ComponentDidMount

This is used to render component is create/mount after it will working on or calling on. Just like:- html, css, javascript they are components.

And there used in calling for API.

App.js:-

import './App.css';

import React from 'react';

class App extends React.Component {

    constructor(){

        super();

        console.warn("constructor")

    }

    componentDidMount(){

        console.warn("componentDidMount")

    }

    render(){

        console.warn("render")

        return(

            <div className="App">

                <h1>Hello hello</h1>

            </div>

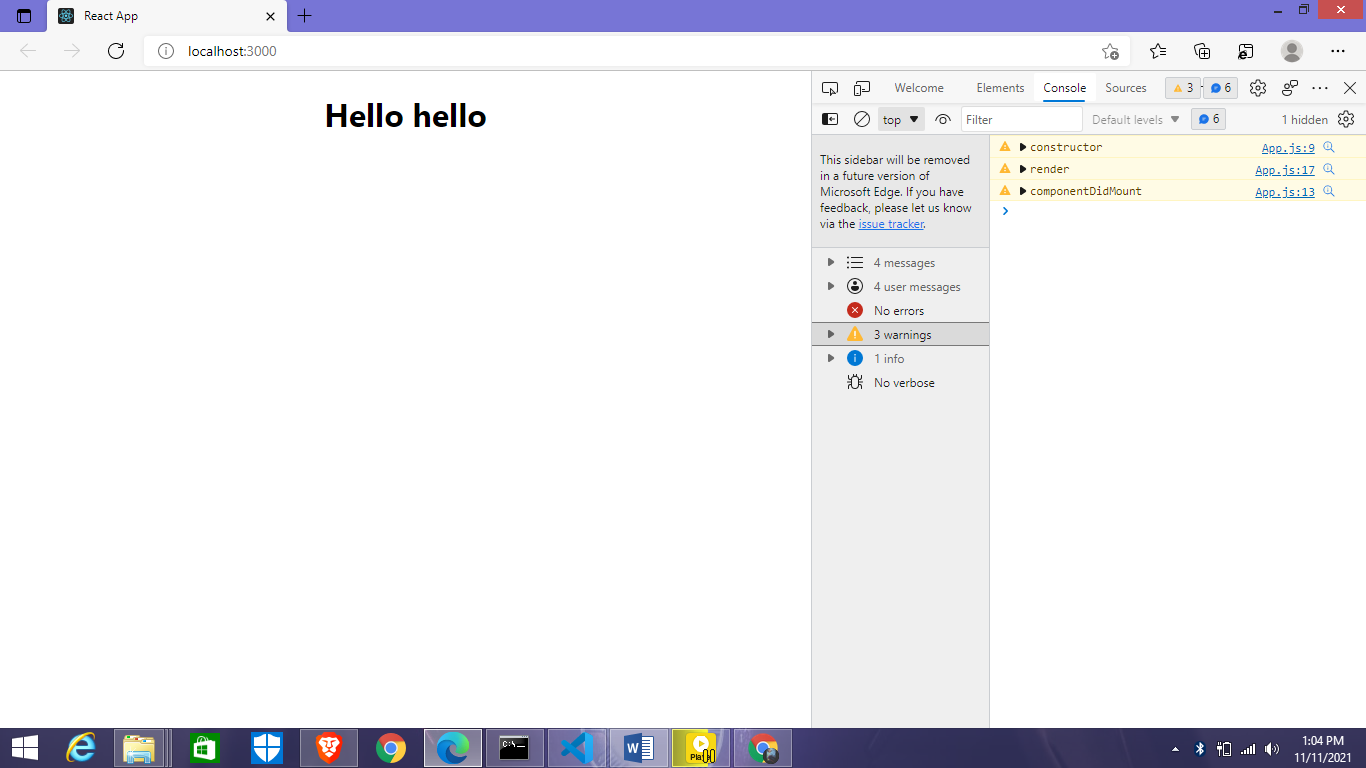
        );

    }

}

export default App;

Just like that: -



Example 2:

import './App.css';

import React from 'react';

class App extends React.Component {

    constructor(){

        super();

        this.state={

            name:"lucky"

        }

    }

    componentDidMount(){

        console.warn("componentDidMount")

    }

    render(){

        console.warn("render")

        return(

            <div className="App">

                <h1>Hello hello {this.state.name}</h1>

                <button onClick={()=>this.setState({name:"bandhey"})}>update</button>

            </div>

        );

    }

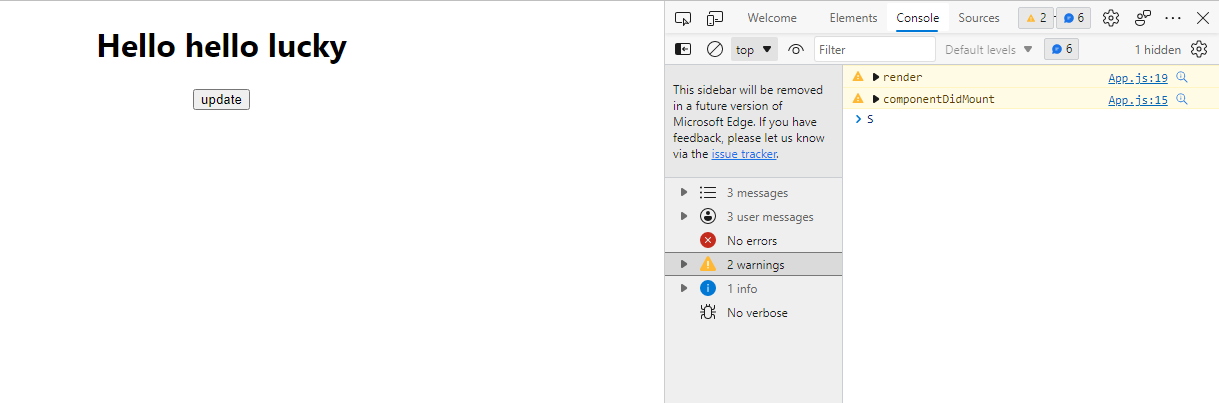
}

export default App;

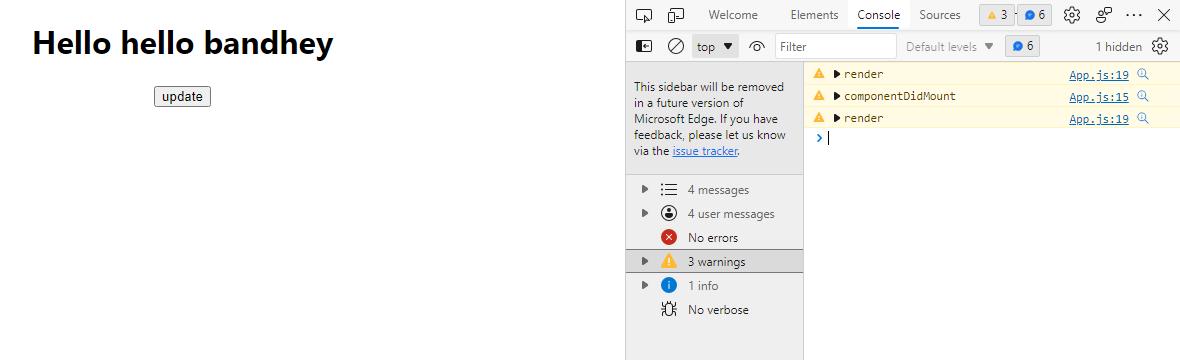
Mainly render is first call than componentDidMount is Call but when if render will update so componentDidMount isn’t call because the componentDidMount is Mounting phase it isn’t existing the Updating phase.

And there isn’t depend of state and props.

Case 1:before render isn’t update.



Case 2: render component was update.



State and Props are same we are equally receive.

ComponentDidUpdate:-

import './App.CSS;

import React from 'react';

class App extends React.Component {

    constructor(){

        super();

        this.state={

            num:0

        }

         console.warn("constructor")

    }

    componentDidUpdate(preProps,preState,snapshot){

        console.warn("componentDidUpdate",preState.num,this.state.num)

        if(preState.num===this.state.num)

        {

            alert("it's already same")

        }

    }

    render(){

        console.warn("render")

        return(

            <div className="App">

                <h1>Hello hello {this.state.num}</h1>

                <button onClick={()=>{this.setState({num:1})}}>update</button>

            </div>

        );

    }

}

export default App;

shouldComponentUpdate

There is confirmation method for component should update and actually it is stopping a rendering for users continue updating the state and props.

import './App.css';

import React from 'react';

class App extends React.Component {

    constructor(){

        super();

        this.state={

            count:0

        }

    }

    shouldComponentUpdate()

    {

        console.warn("shouldComponentUpdate",this.state.count);

        if(this.state.count>5 && this.state.count<10){

            return true; // states are update is show for ‘return true’

        }

    }

    render(){

        return(

            <div className="App">

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

                <button onClick={()=>{this.setState({count:this.state.count+1})}}>update</button>

            </div>

        );

    }

}

export default App;

componentWillUnmount

App.js

import './App.css';

import React from 'react';

import User from './User';

class App extends React.Component{

    constructor()

    {

        super();

        this.state={

            show:true

        }

    }

    render()

    {

        return(

            <div className="App">

                {

                    this.state.show?<User/>:<h1>child component remove</h1>

                }

                <button onClick={()=>this.setState({show:!this.state.show})}>Toggle</button>

            </div>

        );

    }

}

export default App;

User.js

import React from "react";

class User extends React.Component{

    componentWillUnmount(){

        alert("componentWillUnmount call")

    }

    render()

    {

        return(

            <div>

                <h1>hello</h1>

            </div>

        );

    }

}

export default User;

Example:- how to call

import React from "react";

class User extends React.Component{

    componentWillUnmount(){

        console.warn("componentWillUnmount")

    }

    render()

    {

        return(

            <div>

                <h1>hello</h1>

            </div>

        );

    }

}

export default User;

Hooks

We can use class component features in functional components such as state, life cycle, pure component etc.

import './App.css';

import React,{useState} from 'react'; // {useState} there is hooking process.

function App() {

    const [data,setData]=useState("alpha")

    return(

        <div className="App">

            <h1>{data}</h1>

            <button onClick={()=>setData("gama")}>update</button>

        </div>

    );

}

export default App;

useEffect

This hook is used like componentDidMount, ComponentDidUpdate, componentWillUnmount.

**Using props:-**

App.js

import './App.css';

import React,{useState} from 'react';

import User from './User';

function App(){

    const[data,setData]=useState(10)

    const[count,setCount]=useState(10)

    return(

        <div className="App">

            <User data={data} count={count} />

            <button onClick={()=>setData(data+1)}>Update Data</button>

            <button onClick={()=>setCount(count+1)}>Update Count</button>

        </div>

    );

}

export default App;

**User.js**

import React from "react";

function User(props){

    React.useEffect(()=>{

        alert("hello")

    },[props.data]) // this calling for data property

    return(

        <div>

            <h1>DATA props :{props.data}</h1>

            <h1>COUNT props :{props.count}</h1>

        </div>

    );

}

export default User;

useMemo

It’s used to close multi-time functions calls or unwanted calls.

import './App.css';

import React, { useState, useMemo } from 'react';

function App() {

    const [count, setCount] = useState(0);

    const [item, setItem] = useState(0);

    const multuCountMemo = useMemo(function sumCount() {

        console.warn("count")

        return count \*\* 2

    }, [count])

    return (

        <div className="App">

            <h1>data: {count}</h1>

            <h1>data: {item}</h1>

            <h1>Data Count Power: {multuCountMemo}</h1>

            <button onClick={() => setCount(count + 1)}>Update Count</button>

            <button onClick={() => setItem(item + 1)}>Update Item</button>

        </div>

    );

}

export default App;

Ref in React.js

It’s used to events calls any functional components but react isn’t accept this Ref function because this is directly manipulate in DOM and React.js is slow

This function you can create DOM events and can call.

But extremely required for project so that can use.

import './App.css';

import React,{createRef} from 'react';

class App extends React.Component{

    constructor()

    {

    super();

    this.inputRef=createRef();

    }

    componentDidMount(){

        // console.warn(this.inputRef.current.value="yh")

    }

    getValue(){

        console.warn(this.inputRef.current.value)

    }

    render()

    {

        return(

            <div className="App">

                <h1>hello</h1>

**-**                <button onClick={()=>this.getValue()}>Update</button>

            </div>

        );

    }

}

export default App;

using functional component

import './App.css'

import React,{useRef} from 'react'

export default function App(){

   const inputRef = useRef(null)

   function getRef() {

       console.warn("function call")

       inputRef.current.value="100"

       inputRef.current.focus();

   }

    return(

        <div className="App">

            <h1>hello: </h1>

            <input type="text" ref={inputRef}/>

            <button onClick={getRef}>Change</button>

        </div>

    );

}

cls

It’s similar to Ref but any child components data to import parent component so using forwardRef hook.

import './App.css'

import React,{useRef} from 'react'

import User from './User';

export default function App(){

  const inputRef=useRef(null);

  function updateRef(){

       inputRef.current.value="100"

  }

    return(

        <div className="App">

            <h1>hello: </h1>

            <User ref={inputRef}/>

            <button onClick={updateRef}> Change</button>

        </div>

    );

}

User.js

import React, { Fragment,forwardRef } from "react";

 function User(props,dataRef){

    return(

        <Fragment>

           <input type="text" ref={dataRef}/>

        </Fragment>

    )

}

export default forwardRef(User);

useContext

Context API uses Context. Provider and Context. Consumer Components pass down the data but it is very cumbersome to write the long functional code to use this Context API. So useContext hook helps to make the code more readable, less verbose and removes the need to introduce Consumer Component. The useContext hook is the new addition in React 16.8.

**App.js**

import React, { useState } from "react";

import Auth from "./Auth";

import AuthContext from "./auth-context";

const App = () => {

  //using the state to dynamicallly pass the values to the context

  const [authstatus, setauthstatus] = useState(false);

  const login = () => {

    setauthstatus(true);

  };

  return (

    <React.Fragment>

      <AuthContext.Provider value={{ status: authstatus, login: login }}>

        <Auth />

      </AuthContext.Provider>

    </React.Fragment>

  );

};

export default App;

**auth-context.js**

import React from 'react';

// Creating the context object and passing the default values.

const authContext = React.createContext({status:null,login:()=>{}});

export default authContext;

**Auth.js**

import React, { useContext } from "react";

import AuthContext from "./auth-context";

const Auth = () => {

// Now all the data stored in the context can

// be accessed with the auth variable

const auth = useContext(AuthContext);

console.log(auth.status);

return (

  <div>

  <h1>Are you authenticated?</h1>

  {auth.status ?

<p>Yes you are</p>

:

<p>Nopes</p>

}

  <button onClick={auth.login}>Click To Login</button>

  </div>

);

};

export default Auth;

useReducer

The **useReducer** Hook is the better alternative to the **[useState](https://www.geeksforgeeks.org/reactjs-usestate-hook/)** hook and is generally more preferred over the **useState** hook when you have complex state-building logic or when the next state value depends upon its previous value or when the components are needed to be optimized.

The **useReducer** hook takes three arguments including reducer, initial state, and the function to load the initial state lazily.

App.js

import React, { useReducer } from "react";

// Defining the initial state and the reducer

const initialState = 0;

const reducer = (state, action) => {

switch (action) {

  case "add":

  return state + 1;

  case "subtract":

  return state - 1;

  case "reset":

  return 0;

  default:

  throw new Error("Unexpected action");

}

};

const App = () => {

  // Initialising useReducer hook

const [count, dispatch] = useReducer(reducer, initialState);

return (

  <div>

  <h2>{count}</h2>

  <button onClick={() => dispatch("add")}>

    add

  </button>

  <button onClick={() => dispatch("subtract")}>

    subtract

  </button>

  <button onClick={() => dispatch("reset")}>

    reset

  </button>

  </div>

);

};

export default App;

useCallback

The useCallback is a react hook that returns a memoized callback when passed a function and a list of dependencies as parameters. It’s very useful when a component is passing a callback to its child component to prevent the rendering of the child component. It only changes the callback when one of its dependencies gets changed.

App.js

import React, { useCallback, useState} from "react"

import List from "./List"

function App(){

    {/\* Initial states \*/}

    const [input, setInput] = useState(1);

    const [light, setLight] = useState(true);

    {/\* useCallback memoizes the getItems() which

       returns a list of number which is number+10

       and number + 100 \*/}

    const getItems = useCallback(() => {

        return [input + 10, input + 100];

    }, [input]);

    {/\* style for changing the theme \*/}

    const theme = {

        backgroundColor: light ? "White": "grey",

        color: light ? "grey" : "white"

    }

    return <>

        {/\* set the theme in the parent div \*/}

        <div style={theme}>

          {/\* When we input a number it is stored in

            our stateful variable \*/}

            <input type="number"

            value={input}

            onChange={event =>

            setInput(parseInt(event.target.value))

            } />

            {/\* on click the button the theme is set to

            the opposite mode, light to dark and vice versa\*/}

            <button onClick={() =>

            setLight(prevLight =>

            !prevLight)}>{light ? "dark mode":"light mode"}

            </button>

            <List getItems={getItems} />

        </div>

    </>;

}

export default App;

useCustomHook

React.Js provides lots of built-in hooks that you can use in your React apps. But besides them, you can make your own custom hooks and use it in your apps resulting in better readability and a reduced amount of code. Custom hooks are normal JavaScript functions whose names start with “use” and they may call other hooks(built-in or custom).

**The need for Custom Hooks:**The main reason for which you should be using Custom hooks is to maintain the concept of [DRY](https://www.geeksforgeeks.org/7-common-programming-principles-that-every-developer-must-follow/)(Don’t Repeat Yourself) in your React apps. For example, suppose you have some logic that makes use of some built-in hooks and you need to use the logic in multiple functional components. So, there are two ways left for you — 1) write the same logic in each and every component (which violates the concept of DRY) and 2) create a separate function that wraps the logic inside it and then call it from those components. The second option is undoubtedly a better choice as you have to write the logic only once. Here, the separate function you created is the custom hook. So, whenever you feel that you have a logic that is to be used in multiple functional components(hooks don’t work in class components), just create a separate custom hook for it and use it.

**Building a custom hook:**Creating a custom hook is the same as creating a JavaScript function whose name starts with “use”. It can use other hooks inside it, return anything you want it to return, take anything as parameters. The function “useCustomHook” in the below example, is a custom hook that uses a [state variable](https://www.geeksforgeeks.org/reactjs-usestate-hook/) “counter”. The function “resetCounter” increases the value of counter by 1 and whenever the value of the counter updates, the function passed to the [useEffect](https://www.geeksforgeeks.org/reactjs-useeffect-hook/) hook is called.  The function executes some code(removed as to focus on how to use custom hooks rather than implementing a logic) that will be used in multiple components. This custom hook returns the “resetCounter” function.

**App.js**

import React from 'react';

import './App.css';

import FirstComponent from './components/FirstComponent';

import SecondComponent from './components/SecondComponent';

function App(){

    return(

        <div className='App'>

        <FirstComponent />

        <SecondComponent />

        </div>

    );

}

export default App;

**useCustomHook.js**

import {useState , useEffect} from "react";

// Remember to start the name of your custom hook with "use"

function useCustomHook(initializer , componentName){

    const [counter , setCounter] = useState(initializer);

    // Increases the value of counter by 1

       function resetCounter(){

        setCounter(counter + 1);

    }

    useEffect(() => {

        // Some logic that will be used in multiple components

        console.log("The button of the "

        + componentName + " is clicked "

        + counter + " times.");

    } , [counter , componentName]);

    // Calls the useEffect hook if the counter updates

    return resetCounter;

}

export default useCustomHook;

**FirstComponent.js**

import React from "react";

// importing the custom hook

import useCustomHook from "../useCustomHook";

function FirstComponent(props){

  // ClickedButton = resetCounter;

  const clickedButton = useCustomHook(0 , "FirstComponent");

  return (

    <div>

      <h1> This is the First Component</h1>

      <button onClick={clickedButton}>

        Click here!

      </button>

    </div>

  );

}

export default FirstComponent;

**SecondComponent.js**

import React from "react";

// Importing the custom hook

import useCustomHook from "../useCustomHook";

function SecondComponent(props){

  // ClickedButton = resetCounter;

  const clickedButton = useCustomHook(0 , "SecondComponent");

  return (

    <div>

      <h1> This is the Second Component</h1>

      <button onClick={clickedButton}>

      Click here!

      </button>

    </div>

  );

}

export default SecondComponent;

**Using Css in React.js**

import './App.css';

import './style.css';

import style from './style.module.css';

import React from 'react';

function App() {

    return (

        <div className="App">

            <h1 className="hello1">Hello</h1> {/\*\* this class is import to style.css \*/}

            <h1 style={{ backgroundColor: "black", color: "goldenrod" }}>Hello</h1> {/\*\*And it's used to JSX script for define style \*/}

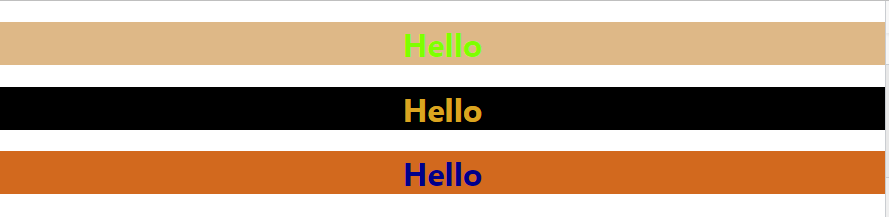
            <h1 className={style.hello2}>Hello</h1>{/\*\* this used similar to class hello1 but it's style.module.css this is css module. \*/}

        </div>

    );

}

export default App;

****

Using bootstrap and Installation

[React-Bootstrap · React-Bootstrap Documentation (react-bootstrap-v4.netlify.app)](https://react-bootstrap-v4.netlify.app/getting-started/introduction)

First of all open your terminal and install bootstrap and choose the installation versions.

Install command

npm install react-bootstrap [bootstrap@4.6.0](mailto:bootstrap@4.6.0)

[bootstrap@4.6.0](mailto:bootstrap@4.6.0) that is bootstrap version

then installation is completed so you can go App.js and open to import bootstrap event from ‘react-bootstrap’

Like that: - Button, Alert, etc.

import { Button } from 'react-bootstrap';

Example:-

import './App.css';

import React from 'react';

import { Button } from 'react-bootstrap';

function App() {

    return (

        <div className="App">

            <Button onClick={() => alert("hello")}>Click me</Button>

        </div>

    );

}

export default App;

Handle Array with list in map function & for loop

React.js is use map function because return statement are not accept or not identify the for loop.

Show the lists on react so it’s mainly using map function.

import './App.css';

import React from 'react';

function App() {

    const students=['ali', 'kavi', 'sanjay', 'azad']

    //map looping

    students.map((items)=>{

        console.warn("students list :" +items)

    });

    //for looping

    for(var i=0;i<students.length;i++){

        console.warn("the For loop students list  :" +students[i])

    }

    return(

        <div className="App">

            {/\* {

                students.map((name)=><h1>{name}</h1>)

            } \*/}

        </div>

    )

}

export default App;

Into return component

import './App.css';

import React from 'react';

function App() {

    const students=['ali', 'kavi', 'sanjay', 'azad']

    return(

        <div className="App">

            {

                students.map((name)=><h1>{name}</h1>)// return statement into map function

            }

            {/\* {

                for(var i=0;i<students.length;i++){

                    <h1>{students[i]}</h1>

                }

            } \*/}

        </div>

    );

}

export default App;

Array Object

import './App.css';

import React from 'react';

function App() {

    // const students=['ali', 'kavi', 'sanjay', 'azad']

    const items = [

        { name: 'abhi', email: "abhi@123", number: 0 },

        { name: 'baba', email: "baba@123", number: 111 },

        { name: 'syam', email: "syam@123", number: 222 },

        { name: 'chal', email: "chal@123", number: 333 }

    ]

    return (

        <div className="App">

            <table border="5" cellPadding="50">

                <tr>

                       <th>Name</th>

                       <th>Email</th>

                       <th>Number</th>

                   </tr>

           {

               items.map((details)=>

               <tr>

               <th>{details.name}</th>

               <th>{details.email}</th>

               <th>{details.number}</th>

                </tr>

               )

           }

           </table>

        </div>

    );

}

export default App;

Bootstrap filter table

import './App.css';

import React from 'react';

import { Table } from 'react-bootstrap'

function App() {

    // const students=['ali', 'kavi', 'sanjay', 'azad']

    const items = [

        { name: 'abhi', email: "abhi@123", number: 111 },

        { name: 'baba', email: "baba@123", number: 111 },

        { name: 'syam', email: "syam@123", number: 222 },

        { name: 'chal', email: "chal@123", number: 111 }

    ]

    return (

        <div className="App">

            <Table striped hover variant="dark">

                <tbody>

                    <tr>

                        <th>Name</th>

                        <th>Email</th>

                        <th>Number</th>

                    </tr>

                    {

                        items.map((details, file) =>

                        details.number === 222 ?<tr key={file}>

                            <td>{details.name}</td>

                            <td>{details.email}</td>

                            <td>{details.number}</td>

                        </tr>:null

                        )

                    }

                </tbody>

            </Table>

        </div>

    );

}

export default App;

Nested Array on table

import './App.css';

import React from 'react';

import { Table } from 'react-bootstrap'

function App() {

    // const students=['ali', 'kavi', 'sanjay', 'azad']

    const items = [

        {

            name: 'abhi', email: "abhi@123", add: [

                { house: 10, city: 'raipur', number: 0 },

                { house: 24, city: 'bhilai', number: 32 },

                { house: 45, city: 'raigarh', number: 67 },

            ]

        },

        {

            name: 'baba', email: "baba@123", add: [

                { house: 10, city: 'raipur', number: 0 },

                { house: 24, city: 'bhilai', number: 32 },

                { house: 45, city: 'raigarh', number: 67 },

            ]

        },

        {

            name: 'syam', email: "syam@123", add: [

                { house: 10, city: 'raipur', number: 0 },

                { house: 24, city: 'bhilai', number: 32 },

                { house: 45, city: 'raigarh', number: 67 },

            ]

        },

        {

            name: 'chal', email: "chal@123", add: [

                { house: 10, city: 'raipur', number: 0 },

                { house: 24, city: 'bhilai', number: 32 },

                { house: 45, city: 'raigarh', number: 67 },

            ]

        }

    ]

    return (

        <div className="App">

            <Table striped variant="dark">

                <tbody>

                    <tr>

                        <th>S No.</th>

                        <th>Name</th>

                        <th>Email</th>

                        <th>Address</th>

                    </tr>

                {

                    items.map((data, i) =>

                        <tr key={i}>

                            <td>{i+1}</td>

                            <td>{data.name}</td>

                            <td>{data.email}</td>

                            <td>

                                <Table  variant="dark">

                                    <tbody>

                                        <tr>

                                            <th>House no.</th>

                                            <th>City</th>

                                            <th>Number</th>

                                        </tr>

                                    {data.add.map((sum, e) =>

                                        <tr key={e}>

                                            <td>{sum.house}</td>

                                            <td>{sum.city}</td>

                                            <td>{sum.number}</td>

                                        </tr>

                                    )}

                                </tbody>

                                </Table>

                            </td>

                        </tr>

                    )

                }

                </tbody>

            </Table>

        </div>

    );

}

export default App;