.....JS Topics for learning react.....

1. ternary operators
2. arrow function
3. higher order and callback functions
4. map()
5. Spread Operator and Rest Parameter
6. OnClick() and OnChange()
7. Promise,Async and Await
8. Destructuring
9. Modules

.....React Topics.....

- Introduction.
- History.
- Key-Features.
- Reconcilation.
- Instalation using Vite.
- Understanding Folder Structure.
- Difference Between react and react-dom.
- Components.
- Difference Between Class Based and Function Based Component.
- jsx and its rules.
- Fragments and Expression.
- Hooks.
- useState().
- Props, Props Children, Default Props, Prop Types and Prop Drilling.
- useContent().
- Styling Components using Class.
- Form Handling using useRef(), useState(), useFormik().
- forwardRef().
- life Cycle Methods.
- useEffect{}.
- Axios.
- PureComponent.
- memo.
- portals.
- useMemo{}.
- useCallback{}.
- Router.

.....ReactJS Notes.....

Introduction

- React is a JavaScript library used for building the User Interfaces(UI) more efficiently.
- Through ReactJS we can build single page applications(SPA).
- Through React we can make use of reusable components.

Note:

- A library is a pre-written code that provides the methods to build a UserInterfaces(UI)
  ex:- ReactJS.
- Whereas framework is also pre-written code that provides entire structure of the UI.
  ex:- BootStrap, VueJS, AngularJS.
- Single Page Applications avoids unnecessary of reloading when switching between the pages of a website.

History

- React was developed by Jordon Walke in the year 2011.
- He was working as a Software Engineer at FaceBook.
- They Published about ReactJS first in the FaceBook newsfeed.
- Due to its popularity ReactJS was made open source in year 2013.

Key Features

React mainly has 3 key Features

1) Declarative Approach.

2) Component Based Approach.

3) Learn Once, Write Anywhere.

1) Declarative Approach

- Declarative Approach means not a direct approach.
- It focuses on describing the final output rather than step by step implementation.
- It makes the code more readable and easier to maintain by allowing virtual DOM to interact with the real DOM.

2) Component Based Approach

- It involves building the UserInterfaces by breaking it down into different parts.
- Each parts are reusable and has its own logic which helps to maintain the code easily.

3) Learn Once, Write Anywhere

- If we learn react, through ReactJS we can build web applications by which very easily we can learn reactNative which is used to build mobile applications.

Reconciliation.

- When browser renders our html document, it creates a tree like structure called Real DOM.
- Along with the real DOM, a light weight copy or clone of real DOM will be created that is virtual DOM.
- Whenever the changes are made in html document, virtual DOM gets destroyed and created once again with necessary changes.
- Virtual DOM keeps on comparing with the real DOM, this comparison is done by DIFFING ALGORITHM.
- Finding the changes and updating in the real DOM is known as patching.
- The entire procedure of creation of virtual dom, comparison and updation of necessary changes is known as Reconciliation.

Installation using Vite

1) Install nodejs.

2) Open command prompt and set the path to desktop.

3) Specify the command npm create vite@latest.

i - project name anything(foldername)

ii - framework:- React

iii - variant:- JavaScript

4) follow the command one by one

:: cd folder-name

:: npm Install

:: npm run dev

React Folder Structure

1. Node Modules - it is the folder which consists of all the dependencies required for our react application to run efficiently.
2. Public - This folder consists of static files that doesnot require much processing.
   ex-- images,svg files etc
3. src - It is the main folder where most of the code resides.
4. gitignore - This file specifies which folders or the directories should be ignored while uploading to github.
5. index.html - It is the main file that gets rendered on the user interface.
6. package.json & package-lock.json - Both of the files consist of the dependencies and their versions but package-lock.json will lock the versions of dependencies to avoid unneccessery updates.
7. Readme.md - It consist of the description regarding our react application.

React vs ReactDOM

1) React

- It is an object or library or dependency which is responsible for writing html like structure(jsx) into JavaScript file.
- Some of the methods present under react library are fragment, PureComponent, CreateContext, forwardRef, memo, useCallback, useContext,useEffect, useMemo, useRef, useState.

2) React-dom

- It is an object or library or dependency that allows to render the jsx on the UserInterface and to interact with the real DOM.
- Some of the methods present under react-dom are createPortal, render, createRoot(react-dom/client).

Components

- Components are building blocks of react application.
- Components are reusable.
- Components are simply a part of UserInterface.
- Dividing our code into parts(components) will improve readability.

Ways of creating a Component.

There are two ways of creating a component in react.

1) Class Based Component(CBC)

2) Function Based Component(FBC)

1)Class Based Component

syntax:=

--App.jsx

```
import React from "react"
class App extends React.Component
{
   render()
   {
      return <h1>jsx</h1>;
   }
}
export default App;
```

> Function Based Component

syntax:-

--App.jsx

```
let App = () =>{
   return <h1>jsx</h1>;
}
export default App;
```

Note:-

- Name of the component file should always begin with the capital letter and save it with the extension .jsx
  example:- App.jsx, Mahesh.jsx
- While rendering the component into another file use it like a paired tag(<App></App>) or unpaired tag(<App />).

Difference Between CBC and FBC.....

Class Based Component

- Complex Syntax
- Dealing with 'this' keyword
- StateFull Components : because of the presence of state and other features like refs, props, context etc.
- LifeCycle methods are present.
- Also known as Smart Components.

Function Based Component

- Simple Syntax
- Absence of 'this' keyword
- Stateless components : because of the absence of state and other features (Can be made statefull using a hook called useState).
- LifeCycle methods are absent(Can be partially achieved through useEffect).
- Also known as Dumb Components.

JSX and Its Rules.....

.jsx

- jsx stands for JavaScript Xtension or JavaScript & XML.
- It allows to write html like structure into JavaScript file.
- It is just an extension for JavaScript Language Syntax.

example:-

```
let JavascriptXtension = ()=>{
   return <h1>I'm JSX</h1>;
}
export default JavascriptXtension;
```

Rules of JSX.....

1) Multiple jsx elements must be wrapped in a common parent element.

```
let JavascriptXtension = ()=>{
return (
<div>
   <h1>Heading</h1>
   <p>Lorem, ipsum dolor.</p>
</div>
);
};
export default JavascriptXtension;
```

To avoid extra node we use the concept of fragment.

fragment:= fragment helps to render multiple jsx elements without creating a extra node.

to make use of fragment there are two ways

i - <></>

ii - <React.Fragment><React.Fragment/>

2) Unpaired tags must be closed properly.

examples:=

<input type="text"></input>

<hr />

3) Attributes like class and for must be replaced with className and htmlFor.

```
import React from "react";
let JavascriptXtension = ()=>{
   return (
```

```
        <React.Fragment>
          <label htmlFor="email" className="labels">Email : </label>
          <input type="email" id="email" />
          <label htmlFor="pass" className="labels">Password : </label>
          <input type="password" id="pass" />
        </React.Fragment>
        )
    }
    export default JavascriptXtension;
```

4) JSX Elements should always be written in LowerCase.

<Button>Click</Button>❌ <button>Click</button>☑

Expressions.....

- Expressions allows to render the values dynamically into jsx elements.
- The values can be Variables, Calculations, or Function calls.
- Expressions are denoted by the symbol '{}'.

```
const JavascriptXtension = () => {
    let obj = {
        name:"Qatar Papa",
        place:"Naveen Heart",
        skills : ["Kissing","Hug","cuddle","Blow Job"]
    }
    let {name,place,skills:[a,b,c,d]} = obj;
    return (
        <>
        <h1>My name is : {name}</h1>
        <h1>My age is : {20+2}</h1>
        <h1>Iam from : {place} </h1>
        <h2>Iam aware of :</h2>
        <ol>
            <li>{a}</li>
            <li>{b}</li>
```

```
        <li>{c}</li>
        <li>{d}</li>
      </ol>
      </>
    )
    }
    export default JavascriptXtension;
```

Props.....

- Props are generally known as Properties.
- Props are mainly used to send the data from parent component to child component.
- Props always returns an Object.
- Props are unidirectional i.e the data should be sent from parent component to child component but not child to parent component.
- Props are inmutalbe i.e the child component should just recieve the data but should not modify the data.

Note:=

- Props appear like html attributes when sending the data whereas appear like function parameters while consuming the data.

======King.jsx=========

```
  import Chaiy from './Chaiy'
  let obj = {
     name:"Pavan",
     age:24,
     hasGF:true,
     hasExGF:undefined,
     hasAttitude:null,
     skills:["Flirting","Sleeping","Batting"],
     fun:()=>{console.log("Core Java Trainee")},
     address:{
        place:"Suryapet",
        pincode:500012
     }
  }
```

```
let King = () => {
   return (<Chaiy details={obj} />)
}
export default King;
```

```
=========Chaiy==========
const Chaiy = (props) => {
let {name,age,hasGF,hasExGF,hasAttitude,skills:[a,b,c],fun,address:{place,pincode}} =
props.details;
   return (
   <>
   <h1>My name is : {name}</h1>
   <h1>My age is : {age}</h1>
   <h1>has GirlFriends : {hasGF?"True":"False"}</h1>
   <h1>has Ex GirlFriends : {hasExGF==undefined?"So Many":"No One"}</h1>
   <h1>has Attitude : {hasAttitude==null?"No":"Yes"}</h1>
   <h1>These Are My Skills : </h1>
   <ol>
      <li>{a}</li>
      <li>{b}</li>
      <li>{c}</li>
   </ol>
   <h1>I am currently staying in {place} and its Pincode is {pincode}</h1>
   <button onClick={fun}>Click It Bitch</button>
   </>
)
}
export default Chaiy;
```


== useContext() ==

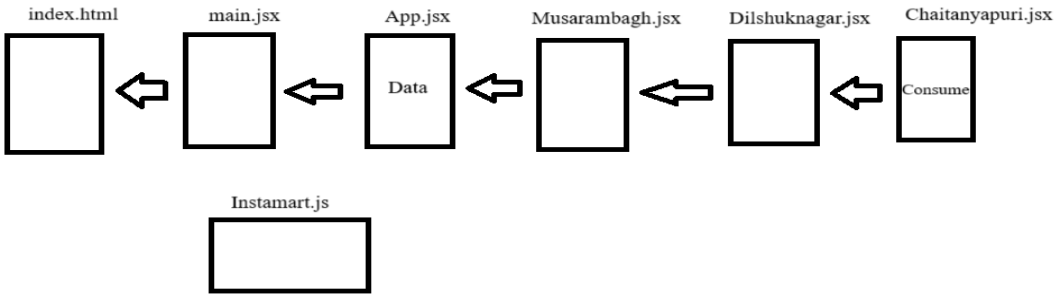
- It is the solution of prop drilling.
- It provides the way to share data with other component without having to pass the prop down to the

Basic Steps to implement Context

1. Create a Context using method “createContext()” from react library
2. Create a provider
3. Implement a “useContext()” hook

Detailed steps to implement context

1. Create a proper folder structure with necessary files and nest them into each other.



2. Into the instamart.js file make use of a method called createContext() from react library and store it in a variable, export it later.

```
import { createContext } from "react";


export let IceCream = createContext()
export let GulabJamun = createContext()
export let Pickle = createContext()
export let Stuff = createContext()
```

3. In app. jsx file, specify the Variable like a Component by attaching provider.. to it, into which all other Component should be rendered

Syntax

```
<variable.provide>
Remaining Components
</variable.provide>
```

Example:

```
import { GulabJamun, IceCream, Pickle, Stuff } from "./useContext/Instamart"
import Musarambagh from "./useContext/Musarambagh"
let App = () => {
  return <IceCream.Provider value="ButterScotch">
    <GulabJamun.Provider value="Haldiram's GulabJamun">
      <Pickle.Provider value="Alekhya Chitti Chicken Pickle">
        <Stuff.Provider value={{stuff1:"Soya Sticks",stuff2:"Boiled
Palli",stuff3:"Chakna",stuff4:"Omlet"}}>
```

```
        <Musarambagh />
      </Stuff.Provider>
    </Pickle.Provider>
  </GulabJamun.Provider>
 </IceCream.Provider>
}
export default App
```

Note:-

- Always pass the value attribute to send the date through the provider.
- import useContext() Hook into the file where the data need to be consumer.
- useContext() taker in Context as its argument (i.e the Name of the variable which stored too create Content method).

Musarambagh.jsx

```
import { useContext } from "react"
import Dilshuknagar from "./Dilshuknagar"
import { Stuff } from "./Instamart"
const Musarambagh = () => {
  let snacks= useContext(Stuff)
  let {stuff1,stuff2,stuff3,stuff4}=snacks
 return <>
 <Dilshuknagar/>
 <h1>My fav snacks items are:</h1>
 <ol>
  <li>{stuff1}</li>
  <li>{stuff2}</li>
  <li>{stuff3}</li>
  <li>{stuff4}</li>
 </ol>
 </>
}
export default Musarambagh
```

Dilshuknagar.jsx

```
import { useContext } from "react"
import Chaitanyapuri from "./Chaitanyapuri"
import { Pickle } from "./Instamart"
```

```
const Dilshuknagar = () => {
   let product3=useContext(Pickle)
  return <>
  <h1>Finally i received {product3}</h1>
  <Chaitanyapuri/>
  </>
}
export default Dilshuknagar
```

Chaitanyapuri.jsx

```
import { useContext } from "react"
import { GulabJamun, IceCream } from "./Instamart"

const Chaitanyapuri = () => {
   let product1 = useContext(IceCream)
   let product2 = useContext(GulabJamun)
   return <div>Chaitanyapuri-{product1}, {product2}</div>
}
export default Chaitanyapuri
```

Adding css to react Component :-

There are three ways of adding the ces to react Components

1) Inline css
2) Global css
3) Module css

1) Inline css :-

- Select the component where you need to apply Inline css.
- select the jsx element and specify style attribute and assign to a expression.
- Inside the expression specify the Stylings in the form of object (key-value pairs)

Ex:- Inline.jsx-

```
Const Inline=()=>{
let btnStyles={
        Color: "white",
        backgroundColor': “royal blue”,
        width: “100%”,
```

```
        border: "none",
        height: "70px"
    }
    return (
    <>
        <h1 style={ color: “white", backgroundColor: "red",textAlign: “center"}}> Inline css
        </h1>
        <button style={btnstyles}> Click <button>
    </>
    )}
export default Inline
```

2) Global css

- Create an external css file at the Global level inside src-folder.
- Import the Global css file where all Other Components Comes under that file (preferably App.jsx).

Example:

App. Jsx

```
import Global from "./AddingStyles/Global"
import Inline from "./AddingStyles/Inline"
import "./Global.css"
const App = () => {
  return (
    <>
    <Inline/>
    <Global/>
    </>
  )
}
export default App
```

Global jsx-

```
const Global = () => {
  return (
    <>
    <img src="path of img" />
```

```
    <p id="para">Lorem10</p>
    </>
  )
}
export default Global
```

Global.css

```
*{
  margin: 0;
  box-sizing: border-box;
}
img{
  height: 400px;
  width: 800px;
  border-radius: 40px;
}
#para{
  color: white;
  background-color: purple;
  padding: 20px;
}
```

3)Module.css

- These files are Saved with the extension "madule.css".
- They provide a way to group the style sheets by Spilling them into different files without having to worry about the repeating classssNames (or) id 's
- To make use of these file, select a Component and import the module css file just like default export.

Example:

Navebar.jsx

```
import "./ModuleStyling.NavbarModule.css"
const  Navbar = () => {
  return (
    <div className={style.container}>Navbar</div>
  )
}
export default  Navbar
```

NavbarModule.css

```
.container{
   height: 10vh;
   background-color: red;
   color: white;
   display: flex;
   justify-content: center;
   align-items: center;
}
```

FormHandling

- Forms are used to collect the data from the user.
- To handle the form data in react we have two different ways.

1)Uncontrolled Form

- It is a way too handle the form data while directly interacting with the JSX elements through DOM or accessing the values using the REF through useRef() hook

UseRef()

- Ref stands for reference.
- It is a hook that helps to get the reference of JSX elements.
- It creates a mutable reference that don't trigger the re-render when change.
- It returns an object that changes overtime (mutable object) with a single property into it known as "CURRENT".

```
Const refContainer = useRef(initialValue)
Console.log(refContainer) //{current:initialValue}
```

Note:

>To link Jsx elements with the Ref-hook we use "useRef()".

2)Controlled Form

- It is a way too handle the form data through state.

Steps to handle the form data through useState()

- Create the structure of the form.
- Import the state hook and assign an object consisting the properties based on upon the number of input fields.
- To link state hook and input fields we use value attribute and assign state object property to it.
- Assign **onChange()** attribute to all the fields and specify **handleChange()** function to it.
- The **handleChange()** takes and optional parameter called as the event object which contains a property named target inside target, there is a property called as value that stores the data entered in the respective input field.
- Assign **e.target.value** to the individual state object property by using a function returned by state hook.

- To fetch the form data on submitting the form, assign **onSubmit()** handler to the form tag and attach handle submit function to it.

Example Form Program:

```
import React, { useState } from 'react'
const ControlledForms = () => {
  let [details, setDetails] = useState({
    name: "",
    email: "",
    password: "",
    phone: "",
    dob: "",
    gender: "",
    skills: [],
    country: "",
    feedback: "",
    photoId: ""
  })
  let { name, email, password, phone, dob, gender, skills, country, feedback } = details
  let handleChange = (e) => {
    let { name, value, type, checked } = e.target

    if (type == "checkbox") {
      if (checked) {
        setDetails({ ...details, [name]: [...skills, value] })
      } else {
        setDetails({
          ...details, [name]: skills.filter((s) =>
            s !== value)
        })
      }
    } else if (type == "file") {
      setDetails({ ...details, [name]: e.target.files[0].name })
    } else {
      setDetails({ ...details, [name]: value })
    }
```

```
}

let handleSubmit = (e) => {
  e.preventDefault()
  console.log(details);

}

return (
  <form onSubmit={handleSubmit}>
    <fieldset>
      <legend>Registration Form</legend>
      <label htmlFor="name">Name:</label>
      <input type="text" id='name' name='name' value={name} onChange={handleChange} />
      <br /><br />
      <label htmlFor="email">Email:</label>
      <input type="email" id='email' name='email' value={email} onChange={handleChange} />
      <br /><br />
      <label htmlFor="password">Password:</label>
            <input type="password" id='password' name='password' value={password}
onChange={handleChange} />
      <br /><br />
      <label htmlFor="phone">Phone No:</label>
      <input type="tel" id='phone' name='phone' value={phone} onChange={handleChange} />
      <br /><br />
      <label htmlFor="dob">Dob:</label>
      <input type="date" id='dob' name='dob' value={dob} onChange={handleChange} />
      <br /><br />

      <div value={gender} onChange={handleChange}>
        <label htmlFor="gender">Gender:</label>
        <input type="radio" name="gender" id="male" value="male" />
        <label htmlFor="male">Male</label>
        <input type="radio" name="gender" id="female" value="female" />
        <label htmlFor="female">Female</label>
        <input type="radio" name="gender" id="others" value="others" />
```

```
          <label htmlFor="others">Others</label>
        </div>
        <br />

        <div value={skills} onChange={handleChange}>
          <label htmlFor="skills">Skills:</label>
          <input type="checkbox" name="skills" id="html" value="Html" />
          <label htmlFor="html">Html</label>
          <input type="checkbox" name="skills" id="css" value="Css" />
          <label htmlFor="css">Css</label>
          <input type="checkbox" name="skills" id="js" value="Js" />
          <label htmlFor="js">Js</label>
          <input type="checkbox" name="skills" id="react" value="React" />
          <label htmlFor="react">React</label>
        </div>
        <br />
        <label htmlFor="country">Country:</label>
        <select name="country" id="country" value={country} onChange={handleChange}>
          <option value="">--Select Country--</option>
          <option value="india">India</option>
          <option value="usa">United States</option>
          <option value="uk">United Kingdom</option>
          <option value="australia">Australia</option>
          <option value="canada">Canada</option>
        </select>
        <br /><br />
        <label htmlFor="feedback">FeedBack:</label>
        <textarea name="feedback" id="feedback" value={feedback} onChange={handleChange}
rows="10" cols="50" ></textarea>
        <br /><br />
        <label htmlFor="photoId">UploadID:</label>
        <input type="file" name='photoId' id="photoId" onChange={handleChange} />
        <br /><br />
        <input type="submit" value="Register" />
      </fieldset>
    </form>
```

```
 )
}
export default ControlledForms.
```

Formik():

- Fromik is a react library that is used to handle the form data eff

Steps to handle the form data through useFormik() hook

- Install the formik library by specifying the command **npm I formik** .
- Create the structure of the form.
- Import the formik hook and assign an object to it. Pass a property called initialValue which stores an object containing the properties corresponding to input field.
- Destructure the object properties from formik.values

```
let { name, email, password, phone, dob, gender, skills, country, feedback,photoId } = formik.values
```

- Link formik.values and input field using value attribute.
- Assign onChange handler for all the input fields and initialize it with **formik.handleChange.**
- Assing the **onSubmit()** attribute and initialize it with formik.handle submit.
- Inside the formik object (after initial values) create a property called onSubmit and assign a function to it which takes an optional parameter that returns an object containing the values entered by the user.

UseEffect():

- It partially replaces life cycle methods in function based components.
- It is mainly used to manage the side Effects.
- Side effects are the operations that occur outside the component render cycle.
- **Example:**

```
Let count=0;
Function increment (){
count++
clg(count)
}
```

- It is used to increase the efficiency of react appliacatiopn by avoiding unnecessary re-renders.

Syntax:

```
useEffect(fn,dependency list)
```

effect fn:

- This functipn will be executed after the component as rendered.
- It is the plece where we write the code for side effects.

Dependency list:

- It is indicated by the symbol **[] .**
- The effect function will only run if one of the dependency(state variable) changes.

UseEffect based on dependencies

Without dependencies;

- If the dependency list is not provided ,the effect function will run after every render of the component i.e acting like 'update-phase'.

Example:

```
useEffect(()=>{
    clg("……")
}
```

UseEffect with empty list as dependency

- If the dependency list is empty .that effect function will run only after the initial render of the component i.e like mounting phase.

Example:

```
useEffect(()=>{
    clg("……")
},[])
```

UseEffect with state as a dependency

- if the dependeny list contain the state variable, effect fn will only run when one of the dependencies changes.

Example:

```
useEffect(()=>{
    clg("……")
},[variable1,v2….])
```

UseEffect with cleanup function

- **If the** effect fn returns another fun(cleaner fun),the effect fn will only run wjen the component is removed from the DOM tree i.e acting like un mounting phase.

Example:

Effect.jsx

```
import { useEffect, useState } from "react"
import Unmount from "./Unmount"

const Effect = () => {
   let [age,setAge]=useState(22)
   let [salary,setSalary]=useState(10000)

   //! without dependency list
   // useEffect(()=>{
```

```
    //     console.log("useEffect Running like updating phase");
    // })
    //! with empty dependency list
    // useEffect(()=>{
    //     console.log("useEffect Running like mounting phase");
    // },[])
    //! with state ad a dependency
    useEffect(()=>{
       console.log("useEffect running");
    },[salary])

    let handleAge=()=>{
       setAge(age+1)
    }

    let handleSalary=()=>{
       setSalary(salary+(salary/2))
    }
  return (
    <>
    <h1>Age - {age}</h1>
    <button onClick={handleAge}>IncreaseAge</button>
    {salary>30000 ? "":<Unmount/>}
    <h1>Salary - {salary}</h1>
    <button onClick={handleSalary}>IncreaseSalary</button>
    </>
  )
}

export default Effect
```

Unmount.jsx

```
import { useEffect } from "react"

const Unmount = () => {
    //! cleanup function
    useEffect(()=>{
       return ()=>{
          console.log("useffect acting like unmounting phase");
       }
    })
  return (
    <h1>Unmount</h1>
  )
}

export default Unmount
```

AXIOS:

- Axios is a library which is use to make http request.
- In simple terms this library use to fetch the data (get()), send the data (host()), update the existing data (pu()), and to delete the data(delete()) from the api.
- Axios is not a part of react ,install it using the command **npm I axios.**

Exapmle:

```
import axios from "axios"
import { useEffect, useState } from "react"


const FetchData = () => {
  let [food, setFood] = useState(null)
  useEffect(() => {
    let extractingData = async () => {
      let { data: { recipes } } = await axios.get("https://dummyjson.com/recipes")
      setFood(recipes)
    }
    extractingData()
  }, [])
  return (
    <table border="1" rules="all" width="100%" cellPadding={20}>
      <thead>
        <tr>
          <th>ID</th>
          <th>NAME</th>
          <th>INGREDIENTS</th>
          <th>INSTRUCTIONS</th>
          <th>CUISINE</th>
          <th>IMAGE</th>
        </tr>
      </thead>
      <tbody>
        {food == null ? <tr><td>Loading...</td></tr> : food.map((item) => {
          return <tr key={item.id}>
            <td>{item.id}</td>
            <td>{item.name}</td>
            <td>{item.ingredients}</td>
            <td>{item.instructions}</td>
```

```
                <td>{item.cuisine}</td>
                <td><img src={item.image} height={200} width={200} /></td>
              </tr>
            })}
          </tbody>
        </table>
      )
    }
    export default FetchData
```

Higher Order Component(HOC):

- These are the components that take another component ass its argument and return a new component.
- This is also a solution for prop-drilling.

Pure Component:

- This is the class based component that render only when ever there is a actual change in the state or props.
- It is mainly used to increase the performance of react application by avoiding un-necessary re-renders.
- in order to create a pure component , the class should be extended from **PureComponent** instead of **Component**

Example:

```
import { Component, PureComponent } from 'react'

export default class Pure extends PureComponent
 {
   constructor() {
     super()
     this.state = {
       name: "Mahender"
     }
   }
   render() {
     console.log("Pure Component Rendering....");
     return (
       <>
         <h1>{this.state.name}</h1>
         <button onClick={() => this.setState({ name: "Ravi Kumar" })}>Pure</button>
```

```
        </>
      )
    }
}
```

Memo:

- memo is higher order component that memorizes(remember) the entire component passed into it.
- It lets you to skip unnecessary re rendering of the component when its props are unchanged.
- It is used to increase the efficiency of our application.

Note: it is similar to the concept of pure component but is specially designed for the function based components.

Example:

App.jsx :-

```
import NormalComponent from './Memo/NormalComponent'
import MemoComponent from './Memo/MemoComponent'
import { useState } from 'react'
const App = () => {
  let [hero, setHero] = useState({
    name: "Asnan"
  })
  return (
    <>
      <NormalComponent name={hero.name} />
      <MemoComponent name={hero.name} />
      <button onClick={() => setHero({ name: "Sharath" })}>Click</button>
    </>
  )
}
export default App
```

NormalComponent.jsx:-

```
const NormalComponent = ({ name }) => {
console.log("Memo Component Rendering....");
return (
<div>NormalComponent-{name}</div>
)
}
export default NormalComponent
```

MemoComponent.jsx:-

```
import { memo } from "react";
const MemoComponent = ({ name }) => {
   console.log("Memo Component Rendering....");
```

```
    return (
      <div>MemoComponent-{name}</div>
    )
}
export default memo(MemoComponent)
```

Portal:

- Portals allows us to render the component outside the dom hierarchy.
  ** <?div>**
- They are mainly used when you need to work with modals (popup's) which act like a individual component.

Steps to implement portals

- Inside the index.html file create a element and specify id attribute it, so that we can render the remining jsx into it.

  ** <?div>**

- **I**n order to render the remaining jsx elements we need to create a portal which can be achieved by using a method create portal from react dom hirachy.
  createPortal(children,container)
  children -> contents to be displayed
  container -> it is the place where contents needs to be stored.

Example:

Index.html:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <link rel="icon" type="image/svg+xml" href="/vite.svg" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Vite + React</title>
 </head>
 <body>
  <div id="root"></div>
  <div id="portal"></div>
  <script type="module" src="/src/main.jsx"></script>
 </body>
</html>
```

Modal.jsx:

```
import { createPortal } from 'react-dom'
```

```
const Modal = () => {
  return createPortal(
    <>
    <h1>Heading...</h1>
    <p>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Sed eveniet incidunt ipsa sint molestiae exercitationem distinctio recusandae hic mollitia modi?</p>
    <button>Click</button>
    </>,
    document.getElementById("portal")
  )
}
export default Modal
```

useMemo:

- It is we to memoize (remember) the output of the heavy function.The Output will only be recalculated when one of its dependencies changed.
- It helps to improve the efficiency of react application by optimizing the performance of heavy function (which makes your application slow).
  Syntax:

  useMemo(fn, dependency List)
- The function (fn) will Contain the competation. that is the heavy function which needs to be memoize .
- Dependency list also known as array of dependencies. will only recalculate the memoized value if one of the dependencies is changed.

Example:

```
import { useMemo, useState } from "react"
const MemoHook = () => {
    let [Counter1, setCounter1] = useState(0)
    let [Counter2, setCounter2] = useState(10)
    let  isEven=useMemo(()=>{
        let i=0
        while(i<1000000000)i++
        return Counter1%2
    },[Counter1])
    return (
        <>
            <h1>Counter1 value is: {Counter1}</h1>
            <button onClick={() => setCounter1(Counter1 + 1)}>Increment Counter1</button>
```

```
            <h2>{isEven ? "even" : "odd"}</h2>
            <h1>Counter2 value is: {Counter2}</h1>
            <button onClick={() => setCounter2(Counter2 + 10)}>Increment Counter2</button>
        </>
    )
}
export default MemoHook
```

UseCallback:

- In react, function recreated every time when the component rerenders even if the function body is same.
- This leads to unnecessary rerendering of the child component that depend upon the function.
- Use callback allows you to memorize the function so that it is created only once and reuse the across the render as long as its dependency remains the same.

  Syntax:

  useCallback(fn,dependency)

- It takes to the function that need to be memorized.
- Dependency list is the array of dependency will only recreate the memorized function when ever one of its dependency changed.

Note:

- ✓ Use callback to work efficiently make use of a method called memo.

Example:

App.jsx

```
import ParentCallback from './UseCallback/ParentCallback'
const App = () => {
  return (
    <ParentCallback />
  )
}
export default App
```

ParentCallback.jsx

```
import React, { useCallback, useState } from 'react'
import Title from './Title'
import Count from './Count'
import Button from './Button'
const ParentCallback = () => {
    let [age, setAge] = useState(23)
```

```
    let [salary, setSalary] = useState(10000)
    let handleAge = useCallback(() => {
      setAge(age + 1)
    }, [age])
    let handleSalary = useCallback(() => {
      setSalary(salary * 1.5)
    }, [salary])
    return (
      <>
        <Title />
        <Count text="Age" value={age} />
        <Button fun={handleAge}>Increment Age</Button>
        <Count text="Salary" value={salary} />
        <Button fun={handleSalary}>Increment Salary</Button>
      </>
    )
  }
  export default ParentCallback
```

Title.jsx

```
  import { memo } from "react"
  const Title = () => {
    console.log("title element rendered")
    return (
      <h1>UseCallback Hook</h1>
    )
  }
  export default memo(Title)
```

Count.jsx

```
  import { memo } from "react";
  const Count = ({ text, value }) => {
    console.log(`${text} element rendered`);
    return (
        <h2>{text}-{value}</h2>
    )
  }
  export default memo(Count)
```

Button.jsx

```
import { memo } from "react";
const Button = ({ children, fun }) => {
   console.log(`${children} element rendered`);
   return (
      <button onClick={fun}>{children}</button>
   )
}
export default memo(Button)
```

Router:

- Router is a library which is mainly used to achieve dynamic navigation(based on the url displaing the content .
- It is used to create interactive single page applications.
- Installation command - **npm I react-router-dom@latest**

Steps to implement router(v5)

a. Import necessary components from the react-router-dom library.
   i. **<BrowserRouter></BrowserRouter> -**helps to connect with url.
   ii. **<Routes></Routes>** -acts as a parent element for the individual route.
   iii. **<Route></Route>-helps to render the**
b. Route component takes in two attributes i.e **path** (which takes in the url path for a component) and **element**(which takes in a component that should be rended based on the specified path).
c. To im[rove the user experience, create a navbar component with the specified links to achieve single page application use **Link** or **NavLink** component instead of anchor tag.
d. To specify the path replace **href** attribute with **to** attribute.

Note:

- the difference b/w link and navlink is that class active will be provide when used with navlink component which indicates the current page being opened.
- To achive nested routing make sure that route component is defined like a paired tag into which specify the individual children routes.
- to render the child component along with the parent component we use **outlet** component inside the parent.

Example:

App.jsx:

```
import { BrowserRouter, NavLink, Route, Routes } from "react-router-dom"
import Home from "./Router/Home"
import NavBar from "./Router/NavBar"
import About from "./Router/About"
import Products from "./Router/Products"
import "./Router.css"
```

```
import PageNotFound from "./Router/PageNotFound"
import Men from "./Router/Men"
import Women from "./Router/Women"
const App = () => {
 return (
  <BrowserRouter>
   <NavBar />
   <Routes>
    <Route path="/" element={<Home />} />
    <Route path="about" element={<About />} />
    <Route path="products" element={<Products />}>
     <Route index path="men" element={<Men />} />
     <Route path="women" element={<Women />} />
    </Route>
    <Route path="*" element={<PageNotFound />} />
   </Routes>
  </BrowserRouter>
 )
}
export default App
```

Navbar.jsx:

```
import { Link, NavLink } from "react-router-dom"
const NavBar = () => {
  return (
    <div id="navbar">
      <NavLink to="/" className="underline">Home</NavLink >
      <NavLink to="/about" className="underline">About</NavLink>
      <NavLink to="/products" className="underline">Products</NavLink>
    </div>
  )
}
export default NavBar
```

Home.jsx:

```
import { useNavigate } from "react-router-dom"
import NavBar from "./NavBar"
const Home = () => {
```

```
 let navigate=useNavigate()
 let handleNavigate=()=>{
  navigate("products")
 }
 return (
  <>
  <NavBar/>
    <h1 id="home">Wellcome to my Store</h1>
    <button onClick={handleNavigate}>Navigate To Product Page</button>
  </>
 )
}
export default Home
```

Steps to implement Router(v6)

1. import the method called browser writer and component called router provider from the react-router-dom library.
2. Creates browser router takes in array of objects into which defines the individual elements as keys.
3. App component returns another component called router provider, which takes in attribute called router into which pass a variable which stores the entire create browser router method.
4. In the case of nested routing pass the key name as children which takes in the array of objects which stores children routes.

Example:

App.jsx:

```
import { createBrowserRouter, RouterProvider } from "react-router-dom"
import Home from "./Router/Home"
import About from "./Router/About"
import Products from "./Router/Products"
import Men from "./Router/Men"
import Women from "./Router/Women"
import PageNotFound from "./Router/PageNotFound"
import "./Router.css"
let routerComponents = createBrowserRouter([
 {
```

```
    path: "/",
    element: <Home/>
   },
   {
    path: "about",
    element: <About />
   },
   {
    path: "products",
    element: <Products />,
    children:[
     {path: "men",
    element: <Men/>},
    {path: "women",
    element: <Women/>}
    ]
   },
   {
    path: "*",
    element: <PageNotFound/>
   }
  ])
  const App = () => {
   return <RouterProvider router={routerComponents}/>
  }
  export default App
```

Use Navigate:

- This hook returns a method that helps to navigate between the routes(pages) in our react application.

  Example:

  Home.jsx

```
import { useNavigate } from "react-router-dom"
import NavBar from "./NavBar"

const Home = () => {
 let navigate=useNavigate()

 let handleNavigate=()=>{
  navigate("products")
```

```
  }
  return (
    <>
    <NavBar/>
      <h1 id="home">Wellcome to my Store</h1>
      <button onClick={handleNavigate}>Navigate To Product Page</button>
    </>
  )
}
export default Home
```

About.jsx:

```
import { useNavigate } from "react-router-dom"
import NavBar from "./NavBar"

const About = () => {
  let navigate=useNavigate()

  let handleNavigate=()=>{
    navigate(-1)
  }
  return (
    <>
     <NavBar/>
    <h3 id="about">Lorem ipsum dolor, sit amet consectetur adipisicing elit.Molestiae!</h3>
    <button onClick={handleNavigate}>Navigate To Previous Page</button>
    </>
  )
}

export default About
```