function as first parameter in setState**

handleClick = () => {

this.setState**((prevState,prevProps)=>{** // prevState = this.state (as args)

**return {num:prevState.num+1}** //

},

() => console.log(this.state.num) // second parameter

)

}

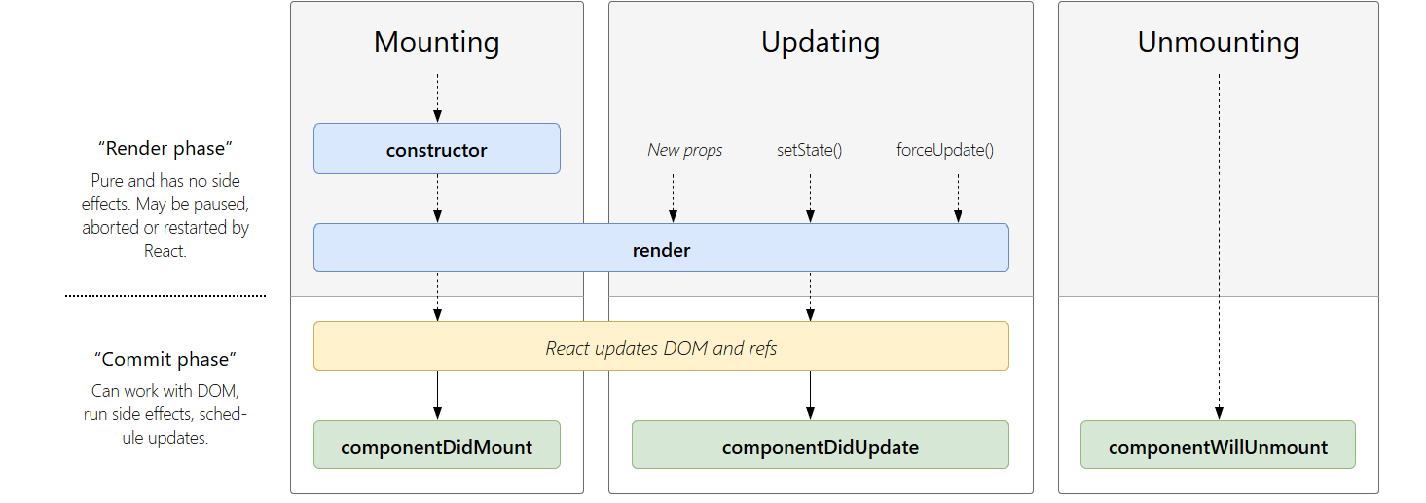
**Life cycle methods:**

Let change = true;

!change => ! Operator changes the Boolean value from true to false or false to true

**Mounting 🡪**Constructor 🡪Render 🡪ComponentDidMount

**Updating**🡪 Render🡪ComponentDidUpdate



**state / new props / forceupdate -> when changes are there the whole component changes and the child component which doesn’t need change also gets rendered (which is waste) . So use**

**shouldComponentUpdate(nextProps,nextState){ // nextProps -> receives the props that is changed**

console.log(‘shouldComponentUpdate!’,nextProps); // nextState - > receives changed state

return true/false; // true indicates that the component should re render , false not to render

ex:

**return nextProps.text !== this.props.text;**

} // render if there is a change in text props value that is coming in

**ComponentWillUnmount - >** when rerendering if a component is not going to be used it is unmounted and this function is called while unmounting