* react is not a framework
* it is js library for building user interfaces
* also known as reactJS and react.js
* developed by jorden walke, a software engineer at google
* first deployed for facebook News feed around 2011
* in 2013, react was open sourced at JS conference

***Installation :-***

install node

npm will be installed itself

install nvm

***install create-react-app globally :-***

npm install -g create-react-app

***to check version of react app :-***

npx create-react-app --version

***now we can create react app anywhere but we also need internet connection :-***

create-react-app appName

***\*\**** app name should not contain capital letter otherwise it will give error and not create it

create-react-app .

to create app within a given folder

* webPack makes the bundle of our project is also installed by itself
* babel converts the ECMA5 code into older version which is supported by browser is also installed by itself

***JSX***

JSX in present in 'react' module

JSX is html inside js used to make skeleton of our webpage

JSX has same attributes as that of HTML tags but they are in camel case inside JSX.

When we compile the react file, React.createElement() is called by itself inside ReactDOM.reder() as first argument

javascript inside HTML which is itself inside javascript is called 'javascript expression'

we can use only values to provide on web browser but we cannot use statements inside it

ReactDOM.render(<h1>hello world</h1>,document.querySelector('#root'));

converted to

ReactDOM.render(React.createElement('h1',null,'hello world'),document.querySelector('#root'));

this is all done using react and babel but we can also do all this with pure js in this way but it is lengthy:-

const elem = document.createElement('h1');

elem.innerHTML = 'hello world';

document.querySelector('root').appendChild(elem);

***how to show multiple elemetsrender():-***

***#1 ---> enclose them in a tag***

ReactDOM.render(

<div>

<h1>hello world</h1>

<h1>hello world</h1>

<h1>hello world</h1>

</div>,

document.querySelector('#root')

);

***#2 ---> react version above v16.x.x can render array of elements***

ReactDOM.render(

[

<h1>hello world</h1>,

<h1>hello world</h1>,

<h1>hello world</h1>

],

document.querySelector('#root')

);

\*\*using array method is confusing

\*\*using <div></div> method adds an extra div inside root so it can be a problem while applying CSS so instead use React.Fragment

\*\* coz it is faster, uses less memory and does not adds any div tag

***#3 ---> using React.Fragment method***

ReactDOM.render(

<React.Fragment>

<h1>hello world</h1>

<h1>hello world</h1>

<h1>hello world</h1>

</React.Fragment>,

document.querySelector('#root')

);

***\*\* more easier way for it (short hand of React.Fragment):--->***

ReactDOM.render(

<>

<h1>hello world</h1>

<h1>hello world</h1>

<h1>hello world</h1>

</>,

document.querySelector('#root')

);

***Passing values to the JSX :-***

import React from 'react';

import ReactDOM from 'react-dom';

const name = 'Abhinav Garg';

ReactDOM.render(<h1>hello {name}</h1>,document.querySelector('#root')); **// correct**

ReactDOM.render(<h1>hello {console.log('hello')}</h1>,document.querySelector('#root')); **// incorrect**

***CSS in React***

***INLINE CSS: -*** done in ***kebab*** case (***text-tranform --->textTransform***) by passing an object or writing as object

and all values are passed as string

**\*\*** we doing it in same way we do so in **vanilla js**

import React from 'react';

import ReactDOM from 'react-dom';

ReactDOM.render(<h1 style = {{margin: '10px', textTransform: 'capitalize'}}>hello</h1>,document.querySelector('#root'));

***APP.js***

App.jsx is the master component that holds all data and calls all other components

After collecting all data from all small components it makes a one single big component and sends it to index.js

Then index.js renders it using ReactDOM.reder()

***index.js --->***

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

ReactDOM.render(

<App/>,

document.querySelector('#root')

);

***app.jsx --->***

import React from 'react';

import Heading from './heading';

import List from './list';

function App() {

return (

<>

<Heading />

<List />

</>

);

}

export default App;

***heading.jsx --->***

import React from 'react';

function Heading(){

return(

<h1>hello</h1>

);

}

export default Heading;

***list.jsx --->***

import React from 'react';

function List(){

return(

<ol>

<li>A</li>

<li>B</li>

<li>C</li>

<li>D</li>

<li>E</li>

</ol>

);

}

export default List;

***Exporting Data in React :-***

**default ek hi rhta hai baki non-default ek sath export hote hai**

***data.jsx --->***

const fname = 'abhinav';

const lname = 'garg';

function age() {

return 22;

}

export default fname;

export { lname, age };

***App.jsx --->***

***#1 :***

import fname,{lname,age} from './data';

function App() {

return (

<>

{fname}{lname}{age()}

</>

);

}

export default App;

***App.jsx --->***

***#2***

import \* as data from './data';

function App() {

<>

{data.default}{data.lname}{data.age()}

</>

);

}

export default App;

***Components***

They are isolated and reusable piece of code or custom elements that holds html data to render on browser. They are responsible for UI and UX of react based web application.

we can give them attributes which are called props(properties) that can be user-defined or custom attributes.

***Props*** these are the user defined attributes provided to any component. The receiving component will receive it in form of an object.

***Keying :-***

While building a list of components then we need to give them a unique property to recognize them specifically using ***key*** attribute

If we do not do so then it will give just a warning.

***Data.jsx***

*const* data = [

  {

    id: 1,

    name: "abhinav",

    company: "Amazon",

  },

  {

    id: 2,

    name: "Shristi",

    company: "infosys",

  },

];

export default data;

***App.jsx***

***using map method on array of objects with callback function present outside it***

import React from "react";

import Card from "./Card";

import data from "./data";

*function* singleCard(*d*)

{

  return <*Card* key={*d*.id} name={*d*.name} company={*d*.company}></*Card*>

}

*function* App() {

  return <>{data.map(singleCard(d))}</>;

}

export default App;

***using map method on array of objects with callback function present inside it***

import React from "react";

import Card from "./Card";

import data from "./data";

*function* App() {

  return (

    <>

      {data.map((*val*) *=>* (

        <*Card* key={*val*.id} name={*val*.name} company={*val*.company}></*Card*>

      ))}

    </>

  );

}

export default App;

***if-else conditional***

import React from "react";

import Card from "./Card";

import data from "./data";

*function* App() {

  return (

    <>

      {data.map((*val*) *=>* {

        if (*val*.id & 1) {

          return (

            <*Card* key={*val*.id} name={*val*.name} company={*val*.company}></*Card*>

          );

        } else {

          return console.log(*val*);

        }

      })}

    </>

  );

}

export default App;

***ternary operator***

import React from "react";

import Card from "./Card";

import data from "./data";

*function* App() {

  return (

    <>

      {data.map((*val*) *=>* {

        return *val*.id & 1 ? (

          <*Card* key={*val*.id} name={*val*.name} company={*val*.company}></*Card*>

        ) : (

          console.log(*val*)

        );

      })}

    </>

  );

}

export default App;

***HOOKS***

1. introduced in react 16.8

2. it allows you to use ***state*** and other react features without writing a class. Hooks are the functions which 'hook into' React state and life cycle features from function components.

3. it does not work inside classes

4. hooks should always be used at the top level of React function.

5. node version 6 or above and NPM version 5.2 or above for using hooks.

***useState :-***

import React, { useState } from "react";

*function* App() {

*const* [fname, setFirstName] = useState("");

*const* [lname, setLastName] = useState("");

*const* onLastNameChange = (*event*) *=>* {

    setLastName(*event*.target.value);

  };

  return (

    <>

      <form>

        <h1>{fname}</h1>

        <input

          type="text"

          onChange={(*event*) *=>* {

            setFirstName(*event*.target.value);

          }}

          value={fname}

        ></input>

        <h1>{lname}</h1>

        <input type="text" onChange={onLastNameChange} value={lname}></input>

      </form>

    </>

  );

}

export default App;

***Controlled Component :-***

A component which is bound to a value and it value is updated with setState() method on every call of event based callback (like onChange() in input field).

***Un-Controlled Component :-***

A component which is similar to simple html form input element. It is handled by DOM itself. It maintain its own state and its value is updated when input value changes.

***Form with controlled components :-***

import React, { useState } from "react";

*function* App() {

*const* [fname, setFirstName] = useState("");

*const* [lname, setLastName] = useState("");

*const* [data, setData] = useState({});

*const* onLastNameChange = (*event*) *=>* {

    setLastName(*event*.target.value);

  };

*const* onSubmitForm = (*event*) *=>* {

*event*.preventDefault();

    setData({ fname: fname, lname: lname });

  };

  return (

    <>

      <form onSubmit={onSubmitForm}>

        <h1>{fname}</h1>

        <input

          type="text"

          onChange={(*event*) *=>* {

            setFirstName(*event*.target.value);

          }}

          value={fname}

        ></input>

        <h1>{lname}</h1>

        <input type="text" onChange={onLastNameChange} value={lname}></input>

        <br></br>

        <input type="submit"></input>

      </form>

      <br></br>

      <br></br>

      <br></br>

      <input

        type="button"

        value="show"

        onClick={() *=>* {

          console.log(data);

        }}

      ></input>

    </>

  );

}

export default App;

***Handling too many useStates values***

import React, { useState } from "react";

*function* App() {

*const* [data, setData] = useState({

    fname: "",

    lname: "",

    age: ""

  });

*const* changeData = (*event*) *=>* {

    setData((*prevValue*) *=>* {

      if (*event*.target.name === "fname") {

        return {

          fname: *event*.target.value,

          lname: *prevValue*.lname,

          age: *prevValue*.age

        };

    } else if (*event*.target.name === "lname") {

      return {

        fname: *prevValue*.fname,

        lname: *event*.target.value,

        age: *prevValue*.age

      }

    } else if (*event*.target.name === "age") {

      return {

        fname: *prevValue*.fname,

        lname: *prevValue*.lname,

        age: *event*.target.value

      };

      }

    });

  };

  return (

    <>

      <form>

        <input

          type="text"

          name="fname"

          value={data.fname}

          onChange={changeData}

        />

        <br></br>

        <input

          type="text"

          name="lname"

          value={data.lname}

          onChange={changeData}

        />

        <br></br>

        <input

          type="text"

          name="age"

          value={data.age}

          onChange={changeData}

        />

      </form>

    </>

  );

}

export default App;

first way is we can use if else condition with previous value of our useState() as everything is object

***optimized way is to use spread operator***

import React, { useState } from "react";

*function* App() {

*const* [data, setData] = useState({

    fname: "",

    lname: "",

    age: "",

  });

*const* changeData = (*event*) *=>* {

    setData((*prevValue*) *=>* {

      return {

        ...*prevValue*,

        [*event*.target.name]: *event*.target.value,

      };

    });

  };

  return (

    <>

      <form>

        <input

          type="text"

          name="fname"

          value={data.fname}

          onChange={changeData}

        />

        <br></br>

        <input

          type="text"

          name="lname"

          value={data.lname}

          onChange={changeData}

        />

        <br></br>

        <input type="text" name="age" value={data.age} onChange={changeData} />

      </form>

    </>

  );

}

export default App;

***useContext :-***

React Context API allows to easily access data at different level of component tree, without passing props to every level.

***App.jsx***

import React, { createContext } from "react";

import ComponentA from "./CompA";

*const* Fname = createContext();

*const* Lname = createContext();

*function* App() {

  return (

    <>

      <*Fname.Provider* value={"first name"}>

        <*Lname.Provider* value={"last name"}>

          <*ComponentA* />

        </*Lname.Provider*>

      </*Fname.Provider*>

    </>

  );

}

export default App;

export { Fname, Lname };

***CompA.jsx***

import React from "react";

import ComponentB from "./CompB";

*function* ComponentA() {

  return (

    <>

      <*ComponentB*></*ComponentB*>

    </>

  );

}

export default ComponentA;

***CompB.jsx***

import React, { useContext } from "react";

import { Fname, Lname } from "./App";

*function* ComponentB() {

*const* fname = useContext(Fname);

*const* lname = useContext(Lname);

  return (

    <>

      <h1>

        {fname} -- {lname}

      </h1>

    </>

  );

}

export default ComponentB;

***export default React.memo(ComponentName) :-***

there is a rule in react that if parent is re-rendered then its child will also re-render even if there if no changes happened in child component.

So it will create a problem in large scale apps where a single parent component can have multiple child components and those can have their child components and so on.

To optimize it we use React.memo() that tell parent that if there is any changes happened in child component then only re-render it

***useEffect :-***

useEffect is called by itself every time render method of index.js is called or each time DOM is rendered

It performs the working of ***componentDidMount***, ***componentDidUpdate***, ***componentWillUnmount***

***it will be called everytime page is rendered :-***

  useEffect(() *=>* {

    console.log("first useEffect");

  });

***it will be called only first time page is rendered :-***

  useEffect(() *=>* {

    console.log("second useEffect");

  }, []);

***it will be called only when value of num (which is hook(useState)) is changed :-***

  useEffect(() *=>* {

    console.log("third useEffect");

  }, [num]);

***\*\*we can use it to perform the operation after any state or components of our web app is changed i.e. every time React.render() of index.js is called.***

***useEffect with clean up function :-***

always use clean up function when use setInterval in useEffect otherwise it will create an infinite loop

***wrong code :-***

*const* [number, setNumber] = useState(1);

  useEffect(()*=>*{

    console.count("effect runs");

    setInterval(() *=>* {

      setNumber((*prevValue*) *=>* *prevValue* + 1);

    }, 1000);

  }, [number])

***Correct code :-***

*const* [number, setNumber] = useState(1);

  useEffect(() *=>* {

    console.count("effect run")

*const* intervalObj = setInterval(() *=>* {

      setNumber((*prevValue*) *=>* *prevValue* + 1);

    }, 1000);

    return () *=>*{

      clearInterval(intervalObj)

    }

  }, []);

***AXIOS***

**we use axios package for calling data from API**

***installation :-***

npm install axios

***API is always called in useEffect:------->***

import React, { useState, useEffect } from "react";

import axios from "axios";

*function* App() {

*const* [name, setName] = useState("initial");

  useEffect(() *=>* {

    async *function* getData() {

*const* res = await axios.get(`https://pokeapi.co/api/v2/pokemon/5`);

      setName(res.data.name);

    }

    getData();

  }, []);

  return (

    <>

      <h1>{name}</h1>

    </>

  );

}

export default App;

***Using axios with clean up function***

  useEffect(() *=>* {

*const* axiosCancelTokenObj = axios.CancelToken.source();

*let* url = "";

    axios

      .get(url, { cancelToken: axiosCancelTokenObj.token })

      .then((*res*) *=>* {

        console.log(*res*.data);

      })

      .catch((*e*) *=>* {

        if (axios.isCancel(*e*)) {

          console.log("cancel");

        } else {

          console.log(*e*);

        }

      });

    return () *=>* {

      axiosCancelTokenObj.cancel();

    };

  }, []);

***Server Side Rendering VS Client Side Rendering***

***simple html, css, js*** file se bani web app ***server side routing*** krti hai when clicked on any link( anchor tag )

web app made using ***react route with navlink does client side routing***

web app made using ***react route with node js does server side routing***

* server side routing refreshes the whole page
* client side routing renders or changes the components and makes the single page application

Almost all of the web app made using react perform client side routing.

* client side routing does not have any layer of authenticity while we can provide it with js and api but it will be difficult
* server side routing has layer of authenticity and it is easy. It also provides us provision of session control.

***React Router***

***Index.js***

import React from "react";

import ReactDOM from "react-dom";

import App from "./App";

ReactDOM.render(

  <>

    <*App* />

  </>,

  document.querySelector("#root")

);

***App.jsx***

import "./App.css";

import { BrowserRouter, Routes, Route } from "react-router-dom";

import Parent1 from "./COMPONENTS/Parent1";

import Parent2 from "./COMPONENTS/Parent2";

import HomePage from "./COMPONENTS/HomePage";

import Child1 from "./COMPONENTS/CHILDS/Child1";

import Child2 from "./COMPONENTS/CHILDS/Child2";

*function* App() {

  return (

    <>

      <*BrowserRouter*>

        <*Routes*>

          <*Route* path="/" element={<*HomePage*></*HomePage*>}></*Route*>

          <*Route* path="/Parent1" element={<*Parent1*></*Parent1*>}></*Route*>

          <*Route* path="/Parent2" element={<*Parent2*></*Parent2*>}>

            <*Route* path="Child1" element={<*Child1*></*Child1*>}></*Route*>

            <*Route* path="Child2" element={<*Child2*></*Child2*>}></*Route*>

          </*Route*>

        </*Routes*>

      </*BrowserRouter*>

    </>

  );

}

export default App;

***Parent2.js***

import React from "react";

import { Outlet } from "react-router-dom";

export default *function* Parent2() {

  return (

    <>

      <div>Parent2</div>

      <*Outlet*></*Outlet*>

    </>

  );

}

***NavLink***

import React from "react";

import { NavLink } from "react-router-dom";

import "./index.css";

*function* Menu() {

  return (

    <>

      <*NavLink*

        to="/"

        className={({ *isActive* }) *=>* (*isActive* ? "active" : "inactive")}

      >

        Home Page

      </*NavLink*>

      <*NavLink*

        to="/About"

        className={({ *isActive* }) *=>* (*isActive* ? "active" : "inactive")}

      >

        About Page

      </*NavLink*>

      <*NavLink*

        to="/Contact"

        className={({ *isActive* }) *=>* (*isActive* ? "active" : "inactive")}

      >

        Contact Page

      </*NavLink*>

    </>

  );

}

export default Menu;

***Navigation button***

import React from "react";

import { useNavigate } from "react-router-dom";

*function* HomePage() {

*const* navigate = useNavigate();

  return (

    <>

      <h1>this is Home page</h1>

      <button

        onClick={() *=>* {

          navigate("/About");

        }}

      >

        go to about page

      </button>

    </>

  );

}

export default HomePage;

***useParams***

***App.jsx***

import React from "react";

import { BrowserRouter as Router, Route, Routes } from "react-router-dom";

import Menu from "./Menu";

import ErrorPage from "./ErrorPage";

import UserPage from './UserPage';

*function* App() {

  return (

    <>

      <*Router*>

        <*Menu*></*Menu*>

        <*Routes*>

          <*Route* path="/User/:fname/:lname" element={<*UserPage* />} />

          <*Route* path="\*" element={<*ErrorPage* />} />

        </*Routes*>

      </*Router*>

    </>

  );

}

export default App;

***Menu.jsx***

import React from "react";

import { NavLink } from "react-router-dom";

import "./index.css";

*function* Menu() {

  return (

    <>

      <*NavLink*

        to="/User/firstName/lastName"

        className={({ *isActive* }) *=>* (*isActive* ? "active" : "inactive")}

      >

        User Page

      </*NavLink*>

    </>

  );

}

export default Menu;

***UserPage.jsx***

import React from "react";

import { useParams } from "react-router-dom";

*function* UserPage() {

*const* { fname, lname } = useParams();

  return (

    <>

      <h1>

        {fname} -- {lname}

      </h1>

    </>

  );

}

export default UserPage;

***useLocation***

import React from "react";

import { useParams, useLocation } from "react-router-dom";

*function* UserPage() {

*const* { fname, lname } = useParams();

*const* data = useLocation();

  console.log(data);

  return (

    <>

      <h1>

        {fname} -- {lname}

      </h1>

    </>

  );

}

export default UserPage;

***output:***

1. **hash**: ""
2. **key**: "he9wiqa0"
3. **pathname**: "/User/firstName/lastName"
4. **search**: ""
5. **state**: null

***Showing images***

Make folder in public and add all images into it

We have direct access in public folder

***dataImage.jsx***

*const* sources = ['images/1.jpg', 'images/2.jpg']

export default sources;

***HomePage.jsx***

import React from "react";

import data from "./dataImages";

*function* Card(*props*) {

  return (

    <>

      <img src={*props*.source} alt={*props*.source} height={100} width={200} />

    </>

  );

}

*function* HomePage() {

  return (

    <>

      {data.map((*src*) *=>* {

        return <*Card* source={*src*} />;

      })}

    </>

  );

}

export default HomePage;

in order to send data btw two components which are rendered with help of routes(one at a time), we need to build all working methods in their common parent component and then pass the data as props in react tree

but we can make it easy using history

history can be used for both page navigations as well as data transfer btw sibling components directly

**how to call it**

import { useHistory } from "react-router-dom";

**changing url and sending data**

    history.push({

      pathname:`/product/${product.id}`,

      state: product

    })

**Accessing data in called component**

const SingleItem = (props) => {

  const current=props.location.state;

console.log(current);

}