REACT JS

What is React?

It is a JavaScript Library and not a framework, which is used to design the user interface.

It is a client side,front-end ,javascript library

Using JS alone is not good option because

The code becomes

- Cumbersome (heavy)
- error prone
- maintaining is not easy

Modules is for breaking the code for our developing convenience

It can be achieved with the help of **import** and the **export** keyword

It is again divided into **2 types** as of **NAMED** and **DEFAULT import/export**

NAMED:(Single variable)

Syntax for export

export var/let/const variableName/functionName ="value";

Syntax for import

import {functionName / variableName} from"the path of the js file ";

NAMED:(Multiple variable)

Syntax for export

export var/let/ const varibleName1 = "value";

export var/let/ const varibleName2 = "value";

export var/let/const varibleName3 = "value";

Syntax for import

import {varibleName1,varibleName2,varibleName3} from "the path to the js file ";

DEFAULT:

PRE-REQUISITE

There can only be one default value and it is a must

Syntax for export

export default "value" ;

Syntax for import

import variableName from "the path of the js file";

COMBINATION:

Syntax for export

export default vname ="value";

export var/let/const vname1="value";

export var/let/const vname2="value";

export var/let/const vname3="value";

<u>Syntax for import</u>

import * as util from "the path to the js file";

to acces it use it from util followed by dot/period name;

‘as’ can also be used for **alias** name for the named type too

Eg:

import {name as n} from "the path of the js file";

<u>DESTRUCTING METHOD</u>:

<u>Old method</u>

const arr = [Reactjs,Nodejs];

const f = arr[o];

const s = arr[1];

console.log(f,s);//Reactjs,Nodejs

<u>New method</u>

const [f,s]=[Reactjs,Nodejs];

console.log(f,s);//Reactjs,Nodejs

For Objects:

```
const user={

name : "SudhagarV",

id : 1

}

console.log(user.name,user.id);//SudhagarV,1


const {userName ,id : userId}={ //userId is the alias name and it is the syntax to be followed

name : "SudhagarV",

id : 1

};

console.log(userName,userId);
```

Difference between SinglePage Application(SPA) and MultiPage Application(MPA)

SINGLE PAGE APPLICATION	**MULTI PAGE APPLICATION**
It contains only one HTML page	It contains more than one HTML page
It will not get refresh and reload	Page reloading and refreshing will occur in MPA
SPA is a modern approach	It is a traditional approach
SPA are moderate or small applications	MPA are huge applications
Eg- ReactJS FB,Insta,Swiggy,Netflix,Zomato	Eg- All e-kart and banking applications

JavaScript is having libraries and frameworks

LIBRARIES :

- They are collection of codes.
- Libraries performance is more compared to frameworks
- Applications developed with JS Libraries will have more efficiency and speed.

Eg- ReactJS , JQuery ,Aurella etc

FRAMEWORKS:

- They are collection of libraries.
- Performance is less compared to libraries.
- App developed with JS Frameworks will have less efficiency and speed.

Eg- Angular, VueJs etc.

REACT JS

- React is a JavaScript library which is used to built user-interface.
- It is very fast because of virtual DOM (V-DOM)
- It follows Component based approach which helps in builiding UI components
- It follows uni-directional data flow
- It follows Server-side rendering

HISTORY

It came into existence in 2011 and was only used by Facebook .Later in 2013 Facebook made it as open source.

React is developed by Jordan Walke.

First it was developed and integrated in news feed feature in Facebook.

React is used to built moderate and small applications like Swiggy and Zomato

Current version of ReactJs is 18.2.0(Major Version.Minor Version.Patch Version).

ADVANTAGES

It can be conveniently used on client as well as server (it follows server side rendering)

It increases application performance

Because of Jsx, code readability increases.

The React is easy to integrate with other frameworks like Angular

MODULE:

A module in JavaScript is just a file containing related code.

In JS we use the **import** and **export** keyword to share and receive functionalities respectively across different modules

The export keyword is used to make variables , functions accessible to other modules.

Java Script modules is divided into 2 types

1)CommonJs modules

2)ES6 modules

COMMON JS

It is mainly used in server side JavaScript apps with NodeJs, as browser doesnot support the use of CommonJs.

We are going to export a file in CommonJs by using module.exports and imported by using require( ) method

App.js

```
//!commonjs module

const value= require("./main");

const array= require("./child");

console.log(value.x);//1000

console.log(value.test( ));// "function is executing"

console.log(array.arr)//[1,2,3,4,5,6,7,8,9]

array.arr.map( val=>{

console.log(val)}
```

<u>Main.js</u>

```
let x=1000;

function test ( ){

return "function is executing"

}

module.export{

x,

test

}
```

<u>Child.js</u>

```
let arr=[1,2,3,4,5,6,7,8,9];

module.export={

arr

}
```

ES6 Modules (Ecma Script) ECMA-European Computer Manufacturing Association (initially owned by Netscape Navigator)

- It is a standard that was introduced with ES6 feature (2015).
- ES Modules is a modern approach that is currently supported by browser for exporting and importing the JavaScript files

Exporting in a module can be classified as follows.

1)Named Export

2)Default Export

DEFAULT EXPORT

Default export can export only one variable per file

Eg:

Main.js

```
let user=' Sudhagar ';

export default user;
```

App.js

```
import user from './main.js'

console.log(user); //Sudhagar
```

NAMED EXPORT

It is more than one variables or multiple variables.

Main.js

```
let obj ={ name: "Sudhagar",

company:"QSP"

}

let arr=[1,2,3,4,5];

export{obj,arr};
```

App.js

```
import {obj,arr} from './main.js'

console.log(obj);

console.log(arr); //1,2,3,4,5
```

NAMED & DEFAULT

Main.js

```
let user='Harish';

let obj={name:"Bharath",

company:"QSP"}


let arr=[1,2,3,45,];

export user,{obj,arr};
```

<u>App.js</u>

```
import user,{obj,arr} from './main.js'

console.log(obj);

console.log(arr);

console.log(user);
```

<u>INSTALLATION OF REACT</u>

There are 2 ways to install it

<u>1st way</u>

CDN(Content Delivery Network)

<u>2nd way</u>

NPM way (Node Package Manager)

to **install react in npm manager** we have command line interface

```
npm install create-react-app --globally

 npm init react-app  (application  name) [this is not working currently]

                       (OR)

npx create-react-app my-react-app

cd my-react-app
```

npm start [to run the web app]

Rules while creating react folder name

- We should not use react name for the folder.
- Folder name should always be **small letter**.

1) What is **CLI**?

CLI is a command line interface or program, that accept text input to execute operating system functions.

2) What is create react app command?

- create react app is a tool to create single page react application that is officially supported by the react team.
- the command generates the required files and folders to start the react application and run it on the browser.

FOLDER STRUCTURE IN REACT APPLICATION

- Node modules
- public
- src
- package.json

<u>Node module</u>

Node modules folder is used to save all downloaded packages from npm in the project.

<u>Public folder</u>

The public folder contains a static files such as index.html , images , video , audio , and other asserts.

<u>SRC</u>

- Src is the actual react folder where content will be returned to display on the UI. In this folder there is a main file called index.js will be there as root react file.
- without index.js react application will not be open , it will throw the error that could not find the required file name index.js

<u>package.json</u>

- It is the heart of Node JS. It holds all the dependencies.
- It is a container for registry data which holds versions , descriptions
- it records important meta data about a project.

<u>RECONCILATION</u>

The process of updating the real dom with the virtual dom is called as reconcilation. The entire dom is not scanned only the added thing is updated in the real DOM because of this ,the react is faster.

<u>**VIRTUAL DOM**</u>

In RealDOM every time the content loads, the entire document is loading from the beginning while in virtual DOM the particular new element will be loaded and patch with real DOM.

A virtual DOM object has the same poperties as a REALDOM objects.Virtual DOM cannot change directly on the screen.

Manipulating real DOM is slow whereas manipulating the virtual DOM is much faster because nothing gets drawn on screen.

<u>**Virtual DOM (definition)**</u>

It is a programming concept where an ideal or virtual representation of a UI is kept in memory and synced with the real DOM by a library such as REACT DOM. This process is called as **reconcilation** .

<u>**V-DOM(general definition)**</u> Virtual DOM is nothing but updating/patching the real DOM with the help of DIFFING ALGORITHM is called as reconciliation

<u>**DIFFING ALGORITHM**</u>

- It is used to differentiate the DOM tree for efficient updates.
- React utilise diffing algorithm to identify the changes in the newly created virtual DOM and previous version of real DOM after any changes are made.

React Dom has a method called **render** which is in sync with virtual DOM to real DOM to print on UI.

This **render method** is having 3 arguments

1. Element
2. Container
3. Callback function

Element is used to print contents on the browser

Container is used to store the elements

Element and container is **mandatory** arguments whereas **callback** is an **optional** and is used for **checking** whether the **DOM** is **connected or not.**

React DOM connect only once.

Eg- ReactDOM.render (<h1> Hello </h1> . document.getElementById('root').( ) =>{

console.log("DOM CONNECTED") });

Render - React elements with the help of React.createElement( );

<u>Creating an object in different ways</u>

<u>HTML</u>

```
<div id='demo'>

<h1>Hello Header</h1>

<p>Lorem Ipsum.....</p>

</div>
```

<u>JS way</u>

```
let div= document.createElement('div');

div.setAttribute('id','demo');

let h=document.createElement('h1');

h.innerText='Hello world';

let p= document.createElement('p');

p.innerText='Hello World';


div.append(h,p);

document.body.append(div);
```

<u>React</u>

```
React.createElement('div',{id:'demo'},React.createElement('h1',null,'Hello
header'),React.createElement('p',null,'Lorem Ipsum.....')));
```

CreateElement( ) in react having 3 arguments

1)Type of element

2)Properties of an element

3) Children or contents of the element

Eg:

```
import React from 'react';

import ReactDom from 'react-dom';


let element = React.createElement('div',{id:'demo'},React.createElement('h1',null,'Hello header'),React.createElement('p',null,'Lorem Ipsum.....')));


ReactDOM.render(element,document.getElementById('root'),( )=>{

console.log( 'DOM Connected');

});
```

JSX

It is a JavaScript XML which is a template language or a technique used for react Application.

Babel is a Javascript compiler used to next generation Javascript today

Eg-

```
<div>
<h1> Hello Header </h1>
<span>Span tag</span>
</div>
```

RULES OF JSX

In ReactJsx every tags include atg and unpaired tags

If more than one JSX element we should wrap inside parent element

Eg-

```
<div>
<h1> </h1>
</div>
```

FRAGMENTS

React Fragments allows you to wrap or group multiple elements without adding extra node to the DOM.

Eg:

```
< >
<h1>Hello</h1>
<span> Span </span>
< />
```

JSX Expressions

Jsx expressions is wrapped with curly braces where content between opening and closing curly braces will be evaluated as JavaScript expressions.

We can call variables inside the JavaScript expressions.

Eg-

```
let value = 1000;

let arr=[‘Sudhagar’,Caesar,'Baranee','Allius'];

let iterate = arr.map((val,i)=>{return val});


ReactDOM.render(< >

<h1> {100*200} </h1>

<h1> {value} </h1>

<h1> {iterate} </h1>

))}

</>);
```

Components (or) reusable building blocks

Components are independent and reusable bits of code.

Components serve the same purpose as javascript functions but work in isolation written in Jsx.

Components are divided into 2 types

1) Class based component (ES6 class)

2) Functional based components (JS functions)

CLASS BASED COMPONENTS

Class based Components are used to create a component where state and lifecycle can be used to perform actions for building react applications

Eg-

```
import React,{Component} from 'react';


class App extends React.Component{

render( ) {

return(

< >

<h1>Class Based Component</h1>

</>

)

}

}

export default App;
```

<u>FUNCTIONAL BASED COMPONENT</u>

A functional component is just a plain javascript function which returns React elements

Eg-

```
import React from 'react';

function App(){

return(

<> <h1>Functional Based Componetns </h1> </>

);}


const App=( ) =>{

return(

< >

<h1> Arrow Function based Component</h1>

</>

)

}


export default App;
```

CLASS BASED STATES

```
import React,{Component} from 'react';

import CBCChild1 from './CBCChild1';

export default class CBCProps extends Component {

state = {isLoggedIn : true}

render (){

console.log(this.props);

return (<div>

<h2>{this.props.username}</h2>

<h2>{this.props.sal}</h2>

<h2>{this.props.eid}</h2>

<h2>{this.props.company}</h2>

</div>)

 }

}
```

FUNCTIONAL BASED COMPONENTS

A Functional based components is just a plain Js function which returns react elements

```
Function App(){
return{
< >
<h1> Functional based components </h1>
</>
}
}
export default App;
```

ARROW FUNCTION:

```
import React from 'react';
const App =()=>{
return(
<>
<h1> Functional based components </h1>
</>
);
}
export default App;
```

STATE:

The State is the built-in react object is used to contain data or information or isolation of data above the components.

A State can be modified based on user action or network changes

Every time the state of an object changes react renders the component to the browser.

The state object can be stored multiple properties.

State is a mutable object and it cannot be shared directly to another components

STATE IN CLASS BASED COMPONENT:

In class based component the state object initialized in the constructor

```
constructor (props){

super (props);

this.state ={

users : []

};
```

```
state ={

user:[ ]

count:0

};
```

This state in the class based component can **store only object or null.**

SET STATE METHOD:

this.setstate method is used to change the value of the state object.

set state method performs is shallow merge between new and the previous state.

```
state ={
count:0
}


increment =()=>{this.setstate({count:this.state.count + 1})}


<h1>{this.state.count}</h1>


<button onClick = {this.increment}>Increment</button>
```

DEFAULT PROPS:

In some cases,you might want to provide default values for props in case they aren't explicitly passed.

This ensures your component doesn't break due to missing data.

}

App.jsx

```
import React from 'react';

import Child from './';


const App=()=>{

return ( <div>

<Child username='xx' sal={1000}/>

</div>)

}


export default App;
```

Child.jsx

```
import React from 'react';

const Child =(props)=>{

return(<>

<h1>{props.username}</h1>

<h1>{props.company}</h1>

</>)

}

export default Child;
```

```
Child.defaultProps{

username :'AAA',

company : 'ABC'

};
```

PROPTYPES:

- PropTypes are simply a mechanism that ensures that the passed value is of the correct datatype.
- This makes sure that we dont receive an error at the very end of our app by the console which might not be easy to deal with.
- It serves as the method you can use to ensure the correct datatype is passed for each prop.
- It allows developers to validate and enforce data integrity throughout the application.
- It can be used for validating a props are number, string, boolean , null , undefined , array ,object etc..

HOW TO USE PROPTYPES?

We should install proptypes by using -----> npm i prop-types

import PropTypes from 'prop-types'

It is to specify the data type of the title, we'll create an object and assign a key-value pair for each prop.

```
import React from 'react';

import PropTypes from 'prop-types';

const Child =(props)=>{

return(

< >

<h1>{props.username}</h1>

<h1>{props.salary}</h1>

<h1>{props.isLog}</h1>

< / >

)}

export default Child;
```

//prop validation

```
Child.PropTypes={

username : PropTypes.string,

salary : PropTypes.number;

is Log : PropTypes.bool;

}
```

<u>LIST AND KEYS:</u>

<u>KEY:</u>

A key is one of a kind identifier.It is used in react to determine which list items have changed, been updated or been removed.It is beneficial when we have added components dynamically or when users change the lists.

Eg:

```
Const arr = [10,20,30,40,50];

Const updated arr= arr.map((val,index)=>{

Return <li key={index}>{arr}</li>

})
```

<u>CONDITIONAL RENDERING:</u>

Your components will often need to display different things depending on different conditions. In React,you can conditionally render JSX using JavaScript Syntax like if , if else, switch, ternary conditions etc.

Eg:

```
Import React from ‘react’;

Const isLoggedIn = true,

Const msg = (props) =>{

If (isLoggedIn) {

Return <h1> Welcome </h1>

Else {

Return <h1> Login First </h1>

} } }
```

<u>TERNARY CONDITIONS</u>

```
Return(
<div> {isLoggedin ? <login/> : <Landing/>} </div>
)
```

<u>SWITCH CASE:</u>

```
Switch (isLoggedin){
Case true : return <div> <login/> </div>
                Break;
Case false : return <div> <login/> </div>
                Break;
}
```

<u>KEYS:</u>

When it comes to building a website's UI, list are extremely useful to display menus on a website , such as navbar menu.

Arrays can be used to create lists in normal JavaScript. In React,we can make lists in normal JavaScript. In React we can make lists in the same way as in normal JS

The traversal of lists is done with the map() function. The map() va;ues of an array of numbers added to 10 in the range below:

```
Var arr =[10,20,30,40,50];
Const arrr= arr.map((num)=>{return (num + 10);})
Console.log(arrr); // [20,30,40,50,60]
```

ERROR BOUNDARIES

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.

A class component can become a error boundary if it defines a new lifecycle methods either static getDerivedStateFromError() or componentDidCatch(error,info).

We can use static getDerived StateFromError() to render a fallback UI when an error has been thrown, and can use componentDidCatch() to log error information.

REACT FRAGMENTS

React Fragments allow you to wrap or group multiple elements without adding an extra node to the DOM.

This can be useful when rendering multiple child elements/components in a single parent component.

Example:

```
Function Column( ){

Return(

<div>

<td> Hello World</td>

</div>

)

}
```

```
Function Table(){
Return(
<table>
<tr>
<Column/>
</tr>
</table>
)
}
```

If we wrap the <td> elements in a <div>,this will add a <div> element in the middle of <tr> and and <td> and break the parent child relationship.

To overcome this we use React Fragments which wrap around child elements without adding an extra DOM element.

EX:

```
Function Column( ){
Return(
<React.Fragments>
Components
</React.Fragments>
)
);}
```

REACT-HOOKS (BASIC HOOKS)

React Hooks are simple JavaSript functions that we can use to isolate the reusable part from a functional components.

Hooks can be stateful and can manage side-effects.

It was introduced in 16.8 version of **reactJs**

5 IMPORTANT RULES OF HOOKS

- Never call Hooks from inside a loop, conditional or nested Function.
- Hooks should always sit at the top-level of your components
- Only call Hooks from React functional components
- Never call a Hook from a regular JavaScript function
- Hooks can call other Hooks,yes it is possible.

The 3 Basic hooks are
useState( )
useEffect( )
useContext( )

USESTATE( ) :

It is a React Hooks that allows you to add state to a functional component.It returns an array with two values : the current state and a function to update it.
The Hook takes an initial state value as an argument and returns an updated state value whenever the setter function is called.It can be used like this:

```
const [ state , setState ] = useState (initialValue);
```

Here the initialValue is the value you want to start with and state is the current state value that can be used in your component. The setState function can be used to update the state, triggering a re-render of your component.

```
import React ,{ useState } from ‘react’;

Const Message = ( ) =>{

const  [message , setMessage] = useState( ‘ HI’ );

let handleClick = ( )=>{
setMessage(‘HELLO’);
}

return(
< >
<p>
<strong> {message} </strong>
</p>
<button onClick={handleClick}> Click</button>

</ >
);

}
```

<u>USE EFFECT</u>:

useEffect () is a hook that manages side effects in functional react components.

That sideEffect may be of

1. Making a request to an API for data from a backend server.
2. To interact with browser APIs (that is, to use document or window directly).
3. Using unpredictable timing functions like setTimeout or setInterval.

useEffect() hook accepts 2 arguments:

useEffect(callback, [dependencies]);

- Callback is a function that contains the side-effect logic. Callback is executed right after the DOM update.
- Dependencies is an optional array of dependencies .

```
import React ,{ useState, useEffect } from ‘react’;

Const Message = ( ) =>{

const  [message , setMessage] = useState( ‘ HI’ );

const [state,setState]=useState(‘Name’);

useEffect(( )=>{
document.title = state
},[ state] ) ;
let handleClick = ( )=>{
setMessage('HELLO');
}

return(
< >
<p>
<strong> {message} </strong>
</p>
<button onClick={handleClick}> Click</button>

</ >
);
}
```

Use Context :

Context in React provides a way to pass data through a component free without the need to prop-drill (i.e., pass props down manually at every level).

To begin, we first need to create the Context object with th createContext( ) function.

```
Import React , {createContext} from ‘react’;

Const Context = createContext();

Const App = (props) =>{
Return(
<Context.Provider value={{data : ‘Data from Context’}}>
{props.children}
 </Context.Provider>

)
}
```

Context.Provider will accept only one value that is value

To access value from the context Provider by using a hook called useContext( ).

The useContext Hooks provides function components access to the context value for a context object.

SYNTAX:

```
 Const contextValue = useContext( contextObject);
```

```
Import React, {useContext} from ‘react’;

Import {Context} from ‘./App’;

Const Child = ( ) =>{

Const context = useContext(Context);

Return <div> {Context.data}</div>

};

Export default Child;
```

HIGHER ORDER COMPONENT

It is also known as HOC.

It is an advanced technique for reusing component logic.

It is a function that takes a component and returns a new component.

They are similar to JavaScript functions used for adding functionalities to the existing component.

It is used to overcome the **props drilling**

EG:

Hoc.js

```
Import React,{Components} from ‘react’;

Export default function HOC(hoc component){

Return class extends Component{

Render(){

Return(
```

<div>

<HOCComponent> </HOCComponent>

</div>

)

}

}

}

App.jsx

Import React from ‘react’;

Import {HOC} from ‘./Hoc’;

Const App =( ) =>{

Return(

<div></div>

)

}

Export default HOC(App)

Ref:

Refs in React are short for references.

As the name suggests they allow you to reference and interact with DOM nodes or React components directly.

Refs come in handy when you need to reference some information in a component,but you don’t want that information to trigger new renders.

Common use cases of React refs include:

- Managing focus, text selection, or media playback
- Triggering imperative animations
- Integrating with third-party DOM libraries.

CREATING REFS USING THE createRef HOOK

To create refs in class component, we use React.createRef( )

```
Import React,{Component} from ‘react’;

Class ActionButton extends Component{

Constructor(props){

Super(props);

This.buttonRef = React.createRef( );

}

handleClick = ( ) =>{

console.log(‘this.buttonRef.current’);

}

};
```

CREATING Refs USING useRef Hook

```
Import React,{useRef} from 'react';

Function ActionButton( ){

Const buttonRef = useRef( null);


        Function handleClick(){

Console.log('buttonRef.current');

       }

Return(

<button onClick={handleClick} ref={buttonRef}> CLICK </button>

)

}
```

FORWARD REF

forwardRef is a utility function that passes down a ref through a component to one of its chidren.

This is particularly useful when you need to access a DOM element or component instance directly in a parent component.

EG:

```
Import React ,{forwardRef,useRef} from ‘react’;

Const App =( ) =>{

Let demoref =useRef( );

Let handleClick = ( )=>{

demoRef.current.style.color =’red’;

}

Return(

<div> <Child value={handleClick}

Ref={demoRef}/>


 </div>

)

}

Export default App


Const Child = forwardRef(

(props , ref )=>{

Console.log(props);

Return(
<h1 ref={demoRef}> Header</h1>

<Button onClick ={props.value}>  Change color  </button>

) } )
```

<u>TO INSTALL TOASTIFY :</u>

toastify – npm i react-toastify

<u>FORMS</u>

<u>WHY REACT FORMS ?</u>

Handling React Forms is an important part of many online web applications and it's one of React's strong suits. You have a lot of flexibility in implementing and controlling those input Controls. And there are many other approaches to accomplish the same purpose.

In react we have 2 types of components

1) Uncontrolled Component
2) Controlled Component

<u>Uncontrolled Component :</u>

It refers to component that manages their own state internally.

They use a ref to access the DOM element's current value and update the state accordingly.

They are completely handled by DOM.

EG:

// FileName - App.js

```
import React, { useRef } from "react";

import "./App.css";

function App() {

	const inputRef = useRef(null);

	function handleSubmit() {

		alert(Name: ${inputRef.current.value});

	}

	return (

		<div className="App">

			<h1 className="geeks">GeeksForGeeks</h1>

			<h3>Uncontrolled Component</h3>

			<form onSubmit={handleSubmit}>

				<label>Name :</label>

				<input

					type="text"

					name="name"

					ref={inputRef}

				/>

				<button type="submit">Submit</button>

			</form>
```

```
                </div>   ) ; }
```

export default App;

CONTROLLED COMPONENT :

In React, Controlled Components are those in which form’s data is handled by the component’s state.

The Controlled Component is a state-based method of handling form imput. If you are using React Hooks, you may alter the form input value in a single step, and when the user starts typing setState or useState characters,this state can be called and updated using a single event, such as an onChange.

EG:

```
Const LoginForm = ( ) =>{

Const [email , setEmail] = useState( “ ” );

Const [password , setPassword] = useState( “ ” );

Let handleSubmit = ( e) =>{

e.preventDefault( );

clg({email,password});

}

Return(
<form onSubmit = {handleSubmit}>

<input type = ‘email’ value={email} onChange={e =.> setEmail(e.target.value)} placeholder
= ‘Email’ / >

<input type=’password’

Value={password}
onChange={e => setPassword (e.target.value)}

placeholder = ‘Password’

/ >

<button type=’submit’ > Submit </button>

</form>

);

};
```

<u>REACTJS -FORM</u>

Forms are a common part of web applications which are mainly used allow users to interact with the application. These forms maybe included on the web page to gather information of the user, to let the user search a website, to make payments etc.

*The basic elements a form can contain are input fields, buttons, checklists, drop-down menus and so on. The data obtained from these forms is usually handled by components in React.

React provides two types of form

Controlled component

Uncontrolled component

1**<u>.Controlled component :</u>**In controlled component, React provides a special attribute, value for all input elements and controls the input elements with state of a component.

*The value attribute can be used to get and set the value of the input element. It has to be in sync with state of the component.

*An input form element whose value is controlled by React State is called a "controlled component".

ex:-refer the codes

```
        <input type="text" name="username" />
        this.state = {
                        username: ''
                       }
        <input     type="text"     name="username"     value={this.state.username}
onChange={this.handleUsernameChange} />
        handleUsernameChange(e) {
  this.setState({
    username = e.target.value
  });
}
```

2**.Uncontrolled component:** a React DOM element (form element) is not possible without using React api. One way to get the content of the react component is using React ref feature.

React provides a ref attribute for all its DOM element and a corresponding api, React.createRef() to create a new reference (this.ref). The newly created reference can be attached to the form element and the attached form element's value can be accessed using this.ref.current.value whenever necessary (during validation and submission).

ex:---this.inputRef = React.createRef();

```
<input type="text" name="username" ref={this.inputRef} />
handleSubmit(e) {
  e.preventDefault();
  alert(this.inputRef.current.value);
}
```

<u>Routing</u>

In web application, Routing is a process of binding a web URL to a specific resource in the web application. In React, it is binding an URL to a component. React does not support routing natively as it is basically an user interface library. React community provides many third party component to handle routing in the React application. Let us learn React Router, a top choice routing library for React application.

<u>Install React Router</u>

*Install the react router using below command.

```
npm install react-router-dom
```

<u>React Router</u>

React router provides four components to manage navigation in React application.

1.<BrowserRouter>: BrowserRouter is a parent component in react-router-dom that stores all the other route components.

2.Router − Router is the top level component. It encloses the entire application.

3.Link − Similar to anchor tag in html. It sets the target url along with reference text.

```
<Link to="/">Home</Link>
```

Here, to attribute is used to set the target url.

4.Route − Maps the target url to the component.

for examples refer codes

STYLING

In general, React allows component to be styled using CSS class through className attribute. Since, the React JSX supports JavaScript expression, a lot of common CSS methodology can be used. Some of the top options are as follows −

1.CSS Stylesheet

2.Inline Styling

3.CSS Modules

1.**CSS Stylesheet**

CSS stylesheet is usual, common and time-tested methodology. Simply create a CSS stylesheet for a component and enter all your styles for that particular component.

2.**Inline Styling**

Inline Styling is one of the safest ways to style the React component. It declares all the styles as JavaScript objects using DOM based css properties and set it to the component through style attributes.

ex:- `<div style={{color:"red"}}>hello</div>`

3.**CSS Modules**

Css Modules provides safest as well as easiest way to define the style. It uses normal css stylesheet with normal syntax by creating a seperate file with module.css extension. While importing the styles, CSS modules converts all the styles into locally scoped styles so that the name conflicts will not happen.

```
ex: import styles from './ExpenseEntryItem.module.css'

                    <div className={styles.itemStyle}>
```

React Events

An event is an action that could be triggered as a result of the user action or system generated event. For example, a mouse click, loading of a web page, pressing a key, window resizes, and other interactions are called events.

React has its own event handling system which is very similar to handling events on DOM elements. The react event handling system is known as Synthetic Events. The synthetic event is a cross-browser wrapper of the browser's native event.

Handling events with react have some syntactic differences from handling events on DOM. These are:

React events are named as camelCase instead of lowercase.

With JSX, a function is passed as the event handler instead of a string. For example:

Event declaration in plain HTML:

```
<button onclick="showMessage()">

    Hello JavaTpoint

</button>
```

Event declaration in React:

```
<button onClick={showMessage}>
      Hello JavaTpoint
</button>
```

ex: In React, we can write it as:

```
function ActionLink() {
    function handleClick(e) {
        e.preventDefault();
        console.log('You had clicked a Link.');
    }
    return (
        <a href="#" onClick={handleClick}>
              Click_Me
        </a>
    );
}
```

React Code Splitting

The React app bundled their files using tools like Webpack or Browserfy. Bundling is a process which takes multiple files and merges them into a single file, which is called a bundle. The bundle is responsible for loading an entire app at once on the webpage.

As our app grows, our bundle will grow too, especially when we are using large third-party libraries. If the bundle size gets large, it takes a long time to load on a webpage. For avoiding the large bundling,we are using splitting in react.

Code-Splitting is a feature supported by Webpack and Browserify, which can create multiple bundles that can be dynamically loaded at runtime.

Code splitting uses React.lazy and Suspense tool/library, which helps you to load a dependency lazily and only load it when needed by the user.

React.lazy

The best way for code splitting into the app is through the dynamic import() syntax. The React.lazy function allows us to render a dynamic import as a regular component.

```
);

}
```

<u>Before</u>

```
import ExampleComponent from './ExampleComponent';

function MyComponent() {
  return (
    <div>
      <ExampleComponent />
    </div>
  );
}
```

<u>After</u>

```
const ExampleComponent = React.lazy(() => import('./ExampleComponent'));

function MyComponent() {
  return (
    <div>
      <ExampleComponent />
    </div>
  );
}
```

-----------------------------X--