REACT JS

by

PRAJWAL SIR

Codes of each Topics, Assignments & Projects

Click Here

- **Library –** Collection of pre-defined codes. ***(React-JS)***
- **Framework –** Collection of Libraries. ***(Angular-JS, Vue-JS, Next-JS)***
- ***Facebook*** created ***React JS*** in ***2013***.
- ***Google*** created ***Angular JS*** in ***2007***.
- **Frameworks -** It is a collection of libraries.
- **React-JS -** It is a library of JavaScript.
- **Why React JS? (Features of React-JS)**
  - There are 3 main reasons. i.e.
    1. To create a ***Single Page Application*** (Gmail, YouTube, Facebook).
    2. It is ***Declarative***.
    3. It follows ***Component-Based Architecture***.
- **Imperative** – We have to write logic for everything.
- **Declarative** – Instead of writing logic, some developers have already written the logic and we are using the logic.

Abbreviations	Full Form	Extension
HTML	Hypertext Markup Language	.html
CSS	Cascading Style Sheet	.css
JS	JavaScript	.js
JSX	JavaScript and XML	.jsx ***(It is a combination of JavaScript and Extensible Markup Language.)***

- **<u>Rules to follow while using XML</u>:**
  - In the place of "class", we will be using "className"
  - lowerCamel Case is mandatory. Example:- ***onClick***, ***onMouseOver*** etc.
  - XML will return a single container, all the tags should be wrapped inside one particular tag.

- **Difference between HTML and XML**

XML	HTML
1. In html we will be using attribute as "className" **Ex:** <h1 className = 'a'> Heading </h1>	**1.** In html we will be using attribute as "class" **Ex:** <h1 class= 'a'>Heading</h1>
2. All the events in XML we use in lowerCamel Case **Ex:** onClick, onMouseOver, onSubmit etc.	**2.** All the events in HTML we use in lowercase **Ex:** onclick, onmouseover, onsubmit etc.
3. In Xml we have to warp up all the content into a single container. **Ex:** Heading para 	**3.** In Html we can write two different tags. **Ex:** Heading para

❒ **Download and Install React JS?**

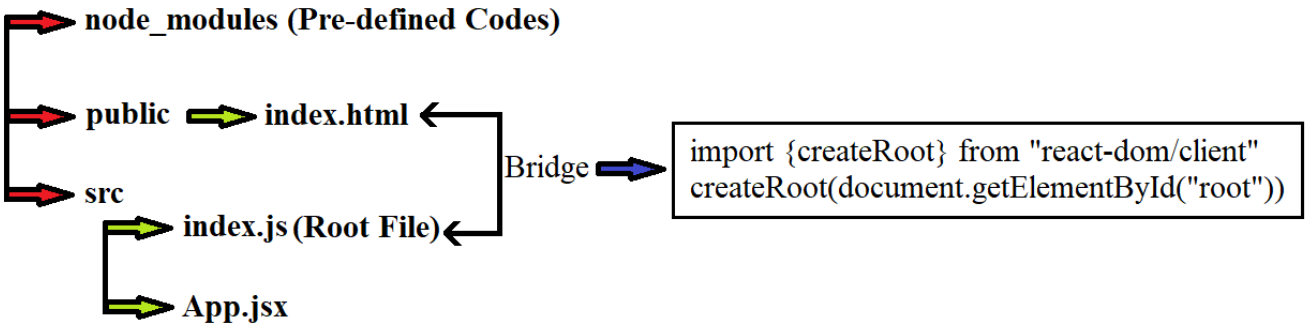
1. Install node JS(after installation close everything)
2. Create a folder (React)
3. Open cmd(Command Prompt)
4. We need to pass some commands
   - ➢ "npx create-react-app project" (project name should not start with num, capital-letter and R). Or,
   - ➢ "npm install -g create-react-app"
     Create-react-app project
5. Ok to proceed? Y/N (Yes)
6. After installation it will say Happy hacking!
7. Type cd project (Project name)
8. Type npm start. ***(You will getting the default UI(user Interface) of react.)***

❒ **Npm** → Node package manager.
❒ **Npx** → Node Package Executor.

❒ **<u>Folder structure-</u>**

- ❖ **node-modulus**
  - ➢ all the pre-defined code are present in this folder (***do not touch***)
- ❖ **public**
  - ➢ This folder contains the main structure of webpage.
  - ➢ One important file we have to maintain "index.html"
- ❖ **src**
  - ➢ It is a source folder where we are going to write the code.
  - ➢ Inside src folder you have to maintain two important files i.e.
    - **I. index.js(root file) -** It is a file where we used two create a root between index.js(src) and index.html(public)

```
import {createRoot} from "react-dom/client"
import App from "./App"

createRoot (document.getElementById("root")).render(<App/>)
```



II.App.jsx(Component) - it is a component where we will be writing our own code.
Whenever you are creating a component first letter should be CAPITAL LETTER.

```
const App=()=>{
   return(
      <div>
        <h1>Hello World</h1>
       <h2>Hello Galaxy</h2>
      </div>
   )
}
export default App
```

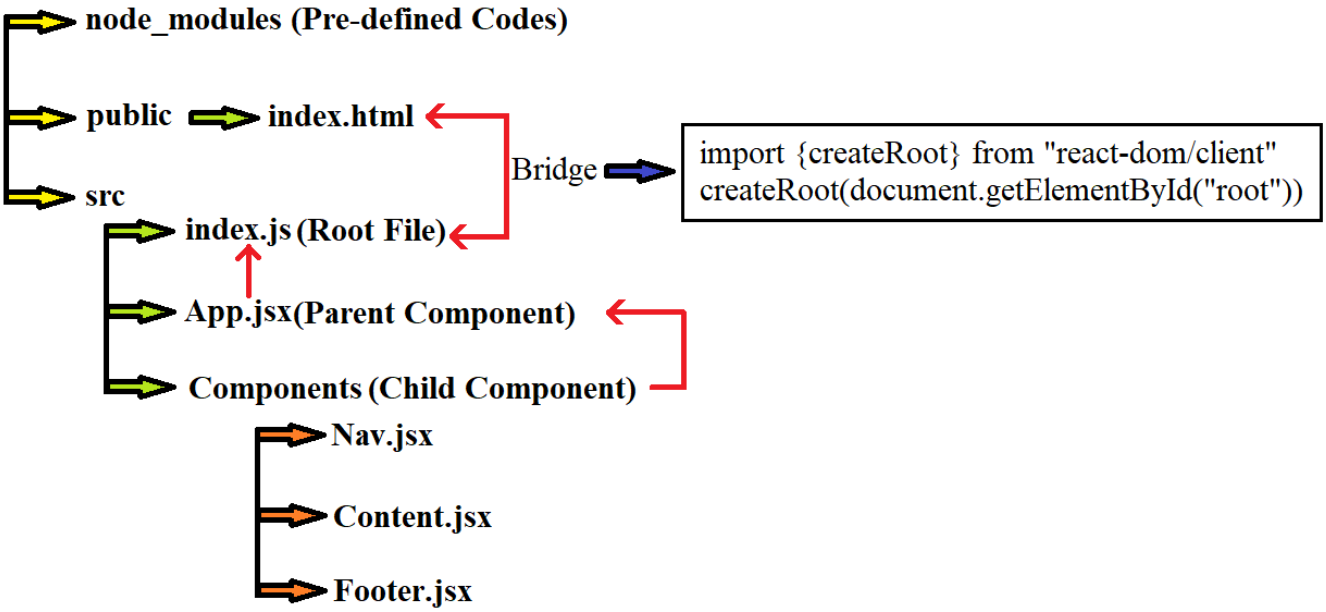
- **package-lock.json & package.json -** These are two files where it is considered as directories of the react folder.
  It will give you all the information about libraries present in the project.

❒ **Rules while creating Component:**

- First letter of the file name should be always capital.
- File extension of component is ".jsx".(ex: App.jsx, Nav.jsx, Content.jsx)

❒ **Components:**

- Components are nothing but building blocks of the web page.
- Web will be divided into multiple Components(files) and we will be joining together in the parent Component(App.jsx). **Ex:** Navbar.jsx, Content.jsx, Footer,jsx
- Component are reusable, as many times you want can declare the components
- We can declare components in two ways:
  - Self-closing tag(<ComponentName/>)
  - Paired tag(<ComponentName></ComponentName>)



❒ Code for index.js-

```
import {createRoot} from "react-dom/client"
import App from "./App"
createRoot(document.getElementById("root")).render(<App/>)
```

❒ Code for App.jsx-

```
import Nav from "./Components/Nav"
import Content from "./Components/Content"
import Footer from "./Components/Footer"
const App = ()=>{
   return(
      <div>
     <h1>Hello World</h1>
     <h1>Hello Galaxy</h1>
     <Nav/>
     <Content/>
     <Footer/>
     </div>
   )
}
export default App
```

❒ Code for Nav.jsx-

```
const Nav = ()=>{
   return(
      <section>
      <h1>Navbar</h1>
      </section>
   )
}

export default Nav
```

❒ Code for Content.jsx-

```
const Content = ()=>{
   return(
      <section>
         <h1>Content</h1>
      </section>
   )
}

export default Content
```

❒ Code for Footer.jsx-

```
const Footer = ()=>{
   return(
      <section>
         <h1>Footer</h1>
      </section>
   )
}

export default Footer
```

❒ Types of Components:

FBC(Function Base Component)	CBC(Class Base Component)
1. Use JS Function	**1.** Use Class
2. State Less	**2.** State Full
3. Use Hooks	**3.** Can't use Hooks
4. Can't use Life-Cycle method	**4.** Use Life-Cycle methods
5. render() is not used in FBC	**5.** render() is used in CBC

6. Ex:
const Fbc=()=>{
return(
 	
Hello FBC
 	
)
}
export default Fbc
	6. Ex:
	Import {Component} from "react"
	Class Cbc extends Component{
	render(){
	return(
	
	Hello CBC
	
	)
	}
	}
	export default Cbc

❒ Props:

- Stands for 'Properties'.
- It is an In-build object.
- It is used to transfer the data from parent component.
- Props are immutable, it means once the value is passed from parent component it can't be changed.

- Props are Uni-directional.

❒ Code for App.jsx-

```
import React from 'react'
import Child from './Components/Child'
const App= ()=> {
 return (
   <div>
    {/* props */}
    <Child data= "World"/>
    <Child data= "Galaxy"/>
    <Child data= "Universe"/>
   </div>
 )
}
 export default App
```

❒ Code for Child.jsx-

```
import React from "react";
const Child = (x) => {
   console.log(x)
 return (
   <div>
      <h1>Hello {x.data} </h1>
   </div>
 )
}
export default Child
```

❒ Props passing array-

➢ Code for App.jsx-

```
import React from 'react'
import Child from './Components/Child'
const App= ()=> {
 return (
   <div>
    {/* props passing Array */}
    <Child data={["Hello", "Bye"]}/>
   </div>
 )
}
 export default App
```

➢ Code for Child.jsx-

```
import React from "react";

const Child = (x) => {
   console.log(x)
 return (
   <div>
      <h1>Hello {x.data[0]}</h1>
```

```
      <h1>Hello {x.data[1]}</h1>
    </div>
  )
}
export default Child
```

❒ Props passing an object-

➢ Code for App.jsx-

```
import React from 'react'
import Child from './Components/Child'
const App= ()=> {
let Person = {
       id:123,
       name: “Debajyoti"
    }


  return (
    <div>
     {/* Props Passing An object  */}
     <Child data={Person}/>
    </div>
  )
}
 export default App
```

➢ Code for Child.jsx-

```
import React from "react";
const Child = (x) => {
   console.log(x)
  return (
    <div>
       <h1>Hello {x.data.name}</h1>
    </div>
  )
}
export default Child
```

❒ Props passing an array of an object-

➢ Code for App.jsx-

```
import React from 'react'
import Child from './Components/Child'
   let webTech = [
      {name: "HTML", id:123},
      {name: "CSS", id:456},
      {name: "JS", id:789}
   ]

  return (
    <div>
     {/* Props passing an array of an object */}
     <Child data={webTech}/>
```

```
        </div>
  )
}
 export default App
```

➢ **Code for Child.jsx-**

```
import React from "react";

const Child = (x) => {
   console.log(x)     // Object: {data:[ ]}
   console.log(x.data)     // Array {[ ], [ ], [ ]}
   console.log(x.data[0])      // Object: {name:" ", id: number}
   console.log(x.data[0].name)     // HTML
 return (
   <div>
      <h1>Hello {x.data[0].name} you have id {x.data[0].id}</h1>
      <h1>Hello {x.data[1].name} you have id {x.data[1].id}</h1>
      <h1>Hello {x.data[2].name} you have id {x.data[2].id}</h1>
   </div>
 )
}
export default Child
```

❒ **Props-Drilling-**

➢ It is a process by which you can pass the data from one part of the react to another by going through other parts that do not need the data but only help in passing it around.

Example-

➢ **Code for App.jsx-**

```
import React from 'react'
import A from './Components/A'

 return (
   <div>
    {/* Props Drilling */}
    <A data1="Hi"/>
      </div>
 )
}
 export default App
```

➢ **Code for A.jsx-**

```
import B from "./B";

const A=(props1)=> {
   console.log(props1)
 return (
   <div>
     <h1><B data2={props1.data1}/></h1>
   </div>
 )
}
```

```
export default A
```

- **Code for B.jsx-**

```
const B=(props2)=> {
   console.log(props2)
  return (
    <div>
      <h1>{props2.data2} Good Morning</h1>
    </div>
  )
}
export default B
```

❒ Props use to access child content-

- **Code for App.jsx-**

```
import React from 'react'
import PropsChildren from './Components/PropsChildren'

const App= ()=> {
  return (
    <div>
      {/* props accessing child contents */}
      <PropsChildren data="Hi">
        Good Morning
      </PropsChildren>
    </div>
  )
}
 export default App
```

- **Code for PropsChildren.jsx-**

```
const PropsChildren = (props)=> {
   console.log(props)
  return (
    <div>
       <h1>{props.data} {props.children}</h1>
    </div>
  )
}
export default PropsChildren
```

❒ **States-**

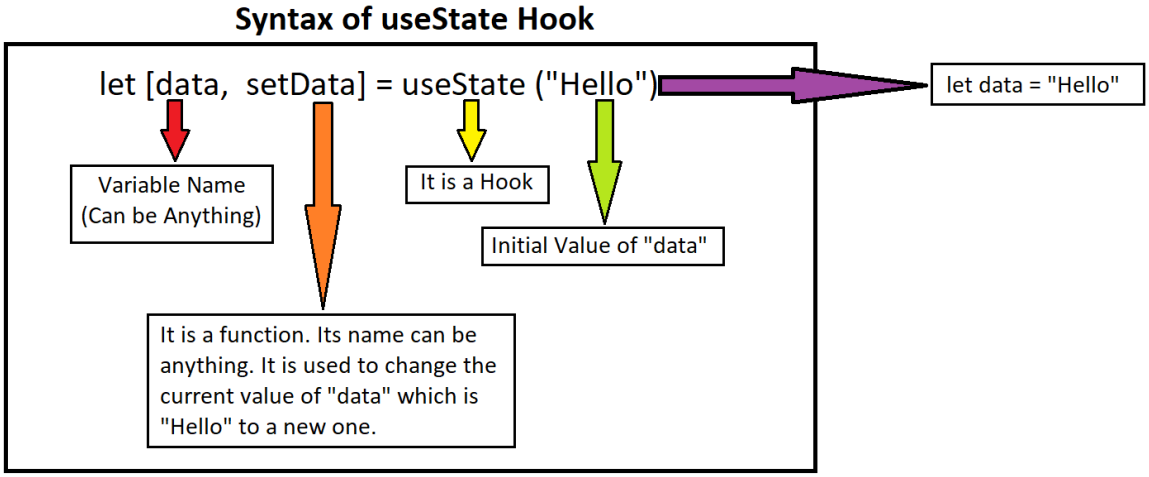
- States are used to create dynamic data on the UI.
- States are mutable, states value can be changed.
- States are local, states belong to one particular component. We cannot share the states between the components like props.
- By default Function based Components are Stateless, it means we don't have any inbuilt state object in Function Based Component, But we can make Function based component stateful by using an advance feature called 'Hooks' and the hook which is used to make Function Based Component stateful is 'useState'.

❒ **Hooks-**

- Hooks are like inbuilt methods in React.
- Hooks are always starts from a prefix word "use".
- Whenever we wanted use hooks we have to import it from React Library, import statement is mandatory.
- Hooks are only used in Function based component.
- We have many hooks in react these are the few below hooks –
  - ❖ useState.
  - ❖ useEffect.
  - ❖ useContext.

Basic Hooks	Advanced Hooks
useState	useRef
useEffect	useCallback
useContext	useMemo
---	useNavigate
	useParams

❒ **useState** – This hook is used to create a state in function based component.
❒ **Lists** – In React List is used to display data in an ordered format.
❒ **Map** – The map() function is used for traversing the list.



➔ Example of States-

❒ Code for App.jsx-

```
import States from "./Components/States"
const App = ()=>{
   return(
      <div>
         <States/>
      </div>
   )
}
export default App
```

❒ Code for States.jsx-

```
import { useState } from "react"
const States = ()=>{
   let [data, setdata] = useState ("Hungry")  // let data = "Hungry"
   let biriyani = () =>{setdata("Biriyani was tasty")}
   return(
      <div>
         <h1>{data}</h1>
         <button onClick={biriyani}>Food</button>
      </div>
   )
}
export default States
```

❒ Creating Increment, Decrement & Reset button using states-

➢ Code for App.jsx-

```
import Task1 from "./Components/Task1"
const App = ()=>{
   return(
      <div>
         <Task1/>
      </div>
   )
}
export default App
```

➢ Code for Task1-

```
// Create increment, decrement, Reset button by using States //

import { useState } from "react";
const Task1 = ()=>{
   let [count, setCount] = useState(0)  // let count = 0

      // Increment Function
   let incre = ()=>{
      setCount(count+1)
      console.log(count+1);
   }
```

```
    // Decrement Function
    let decre = ()=>{
        setCount(count-1)
    }

    // Reset Function
    let reset = ()=>{
        setCount(0)
    }
    return(
        <div>
            <h1>{count}</h1>
            <button onClick={incre}>Increment</button>
            <button onClick={decre}>Decrement</button>
            <button onClick={reset}>Reset</button>
        </div>
    )
}
export default Task1
```

❒ States returning array-

➢ Code for App.jsx-

```
import Statearr from "./Components/Statearr"
const App = ()=>{
    return(
        <div>
            <Statearr/>
        </div>
    )
}
export default App
```

➢ Code for Statearr.jsx-

```
import { useState } from "react";
const Statearr = ()=>{
    let [arr] = useState(["Debajyoti" , "Keshab" , "Asfar" , "Chiranjit"])
    // let arr = ["Debajyoti" , "Keshab" , "Asfar" , "Chiranjit"] //
    return(
        <div>
            <h1>{arr[0]}</h1>
            <h1>{arr[1]}</h1>
            <h1>{arr[2]}</h1>
            <h1>{arr[3]}</h1>
        </div>

    )
}
export default Statearr
```

❒ States returning object-

➢ Code for Stateobj.jsx-

```
// States returning Object //
import { useState } from "react";
const Stateobj = ()=>{
   let [obj] = useState({name: "Debajyoti" , id: 123, place: "West Bengal"})
   // let obj = {name: "Debajyoti" , id: 123, place: "West Bengal"} //
   return(
      <div>
         <h1>{obj.name}</h1>
         <h1>{obj.id}</h1>
         <h1>{obj.place}</h1>
      </div>
   )
}
export default Stateobj
```

➢ Code for App.jsx-

```
import Stateobj from "./Components/Statesobj"
const App = ()=>{
   return(
      <div>
         <Stateobj/>
      </div>
   )
}
export default App
```

❒ Fetch data using State-

- ❖ Create a file, name *"userData.json"* inside Component folder.
- ❖ Open the following link in browser –
  https://api.github.com/users
- ❖ Copy the whole content and paste inside *"userData.json"*.
- ❖ Next, import it in another child component, name *"Fetchdata.json"*

➢ Code for Fetchdata.json-

```
import { useState } from "react"
import content from "./userData.json"
const Fetchdata = ()=>{
   let [data, setData] = useState(content)
   console.log(data)
   return(
      <div>
         {
            data.map((x)=>{
               return(
                  <div>
                     <h1>{x.id}</h1>
                     <h1>{x.login}</h1>
```

```
                    <img src={x.avatar_url} alt="" />
                </div>
            )
        }
    )
    }
</div>
)
}
export default Fetchdata
```

➢ Code for App.jsx-

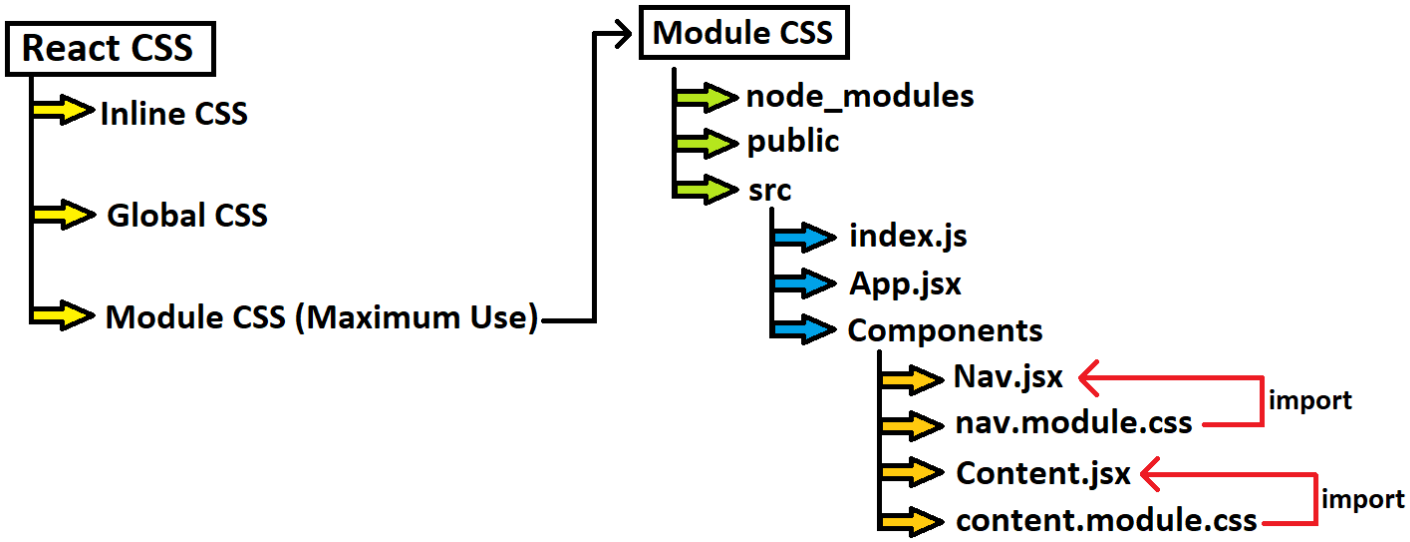
```
import Fetchdata from "./Components/Fetchdata"
const App = ()=>{
    return(
        <div>
            <Fetchdata/>
        </div>
    )
}
export default App
```

➢ Code for Fetchdata.json using 'Fragment' tag instead of 'div' tag in return statement-

```
import { Fragment, useState } from "react"
import content from "./userData.json"
const Fetchdata = ()=>{
    let [data] = useState(content)
    console.log(data)
    return(
        <div>
            {
                data.map((x)=>{
                    return(
                        <Fragment key={x.id}>
                    {/* Fragment creates a container but not create any useless object */}
                            <h1>{x.id}</h1>
                            <h1>{x.login}</h1>
                            <img src={x.avatar_url}/>
                        </Fragment>
                    )
                }
            )
            }
        </div>
    )
}
export default Fetchdata
```

❒ **Myntra Navbar:-**

- Project File Link - https://tinyurl.com/ex6xjxa8
- After download this file (.zip file), open it in VS Code.
- Open New Terminal.
- Type ***"npm i"*** to install ***node_modules package.***
- After that type ***"npm start"***



❒ **Types of CSS in React JS-**

- Inline CSS
- Global CSS
- Module CSS

❒ **Inline CSS-**

- It is a type of CSS which will apply individually inside one particular tags using "style" attribute.
- The CSS properties should be written inside an expression in the form of "Object".

Ex-

```
const Hello = ()=>{
   return(
      <div>
         <h1 style={{backgroundColor:"red" , color:"gold"}}>Hello1</h1>   {/* inline css */}
      </div>
   )
}
export default Hello
```

❒ Global CSS-

- It is a type of CSS which we will use to maintain one CSS file for entire react project.
- It will target all the components.
- We have to create a separate file inside "src" with an extension of ".css" and write all the styles. **Ex-**
- **Code for First Component-**

```
const Hello1 = ()=>{
   return(
      <div>
         <h1>Hello1</h1>
      </div>
   )
}
export default Hello1
```

- **Code for Second Component-**

```
const Hello2 = ()=>{
   return(
      <div>
         <h1>Hello2</h1>
      </div>
   )
}
export default Hello2
```

- **Code for global CSS-**

```
h1{
   background-color: aqua;
}

h2{
   background-color: blueviolet;
}
```

❒ Module CSS-

- In React this CSS is the most used type of CSS.
- We will be creating a separate CSS file for each component.
- The respective styles required for the particular component will be written their respective CSS files.
- Whenever we are using module CSS we have to create CSS file with an extension ".module.css". **Ex-** "filename.module.css"

- **Code for Component-**

```
import style from "./style.module.css"
const Spotify = ()=>{
   return(
      <div id={style.nav}>
         <h1>Hello Spotify</h1>
      </div>
```

```
    )
}
export default Spotify
```

- **Code for style.module.css-**

```
#nav>h1{
    background-color: blue;
}
```

❒ Ref (References)-

- It is an inbuilt object in React JS.
- It is used to target element in React JS.
- By default it will be having key-value pairs of “current:undefined”.
- Ref will always uses Real DOM.
- In Function Based Component we use “useRef” hook to create References.

❒ Example of References-

```
import { useRef } from "react"
const Reference = ()=>{
    let demoref = useRef()
    console.log(demoref) // {current:undefined}

    let color = ()=>{
        demoref.current.style.background="red"
    }

    return(
        <div>
            <h1 ref={demoref}>REFERENCE</h1>
            <h2>Hello 2</h2>
            <p>para</p>
            <button onClick={color}>Change Color</button>
        </div>
    )
}
export default Reference
```

❒ Write a code to change light theme to dark theme and vice-versa-

```
import React from 'react'

const Theme=()=> {

// Function for Dark Theme //
let Dark=()=>{
    document.body.style.backgroundColor = "black"
    document.body.style.color = "White"
}

// Function for Light Theme //
let Light=()=>{
    document.body.style.backgroundColor = "gainsboro"
    document.body.style.color = "Black"
}
```

```
 return (
   <div>
     <input type="radio" name='a' onClick={Dark}/>
     <label htmlFor="">Dark</label>

     <input type="radio" name='a' onClick={Light}/>
     <label htmlFor="">Light</label>

     <h1>Hello World</h1>
   </div>
 )
}
export default Theme
```

❒ Write a code to create a form and print the data in console-

```
import React from 'react'
import { useRef } from 'react'

const Formhandle=()=> {
let name = useRef()
let salary = useRef()
let company = useRef()

let formhandle=(e)=>{
   e.preventDefault()    /* It is used to prevent the default settings of formhandle */
   let a = name.current.value
   let b = salary.current.value
   let c = company.current.value
   console.log({name:a, salary:b, company:c })
}
 return (
   <div>
     <form>
       <label htmlFor="">Name</label>
       <input type="text" ref={name}/>
       <br/><br/>

       <label htmlFor="">Company</label>
       <input type="text" ref={salary}/>
       <br/><br/>

       <label htmlFor="">Salary</label>
       <input type="number" ref={company}/>
       <br/><br/>

       <button onClick={formhandle}>Submit</button>
     </form>
   </div>
 )
}
export default Formhandle
```

Output of the above code –

Name Debajyoti Ghosh

Company Alphabet

Salary 7000000

Submit

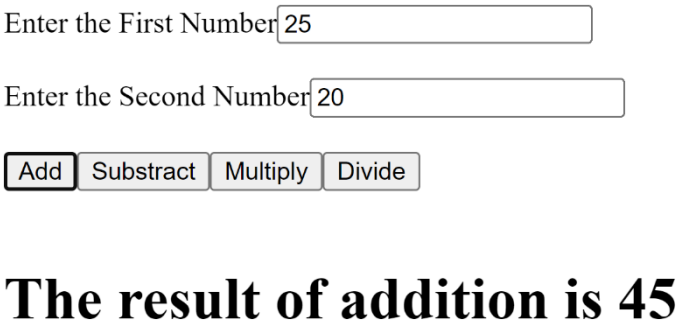
- **Form handling in React-**
  - Uncontrolled Forms
  - Controlled Forms
- **Uncontrolled Forms –**
  - It is a form where it is created using reference concept.
  - In Function Based Component we use *"useRef"* to create uncontrolled forms.
  - This forms are completely handled by DOM itself.
  - Suppose if we wanted to take any data from user first the data will be taken by DOM and then we will be taking the data from DOM.
- **Controlled Forms –**
  - These forms are created using States in React.
  - In Function Based Component we use *"useState"* to create controlled forms.
  - These forms are completely handled by developers where we will be taking the data directly from the users by using *"onChange"* event.

- **Write a code to create a calculator using uncontrolled form-**

```
// Using uncontrolled form

import React from 'react'
import { useRef } from 'react'
import { useState } from 'react';

const Calculator=()=> {
let num1=useRef();
let num2=useRef();

let [res, setResult] = useState("")
console.log(num1)
console.log(num2)
```

```
let add = (e)=>{
    e.preventDefault();
    let a = num1.current.value
    let b = num2.current.value
setResult(parseInt(a) + parseInt(b))
}

let substract = (e)=>{
    e.preventDefault();
    let a = num1.current.value
    let b = num2.current.value
    setResult(a - b)
}

let multiply = (e)=>{
     e.preventDefault();
     let a = num1.current.value
     let b = num2.current.value
     setResult(a * b)
}

let divide = (e)=>{
    e.preventDefault();
    let a = num1.current.value
    let b = num2.current.value
    setResult(a / b)
}

  return (
    <div>
      <form action="">
        <label htmlFor="">Enter the First Number</label>
        <input type="number" ref={num1}/>
        <br /> <br />

        <label htmlFor="">Enter the Second Number</label>
        <input type="number" ref={num2}/>
        <br /><br />


        <button onClick={add}>Add</button>
        <button onClick={substract}>Substract</button>
        <button onClick={multiply}>Multiply</button>
        <button onClick={divide}>Divide</button>
        <br/><br/>
        <h1>{res}</h1>
      </form>
    </div>
  )
}
export default Calculator
```

Output of above code-

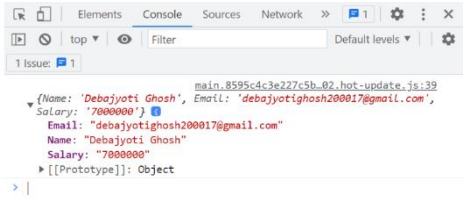


❒ **Write a code to create a form and print the output both UI and Console-**

```
import { useState } from "react"

const Controlled = ()=>{
   let[name,setName]=useState("")
   let[email,setEmail]=useState("")
   let[salary,setSalary]=useState("")

   let nameData=(e)=>{
      setName(e.target.value)
   }

   let emailData=(e)=>{
      setEmail(e.target.value)
   }

   let salaryData=(e)=>{
      setSalary(e.target.value)
   }

   let formhandle = ()=>{
      console.log({'Name':name, 'Email':email, 'Salary':salary})
   }

   return(
      <div>
      <label>Name</label>
      <input type="text" value={name} onChange={nameData}/>
      <br/><br/>

      <label>Email</label>
      <input type="text" value={email} onChange={emailData} />
      <br/><br/>
```

```
      <label>Salary</label>
      <input type="number" value={salary} onChange={salaryData}/>
      <br/><br/>

      <button onClick={formhandle}>Submit</button>

      <h2>{name}</h2>
      <h2>{email}</h2>
      <h2>{salary}</h2>
      </div>
   )
}
export default Controlled
```

Output of above code-



❒ Write a code to create a calculator using controlled form-

```
// Using controlled form
import { useState } from "react"

const Calculator2 = ()=>{
   let[num1,setNum1]=useState("")
   let [num2,setnum2]=useState("")
   let [result, setResult]=useState("")

   let num1Data =(e)=>{
      setNum1(e.target.value)
   }

   let num2Data =(e)=>{
      setnum2(e.target.value)
   }


   let add = (e)=>{
      e.preventDefault();
      let a = num1
      let b = num2
   setResult(parseInt(a) + parseInt(b))
```

```
    }

    let substract = (e)=>{
        e.preventDefault();
        let a = num1
        let b = num2
        setResult(a - b)
    }

    let multiply = (e)=>{
         e.preventDefault();
         let a = num1
         let b = num2
         setResult(a * b)
    }

    let divide = (e)=>{
        e.preventDefault();
        let a = num1
        let b = num2
        setResult(a / b)
    }

    return(
        <div>
        <form action="">
        <label htmlFor="">Enter the First Number:- </label>

        <input type="number" value={num1} onChange={num1Data}/>
        <br /> <br />

        <label htmlFor="">Enter the Second Number:- </label>
        <input type="number" value={num2} onChange={num2Data}/>
        <br /><br />

        <button onClick={add}>Add</button>
        <button onClick={substract}>Substract</button>
        <button onClick={multiply}>Multiply</button>
        <button onClick={divide}>Divide</button>
        <br/><br/>
        {/* <input type="text" value={result} readOnly /> */}
        <h1>Result is:- {result}</h1>
        </form>
        </div>
    )
}
export default Calculator2
```

Output of above code-

Enter the First Number:- 25

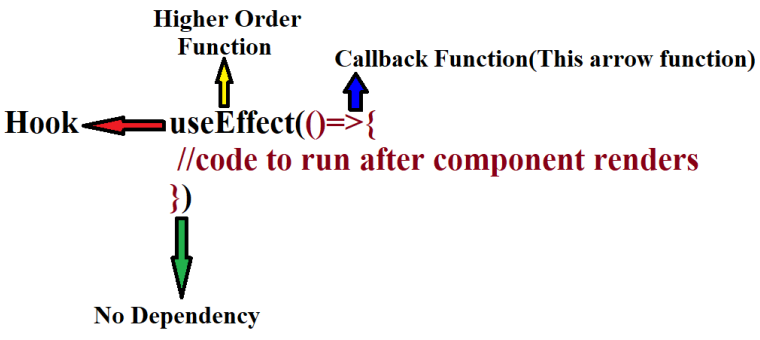
Enter the Second Number:- 20

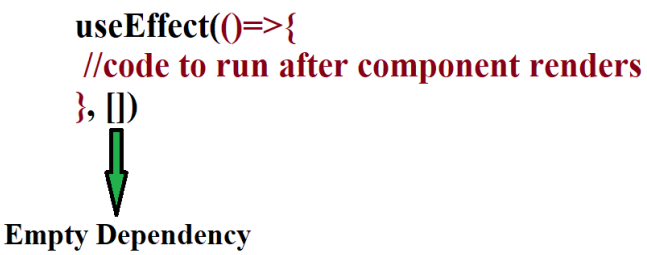
Add | Substract | Multiply | Divide

Result is:- 45

❒ useEffect –

```
import React, { useEffect, useState } from 'react'

const SideEffect = () => {
   let [count1, setCount1] = useState(0)
   let [count2, setCount2] = useState(10)

   let incre1=()=>{setCount1(count1+1)}
   let incre2=()=>{setCount2(count2+1)}

   useEffect(()=>{
      console.log("Hello")
   },[count1])

 return (
   <div>
     <h1>{count1}</h1>
     <button onClick={incre1}>Increment1</button>

     <h1>{count2}</h1>
     <button onClick={incre2}>Increment2</button>
   </div>
 )
}

export default SideEffect
```





❒ **How to fetch data from a given link of Database –**

➢ **Link of dummy data -** *https://jsonplaceholder.typicode.com/posts*

❒ **Code to fetch the data-**

```
import React, { useEffect, useState } from 'react'
import axios from 'axios'

const FetchData = () => {
let [content, setContent] = useState([])

   useEffect(()=>{
      axios.get("https://jsonplaceholder.typicode.com/posts")
      .then((response)=>{
        setContent(response.data)
        console.log(response.data)
          console.log("Got the Data");
      }).catch(()=>{
          console.log("Something is wrong");
      })
   },[])
  return (
    <div>
     {content.map((x)=>{
      return(
        <div>
          <h1>{x.id}</h1>
          <h1>{x.title}</h1>
        </div>
      )
     })}
    </div>
  )
}

export default FetchData
```

❒ **How to fetch data of user input from a given link of Database –**

➢ **Link of dummy data -** *https://jsonplaceholder.typicode.com/posts*

❒ **Code to fetch the data-**

```
import React, { useEffect, useState } from 'react'
import axios from 'axios'

const FetchData2 = () => {
let [content, setContent] = useState([])
let [id, setId] = useState("")
let [btn, setBtn] = useState("")

   useEffect(()=>{
      axios.get(`https://jsonplaceholder.typicode.com/posts/${btn}`)
      .then((response)=>{
        setContent(response.data)
        console.log("Got the Data");
        console.log(response.data)
```

```
      }).catch(()=>{
         console.log("Something is wrong");
      })
   },[btn])

let iddata = (e)=>{
setId(e.target.value)

}

let formHandle=()=>{
setBtn(id)
}

  return (
    <div>
      <input type="text" value={id} onChange={iddata}/>
      <button onClick={formHandle}>FetchData</button>
      <h1>{content.title}</h1>
    </div>
  )
}

export default FetchData2
```

❒ **Q. Fetch the data from the following link –**

Link - *https://api.github.com/users*

```
import axios from 'axios'
import React, { useState } from 'react'
import { useEffect } from 'react'

const FetchData3 = () => {
   let [content, setContent] = useState([])

   useEffect(() => {
      axios.get("https://api.github.com/users")
      .then((response)=>{
         setContent(response.data)
         console.log("Got the data")
         console.log(response.data)
      }).catch(()=>{
         console.log("Error is happening")
      })
   },[])
  return (
    <div>
     {content.map((x)=>{
       return(
          <div>
            <h1>{x.id}</h1>
            <img src={x.avatar_url} alt=""/>
          </div>
       )
```

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

export default FetchData3
```

❒ useContext-

- ***Context*** is a way to share data between components without having to pass props down the component tree.
- ***useContext*** is a hook that allows you to consume context values in your functional components.
- To create a context, use the ***createContext()*** method. This returns a Context object that can be used to provide and consume values.
- To provide a value to a child component, use the ***Context.Provider*** component. This component accepts a ***value*** prop that is passed down to child components.
- To consume a ***value*** in a child component, use the ***useContext*** hook. This hook takes a Context object as its argument and returns the current value of the context.
- We can use multiple context values in a single component by ***nesting Context.Provider components***.
- Overuse of context can make our code harder to understand and maintain.

❒ Example of UseContext Hook-

➢ Code for parent Component-

```
import React, { createContext } from 'react'
import B from './B'

export let userData = createContext()
const A = () => {
  return (
    <div>
      <userData.Provider value="Hello">
        <B />
      </userData.Provider>
    </div>
  )
}
export default A
```

➢ Code for child component-

```
import React from 'react'
import C from './C'

const B = (props) => {
   console.log(props)
  return (
    <div>
      <C />
```

```
    </div>
  )
}
export default B
```

- **Code for grandchild / nested child component –**

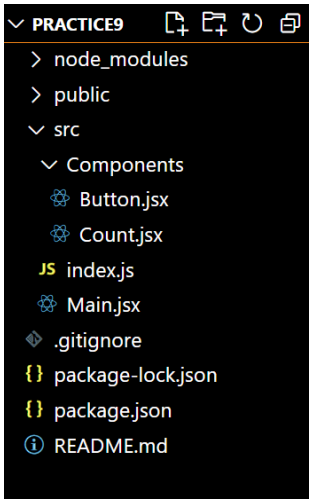
```
import React, { useContext } from 'react'
import { userData } from './A'

const C = () => {
   let user = useContext(userData)
  return (
    <div>
       <h1>{user} Good Morning</h1>
    </div>
  )
}
export default C
```

❒ useCallback-

- **useCallback** is a hook provided by React JS that memorizes a function so that it can be reused across re-renders of a component without causing unnecessary re-renders.
- When a function is defined within a component, it is recreated every time the component is rendered. This can be problematic if the function is used as a prop to child components that use memorization, as the child components will always re-render even if the props they receive have not changed.
- The **useCallback** hook takes two arguments: the function to memorize and an array of dependencies that the function depends on. If any of the dependencies change, the function is re-created. If none of the dependencies change, the memorized function is returned.
- Memorizing functions with **useCallback** can help improve performance in certain scenarios, such as when a component has a large number of child components that use memorization and receive functions as props.
- It is important to be careful when using **useCallback** and make sure that the dependencies array is set correctly. If the dependencies array is not set correctly, it can cause bugs and unexpected behaviour.
- **useCallback** can also be used in conjunction with the **useMemo** hook to further optimize performance by memorizing expensive computations that are used as props to child components.

- **Example –**
  - **Code for Button.jsx –**

```
import React from 'react'

const Button = (x) => {
   console.log(x)
  return (
    <div>
      <button onClick={x.Func}>{x.children}</button>
    </div>
  )
}
export default React.memo(Button)
```



Folder Structure

❖ **Code for Count.jsx –**

```
import React from 'react'

const Count = (props) => {
   console.log(props)
  return (
    <div>
      <h1>{props.children}:{props.data}</h1>
    </div>
  )
}

export default React.memo(Count)
```

❖ **Code for Main.jsx –**

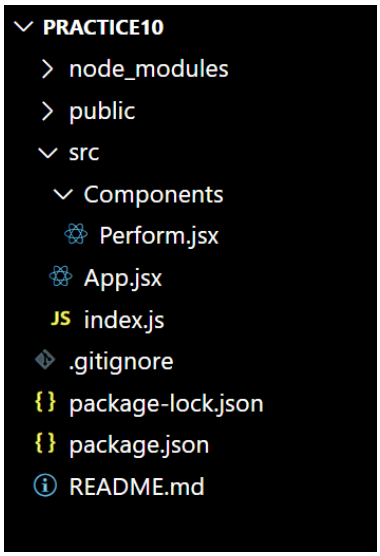
```
import React, { useCallback, useState } from 'react'
import Count from './Components/Count'
import Button from './Components/Button'

const Main = () => {
   let [age,setAge] = useState(25)
   let [sal, setSalary] = useState(30000)
   let increAge = useCallback(()=>{
      setAge(age+1)
   },[age])
   let increSal = useCallback(()=>{
      setSalary(sal+5000)
   },[sal])
  return (
    <div>
      <Count data={age}>Age </Count>
      <Button Func={increAge}>IncreAge</Button>
      <Count data={sal}>Salary </Count>
      <Button Func={increSal}>IncreSal</Button>
    </div>
  )
}
export default Main
```

❒ **<u>useMemo</u>-**

- The **useMemo** hook takes two arguments: a function and an array of dependencies.
- The function passed to **useMemo** is called a memorized function. This function is only executed when its dependencies change. If the dependencies don't change, the memorized value returned by the function is reused.
- The second argument to **useMemo** is an array of dependencies. If any of these dependencies change, the memorized function will be re-executed to produce a new memorized value.
- The memorized value returned by **useMemo** can be any value - a number, a string, an object, or even a function.
- **useMemo** is typically used to optimize the performance of expensive calculations or computations in a component. By memorizing the result of these calculations, React can avoid unnecessary re-renders of the component.

- **Example -**
  - **Code for Perform.jsx –**

```jsx
import { useState } from "react"
import { useMemo } from "react"

const Perform = ()=>{

   let [count1,setCount1]=useState(0)
   let [count2,setCount2]=useState(0)

   let incre1 = ()=>{setCount1(count1+1)}
   let incre2 = ()=>{setCount2(count2+1)}

   let Even = useMemo(()=>{
      let i = 0
      while (i < 1000000000) i++
      return count1%2==0
   },[count1])

   return(
      <div>
         {count1}
         <button onClick={incre1}>Increment1</button> <br />
         {Even?"Even":"Odd"} <br/>

         {count2}
         <button onClick={incre2}>Increment2</button> <br />
      </div>
   )
}

export default Perform
```



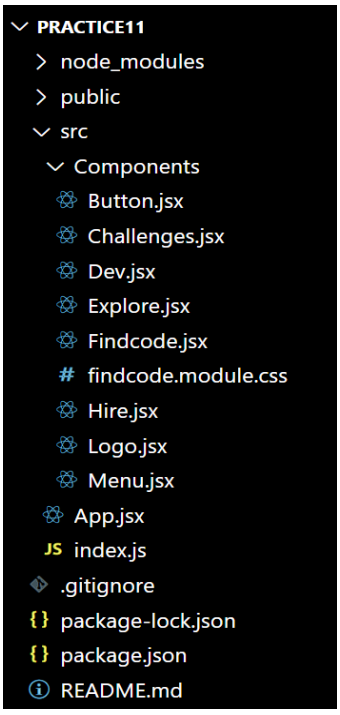
Folder Structure

❒ <u>React Router</u>-

- **What is a Router in React JS -** A router in React JS is a library that allows you to navigate between different pages or views within a single-page application (SPA). It manages the URLs of your application and maps them to the appropriate components to render.
- **Why do we need Router in React JS -** A router is needed in React JS to provide a seamless user experience when navigating through different parts of a SPA. Without a router, you would have to manually manage the state of the application and manually render different components based on the current URL.
- **How to use Router in React JS -** To use a router in React JS, you need to first install a router library such as **react-route-dom** using a package manager like npm. Once installed, you can import the necessary components from the library, such as **BrowserRouter**, **Routes**, **Route**, and

Link, and use them in your React components to define the routes and navigation behaviour of your application.

- **BrowserRouter** is a component that should wrap your entire application and provides the history and location objects to your components. The history object keeps track of the browser's history, and the location object contains information about the current URL.
- **Routes** is a component that wraps multiple **Route** components and renders **Route** that matches the current URL.
- **Route** is a component that defines a route in your application. You can specify the path for the route and the component that should be rendered when the route is matched.
- **Link** is a component that provides declarative, accessible navigation around your application. It creates an HTML anchor tag with the appropriate href attribute based on the specified path.
- **Nested-Routes -** You can also define nested routes in React JS by using the **Route** component inside another **Route** component. This allows you to create more complex routes and nested navigation in your application.
- **Route Parameters** - You can also define route parameters in React JS by adding a colon followed by a parameter name in the path of the **Route** component. The parameter value can then be accessed in the component that is rendered for that route.
- **Route-Guards** - You can also add route guards in React JS to protect certain routes from being accessed by unauthorized users. This can be done by adding a function that checks if the user is authenticated before allowing access to the route.



Project-1:

Single Page Demo Website project

(Website Name - FindCoder)

❒ **Package Installation command in terminal of Visual Studio –**

Router - *npm install react-router-dom*

❖ **Code for Button.jsx -**

```
import style from "./findcode.module.css"
const Button = ()=>{
   return(
      <div id={style.btn}>
         <button>Get Started</button>
      </div>
   )
}
export default Button
```

❖ **Code for Challenge.jsx -**

```
import React from 'react'

const Challenge=()=>{
  return (
    <div>
       <h1>Hello, I'm Challenges Component</h1>
    </div>
  )
}
export default Challenge
```

❖ Code for Dev.jsx -

```
import React from 'react'

const Dev=()=>{
 return (
   <div>
      <h1>Hello, I'm Dev Component</h1>
   </div>
 )
}
export default Dev
```

❖ Code for Explore.jsx -

```
import React from 'react'

const Explore=()=>{
 return (
   <div>
     <h1>Hello, I'm Explore Component</h1>
   </div>
 )
}
export default Explore
```

❖ Code for FindCode.jsx -

```
import Button from "./Button"
import Logo from "./Logo"
import Menu from "./Menu"
import style from "./findcode.module.css"

const Findcode = ()=>{
   return(
      <div>
         <section id={style.nav}>
            <article>
               <div className={style.Logo}> <Logo/> </div>
               <div className={style.Menu}> <Menu/> </div>
               <div className={style.Btn}> <Button/> </div>
            </article>
         </section>
      </div>
   )
}
export default Findcode
```

❖ Code for findcode.module.css -

```
*{
   margin: 0;
   padding: 0;
   box-sizing: border-box;
}
```

```css
#nav{
    height: 72.5px;
    width: 100%;
    display: flex;
    justify-content: center;

}

#nav>article{
    height: 72.5px;
    width: 95%;
    display: flex;
}

#nav>article>.Logo{
    height: 72.5px;
    width: 30%;
    /* background-color: aqua; */
}

#logo{
    height: 100%;
    width: 100%;
    display: flex;
    align-items: center;
    margin-left: 102px;
    cursor: pointer;
}

#logo>p{
    font-size: 25px;
    font-family: Arial, Helvetica, sans-serif;
    margin-left: 9px;
    color: rgb(45,151,234);
}

#nav>article>.Menu{
    height: 72.5px;
    width: 40%;
    /* background-color: red; */
}

#menu{
    height: 100%;
    width: 100%;

}

#menu>ol{
    height: 100%;
    list-style: none;
    display: flex;
    justify-content: space-around;
    align-items: center;
}
```

```
#menu>ol>li>a{
    text-decoration: none;
    font-size: 17px;
    font-family: Arial, Helvetica, sans-serif;
    font-weight:bold;
    color: rgb(115,121,128);
}


#nav>article>.Btn{
    height: 72.5px;
    width: 30%;
    /* background-color: brown; */
}

#btn{
    height: 100%;
    width: 100%;
    display: flex;
    justify-content: center;
    align-items: center;
}

#btn>button{
    height: 30px;
    width: 125px;
    font-size: 14px;
    color: white;
    background-color: rgb(23,124,226);
    border: none;
    border-radius: 5px;
    cursor: pointer;
}
```

❖ Code for Hire.jsx -

```
import React from 'react'

const Hire=()=>{
  return (
    <div>
        <h1>Hello, I'm Hire Component</h1>
    </div>
  )
}
export default Hire
```

❖ Code for Logo.jsx -

```
import style from "./findcode.module.css"
const Logo = ()=>{
   return(
      <div id={style.logo}>
         <svg width="38" height="22" viewBox="0 0 38 22" fill="none"
xmlns="http://www.w3.org/2000/svg"><path d="M0.356678 11.4956C-0.118893 11.2104 -0.118892
10.4976 0.356679 10.2125L14.7197 1.60122C15.4408 1.16889 16.1693 2.15119 15.5826
2.76473L8.34608 10.3321C8.06984 10.6209 8.06984 11.0871 8.34608 11.376L15.5826
18.9433C16.1693 19.5569 15.4408 20.5392 14.7197 20.1068L0.356678 11.4956Z"
fill="#2D88E2"></path><path class="fill-black dark:fill-white" d="M8.51066 1.55168C8.23967
1.27027 8.23967 0.824966 8.51066 0.543559C8.7965 0.246733 9.2717 0.246733 9.55753
0.543559L28.7255 20.4483C28.9964 20.7297 28.9964 21.175 28.7255 21.4564C28.4396 21.7533
27.9644 21.7533 27.6786 21.4564L8.51066 1.55168Z"></path><path d="M37.6433 10.2125C38.1189
10.4976 38.1189 11.2104 37.6433 11.4956L23.2803 20.1068C22.5592 20.5392 21.8307 19.5569
22.4174 18.9433L29.6539 11.376C29.9302 11.0871 29.9302 10.6209 29.6539 10.332L22.4174
2.76473C21.8307 2.15118 22.5592 1.16888 23.2803 1.60122L37.6433 10.2125Z"
fill="#2D88E2"></path></svg>
         <p>FindCoder</p>
      </div>
   )
}
export default Logo
```

❖ Code for Menu.jsx -

```
import { Link } from "react-router-dom"
import style from "./findcode.module.css"

const Menu = ()=>{
   return(
      <div id={style.menu}>
         <ol>
            <li><Link to="/explore">Explore Work</Link></li>
            <li><Link to="/hire">Hire Talents</Link></li>
            <li><Link to="/devboard">Dev Board</Link></li>
            <li><Link to="/challanges">Challenges</Link></li>
         </ol>
      </div>
   )
}

export default Menu
```

❖ Code for App.jsx -

```
import Explore from "./Components/Explore"
import Findcode from "./Components/Findcode"
import { BrowserRouter, Routes, Route } from "react-router-dom"
import Hire from "./Components/Hire"
import Dev from "./Components/Dev"
import Challenge from "./Components/Challenges"
const App = ()=>{
   return(
```

```
      <div>
        <BrowserRouter>
        <Findcode/>
        <Routes>
          <Route element={<Explore/>} path="/explore"></Route>
          <Route element={<Hire/>} path='/hire'></Route>
          <Route element={<Dev/>} path="/devboard"></Route>
          <Route element={<Challenge/>} path="/challanges"></Route>
        </Routes>
        </BrowserRouter>
      </div>
    )
}

export default App
```

❒ **<u>useParam</u>-**

- The ***useParams*** hook is a built-in hook provided by React Router that allows you to access parameters from the current URL.
- It returns an object containing key-value pairs of the parameters defined in the URL.
- To use ***useParams***, you must first import it from the ***react-route-dom*** library.
- You can then call ***useParams()*** inside a functional component to access the parameter values.
- The parameter values can be accessed using the names defined in the URL. For example, if you have a parameter named ***id*** in your URL, you can access its value using ***const {id}=useParams();***
- If the parameter does not exist in the URL, its value will be undefined.
- The ***useParams*** hook can be used in conjunction with other React Router hooks like ***useRouteMatch***, ***useLocation*** and ***useHistory***.
- It is important to note that ***useParams*** should only be used inside a component that is rendered by a ***Route*** component, otherwise it will not work.

❒ **<u>useNavigate</u>-**

- The ***useNavigate*** hook is a built-in hook provided by React Router that allows you to navigate programmatically between different pages or components.
- It returns a ***navigate*** function that can be called with a URL or a location descriptor to navigate to a different page or component.
- To use ***useNavigate***, you must first import it from the ***react-route-dom*** library.
- You can then call ***useNavigate()*** inside a functional component to access the ***navigate*** function.
- The ***navigate*** function can be called with a string representing the URL you want to navigate to, or with a location object that contains the path, search parameters, and state.
- The ***navigate*** function can also accept an options object as a second argument, which can be used to customize the navigation behaviour.
- The ***useNavigate*** hook is useful for navigating programmatically in response to user actions, such as button clicks or form submissions.
- It is important to note that ***useNavigate*** should only be used inside a component that is rendered by a ***Route*** component, otherwise it will not work.
- Overall, the ***useNavigate*** hook is a powerful tool for navigating programmatically between different pages or components in React JS.

Project-2:

CRUD Operation

(Create – Read – Update – Delete)

- **Package Installation command in terminal of Visual Studio –**
  - **axios -** *npm install axios*.
  - **Router -** *npm install react-router-dom*
  - **JSON Server -** *npm install json-server*
- **Command to connect UI with JSON Server –**
  npx json-server Backend/db.json –watch port 4000
  (Port number can be anything except 3000 ***)***
- We have to use two terminals –
  - 1st terminal for JSON Server.
  - 2nd terminal for npm start
- **Project File GDrive Link:-**
  - Project File Download Link - https://tinyurl.com/354ru4ue
  - After download this file (.zip file), Extract it.
  - Open it in VS Code.
  - Open New Terminal.
  - Type ***"npm i"*** to install ***node_modules package.***
  - After that type ***"npm start".***

--- Happy Coding ---