React

react is a java script library.
main focus is building UI develop fast asap
used to create single-page-application
means a complete website in a single page.

https://react.dev

pages not reloaded so fast.

It is a component-based front-end library responsible only for the view layer of a Model View Controller(MVC) architecture. React is used to create modular user interfaces and promotes the development of reusable UI components that display dynamic data.

Note: React is not a framework. It is just a library developed by Facebook .
Each component in a React application is responsible for rendering a separate, reusable piece of HTML code.

While building a client-side application, a team of Facebook developers found that DOM is slow. Document Object Model (DOM) is an application programming interface(API) for HTML and XML documents. It defines the logical structure of documents and how a document is accessed and manipulated). To make it faster, React implements a virtual DOM which is basically a DOM tree representation in JavaScript, and React was invented.

The current stable version of ReactJS is 18.2.0 and released on June 14, 2022 and the first release was on May 29, 2013.

ReactJS Environment Setup

Downloading the Node.js
https://nodejs.org/en/download

install then finish

cmd open
node -v (Verify that Node.js was properly installed or not.

)
If node.js was completely installed on your system, the command prompt will print the version of the node.js installed.

create folder
react_learn
open is vs code
npm install
then select folder - open terminal - npx create-react-app demo (folder name)

npx create-react-app demo
cd demo
npm start

NPX stands for Node Package eXecute. It is simply an NPM package runner. It allows developers to execute any Javascript Package

Running the Server
npm start

NPM is short for node package manager, an online directory that contains the various already registered open-source packages

ReactJS JSX Introduction

JSX stands for **JavaScript XML**. JSX is basically a syntax extension of JavaScript. It helps us to write HTML in JavaScript and forms the basis of React Development.

- It is faster than normal JavaScript as it performs optimizations while translating to regular JavaScript.
- It makes it easier for us to create templates.

- Instead of separating the markup and logic in separate files, React uses *components* for this purpose. We will learn about components in detail in further articles.
- As JSX is an expression, we can use it inside of if statements and for loops, assign it to variables, accept it as arguments, or return it from functions.

Note :
react used virtual DOM because update only required list .
DOM complete list update.

used app
Youtube
whatsapp
netflix

most important file in react - package.json
(name , version ,installed package , react version also check here)
**** Application detail maintain , project detail**

package-lock.json– version , dependency , frame , package -history maintain

index.js - react entry point file

reportWebVitals – project performance report
call in index.js

public folder - html kind of detail

manifest.json – icon theme store

robot.txt – (no use in react)search engine protect

node module - react package store

install react validation

```
npm i react-validation
```

react bootstrap

```
npm i react-bootstrap
```

react-router-dom

```
npm i react-router-dom
```

Components

Components are independent and reusable bits of code.
A Component is one of the core building blocks of React. In other words, we can say that every application you will develop in React will be made up of pieces called components. Components make the task of building UIs much easier. UI broken down into multiple individual pieces called components and work on them independently and merge them all in a parent component which will be your final UI.

Example

header
footer
create folder : pages

header.jsx
footer.jsx
services.jsx
carrier.jsx

and call in app.js

class component
function component

render Method

The `render()` method is then called to define the React component that should be rendered.

user.jsx

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

class User extends Component{
    render() {
      return (
        <div>
          <h1>hellllo world !!!!!</h1>
        </div>
      )
    }
}
export default User
```

call in App.js

```
import './App.css';
import User from './Pages/Users';

function App() {
  return (
    <div className="App">
      <User/>
    </div>
  );
}

export default App;
```

alert msg without click button

User.jsx

```
function User()
{
    function Apple()
    {
      alert("function called");
```

```
    }

  return(
    <>
    <div>
      <button onClick={Apple()}>Click Me</button>
 </div>
    </>
  )
}
export default User
```

alert msg using click button

```
function User()
{
    function Apple()
    {
      alert("function called");
    }

  return(
    <>
    <div>
       // <button onClick={Apple}>Click Me</button>
// using arrow ()
      <button onClick={()=>alert("hellooo")}>Click me</button>

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

React-router-dom

React Router Dom is used to build single-page applications i.e. applications that have many pages or components but the page is never refreshed instead the content is dynamically fetched based on the URL.

React Router Dom has many useful components and to create fully functioning routing, you need most of these.

(BrowserRouter , Router , Route)

1. **Router(usually imported as BrowserRouter): It is the parent component that is used to store all of the other components. Everything within this will be part of the routing functionality .**

2. Route: This component checks the current URL and displays the component associated with that exact path. All routes are placed within the switch components.

3. Link: Link component is used to create links to different routes.

install react-router-dom

```
npm i react-router-dom
```

```
install bootstrap
```

```
npm install bootstrap
```

```
pages create
```

Home

About us

Contact Us

Services

Const use in useState bcz const reassigned value .

Home.jsx

```
import React from 'react';
import { Link } from 'react-router-dom';
export default function Header() {
  return (
    <div>
      {/* <a href="/">Home</a>
      <a href="">About Us</a>
      <a href="">Contact Us</a>
      <a href="">Service</a> */}
      <ul>
        <li><Link to="/">Home</Link></li>
        <li><Link to="/About">About Us</Link></li>
        <li><Link to="/Contact">Contact Us</Link></li>
        <li><Link to="/Services">Services</Link></li>
      </ul>
    </div>
  )
}
```

App.js

```
import './App.css';
import React from 'react';
import {BrowserRouter , Routes,Route} from "react-router-dom";
import About from './pages/About';
import Contact from './pages/Contact';
import Header from './pages/Header';
import Home from './pages/Home';
import Services from './pages/Services';
```

```
function App() {
  return (
   <>
      <BrowserRouter>
        <Routes>
              <Route path="/" element={<Home/>}></Route>
              <Route path="/About" element={<About/>}></Route>
              <Route path="/Contact" element={<Contact/>}></Route>
              <Route path="/Services" element={<Services/>}></Route>
         </Routes>
</BrowserRouter>
    {/* <Header/>
    <Home/>
    <About/>
    <Contact/>
    <Services/> */}
</>
    );
}
export default App;
```

List and key

React Keys

A key is a unique identifier. In React, it is used to identify which items have changed, updated, or deleted from the Lists. It is useful when we dynamically created components or when the users alter the lists.

A “key” is a special string attribute you need to include when creating lists of elements in React. Keys are used in React to identify which items in the list are changed, updated, or deleted.

Keys are used to give an identity to the elements in the lists. It is use to string as a key that uniquely identifies the items in the list.

Keys are internal to React and can not be accessed from inside of the component.

React Lists

Lists are used to display data in an ordered format and mainly used to display menus on websites.

map():

The `map()` method creates a new array with the results of calling a function for every array element.

The map() function is used for traversing the lists and updates elements to be enclosed between <li> elements.

The Javascript map() method in JavaScript creates an array by calling a specific function on each element present in the parent array.This method doesn't change the original array.

for example

```
<html>
<body>
<script>
var arr=[2.1,3.5,4.7];
var result=arr.map(Math.round);
document.writeln(result);
</script>
</body>
</html>
```

```
<html>
<body>
<script>
var arr=[2,4,6];
var result=arr.map(x=>x*3);
document.writeln(result);
</script>
</body>
</html>
```

map()

```
<script>
  const numbers = [1, 2, 3, 4, 5];
  const num = numbers.map((number) => number * 2);
document.write(num);
</script>
```

JSON

In React , work with JSON data that needs to be passed into components for rendering dynamic content.

JSON (JavaScript Object Notation) is a lightweight data interchange format for storing and exchanging data. It consists of key-value pairs and is widely supported by different programming languages and APIs.

{

key<– “Name” : “varsha Shringi” , →value

“Age” : 30,

“City” : “Bundi” ,

“Country” : “India”

}

Dynamic Header using json

```
import React from "react";
const Dhead=[
    {
```

```
        id:1,
        title:"Home",
        path:"/",
    },
    {
        id:2,
        title:"Service",
        path:"/Services",
    },
    {
        id:3,
        title:"About",
        path:"/About",
    },
    {
        id:4,
        title:"Contact",
        path:"/Contact"
    }
]
export default Dhead
```

Dynamic Header

```
import React from 'react';
import { Link } from 'react-router-dom';
import Dhead from './Header';

export default function Header() {
  return (
    <>
    {
        Dhead.map((items)=>{
          return(
            <>

              <li key={items.id}>
                <Link to={items.path}>{items.title}</Link>
              </li>

            </>
          )
```

```
        })
}
        {/* <a href="/">Home</a>
        <a href="">About Us</a>
        <a href="">Contact Us</a>
        <a href="">Service</a> */}
        <ul>
                    {/* <li><Link to="/About">About Us</Link></li>
          <li><Link to="/Contact">Contact Us</Link></li>
          <li><Link to="/Services">Services</Link></li> */}
        </ul>
    </>
    )
  }
```

Create table in react js

A table is an arrangement that divides information into rows and columns. Its purpose is to store and display structured data. React-Table is used for this purpose in react. React-table is an open-source library that allows you to create tables in the React framework. It is a lightweight, fast, and fully customizable.

Table.jsx

```
import React from 'react'

const Table = () => {
  return (
    <div className="App">
    <table>
        <tr>
            <th>Name</th>
            <th>Age</th>
            <th>Gender</th>
        </tr>
        <tr>
            <td>Anom</td>
            <td>19</td>
            <td>Male</td>
        </tr>
        <tr>
            <td>Megha</td>
```

```
            <td>19</td>
            <td>Female</td>
        </tr>
        <tr>
            <td>Subham</td>
            <td>25</td>
            <td>Male</td>
        </tr>
    </table>
</div>
)
}
export default Table
```

add in json file pr app.js

Example 2

```
import React from 'react'

const Table = () => {
  return (
    <div className="App">
    <h1>Students Table</h1>
    <table>
      <thead>
        <tr>
          <th>Name</th>
          <th>Age</th>
          <th>Gender</th>
          <th>Roll Number</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>Akram</td>
          <td>21</td>
          <td>Male</td>
          <td>2019MEB1235</td>
        </tr>
        <tr>
          <td>Jason</td>
          <td>22</td>
```

```
          <td>Male</td>
          <td>2018CSB1234</td>
        </tr>
        <tr>
          <td>Dave</td>
          <td>20</td>
          <td>Female</td>
          <td>2019eeb1242</td>
        </tr>
        <tr>
          <td>Tom</td>
          <td>20</td>
          <td>Male</td>
          <td>2019mmb1235</td>
        </tr>
        <tr>
          <td>Stark</td>
          <td>20</td>
          <td>Male</td>
          <td>2019meb1290</td>
        </tr>
      </tbody>
    </table>
  </div>

  )
}

export default Table
```

index.css

```
table {
  width: 800px;
  height: 200px;
  border-collapse: collapse;
}

th {
  border-bottom: 1px solid black;
  border: 2px solid black;
}
```

```
td {
  text-align: center;
  border: 2px solid black;
}
```

Form in react

Form.jsx

```
import React from 'react'

const Form = () => {

return (

<>

<div className="App">

            <h1>Form in React</h1>

            <fieldset>

                <form>

                    <label for="firstname">First Name*</label>

                    <input type="text" name="firstname"

                  id="firstname" placeholder="Enter First Name"

                    required />

                    <br /><br />

                    <label for="lastname">Last Name*</label>

                    <input

                        type="text"

                        name="lastname"

                        id="lastname"

                        placeholder="Enter Last Name"

                        required
```

```
                    />
                    <br /><br />
                    <label for="email">Enter Email* </label>
                    <input
                        type="email"
                        name="email"
                        id="email"
                        placeholder="Enter email"
                        required
                    />
                    <br /><br />
                    <label for="tel">Contact*</label>
                    <input
                        type="tel"
                        name="tel"
                        id="tel"
                        placeholder="Enter Mobile number"
                        required
                    />
                    <br /><br />
                    <label for="gender">Gender*</label>
                    <br />
                    <input type="radio" name="gender"
                        value="" id="male" />
```

```
                    Male
                    <input type="radio" name="gender"
                        value="" id="female" />
                    Female
                    <input type="radio" name="gender"
                        value="" id="other" />
                    Other
                    <br /><br />
                    <label for="lang">Your best Subject</label>
                    <br />
                    <input type="checkbox" name="lang"
                        id="english" checked />
                    English
                    <input type="checkbox" name="lang"
                        id="maths" />
                    Maths
                    <input type="checkbox" name="lang"
                        id="physics" />
                    Physics
                    <br /><br />
                    <label for="file">Upload Resume*</label>
                    <input
                        type="file"
                        name="file"
```

```
                    id="file"
                    placeholder="Enter Upload File"
                    required
                />
                <br /><br />
                <label for="url">Enter URL*</label>
                <input
                    type="url"
                    name="url"
                    id="url"
                    placeholder="Enter url"
                    required
                />
                <br /><br />
                <label>Select your choice</label>
                <select name="select" id="select">
                    <option value="" disabled selected>
                        Select your Ans
                    </option>
                    <optgroup label="Beginers">
                        <option value="1">HTML</option>
                        <option value="2">CSS</option>
                        <option value="3">JavaScript</option>
                    </optgroup>
```

```
                    <optgroup label="Advance">
                        <option value="1">React</option>
                        <option value="2">Node</option>
                        <option value="3">Express</option>
                        <option value="4">MongoDB</option>
                    </optgroup>
                </select>
                <br /><br />
                <label for="about">About</label>
                <br />
                <textarea
                    name="about"
                    id="about"
                    cols="30"
                    rows="10"
                    placeholder="About your self"
                    required
                ></textarea>
                <br /><br />
                <label>Submit OR Reset</label>
                <br />
                <button type="reset" value="reset">
                    Reset
                </button>
```

```
                    <button type="submit" value="Submit">

                        Submit

                    </button>

                </form>

            </fieldset>

        </div>

</>

  )

}

export default Form
```

index.css

Form css

```
body {

  background-color: azure;

  color:#fff;

}

h1 {

  text-align: center;

  color: black;

}

fieldset {

  margin-left: 20%;

  margin-right: 20%;

  background-color: rgb(23, 49, 41);
```

```
}

input {
  padding: .3rem;
  border: 1px solid #ccc;
  border-radius: 4px;
  margin: 10px;
}

label {
  margin: 10px;
}

button {
  padding: .3rem;
  font-size: 15px;
  background-color: rgb(9, 17, 6);
  border: 4px solid #ccc;
  color: #ccc;
  padding: 10px;
  border-radius: 8px;
  margin: 10px;
  cursor: pointer;
}
```

Types of components in ReactJS

1.Class component

2.functional Component

Functional Components: Functional components are simply javascript functions. We can create a functional component in React by writing a javascript function.

In the functional Components, the return value is the JSX code to render to the DOM tree.

Example

```
function demoComponent() {
   return (
          <h1>
               Welcome Message!
           </h1>);
}
```

Class Components: The class components are a little more complex than the functional components.We can pass data from one class component to another class component.

Example

```
class Democomponent extends React.Component {
   render() {
       return (
<>
<h1>Welcome Message!</h1>;
</>
)
   }
}
```

Functional Components vs Class Components:

Functional Components	Class Components
A functional component is just a plain JavaScript pure function that accepts props as an argument and returns a React element(JSX).	A class component requires you to extend from React. Component and create a render function that returns a React element.
There is no render method used in functional components.	It must have the render() method returning JSX (which is syntactically similar to HTML)
Functional components run from top to bottom and once the function is returned it can't be kept alive.	The class component is instantiated and different life cycle method is kept alive and is run and invoked depending on the phase of the class component.
Also known as Stateless components as they simply accept data and display them in some form, they are mainly responsible for rendering UI.	Also known as Stateful components because they implement logic and state.
React lifecycle methods (for example, componentDidMount) cannot be used in functional components.	React lifecycle methods can be used inside class components (for example, componentDidMount).

Hooks can be easily used in functional components to make them Stateful.
 	
Example:
 	
`const [name,SetName]= React.useState(' ')`
	It requires different syntax inside a class component to implement hooks.
	
	Example:
	
	`constructor(props) {`
	
	`super(props);`
	
	`this.state = {name: ' '}`
	
	`}`

Constructors are not used.	Constructor is used as it needs to store state.

class component example

```
import React from "react";

class About extends React.Component {

render() {

  return (

   <>

    <h1>hellllooo world</h1>

   </>

 )
```

```
}

}

export default About
```

React Hooks

Hooks were added to React in version 16.8. Hooks allow function components to have access to state and other React features.

It allows you to use state and other React features without writing a class. Hooks are the functions which "hook into" React state and lifecycle features from function components. It does not work inside classes.

Rules of Hooks

Only call Hooks at the top level

Do not call Hooks inside loops, conditions, or nested functions. Hooks should always be used at the top level of the React functions. This rule ensures that Hooks are called in the same order each time a components renders.

2. Only call Hooks from React functions

Note : in react data not update using variable , react need state to update their data.

Hooks State

Hook state is the new way of declaring a state in React app. Hook uses useState() functional component for setting and retrieving state

Built-in Hooks

Basic Hooks

- **useState**

- **useEffect**
- **useContext**

Additional Hooks

- **useReducer**
- **useCallback**
- **useMemo**
- **useRef**

useState : is a hook that allow state update

state **The state is a built-in React object that is used to contain data or information about the component. A component's state can change over time; whenever it changes, the component re-renders.**

The React useState Hook allows us to track state in a function component. State generally refers to **data or properties that need to be tracking in an application.**

About.jsx

after click state change

```
import React from 'react'

import { useState } from 'react'

const About = () => {

const[data,setdata]=useState("varsha")

//const[data,setdata]=useState("")
```

```
const Update=()=>{

    setdata("IT Trainer of ReactJS");

}

  return (

    <>

    <div className="App">

          <h1>{data}</h1>

          <button onClick={Update}>Update Data</button>

</div>

    </>

  )

}

export default About
```

Counter program using useState

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

const Counter = () => {

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

  const inc=()=>{

    setCount(count+1);
```

```
  }

  const dec=()=>{

    setCount(count-1);

  }

  return (

    <>

    <div>

    <h1> {count}</h1>

      <button onClick={inc}>+</button>

           <button onClick={dec}> -</button>

    </div>

    </>

  )

}

export default Counter
```

color program using useState

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

function Counter() {

  const [color, setColor] = useState("red");
```

```
 return (
    <>
      <h1>My favorite color is {color}!</h1>
      <button type="button" onClick={() => setColor("blue")}
      >Update Color</button>
    </>
  )
}
export default Counter
```

Counter program count times

```
import React, { useState } from 'react';
function Counter() {
  const [click, setClick] = useState(0);
 return (
    <>
    <div>
      <p>You clicked {click} times</p>
      <button onClick={() => setClick(click + 1)}> Click me </button>
    </div>
```

```
    </>
  );
}
export default Counter
```

Note The state is a built-in React object that is used to contain data or information about the component. A component's state can change over time; whenever it changes, the component re-renders.React requires state variables as it does not keep track of local variables, so when you attempt to modify them, React does not re-render the DOM. In React, state is a JavaScript object that stores data that can be modified over time

Form.jsx

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

function Form() {

  const [data, setData] = useState({

    username: '',

    password: ''

  });

  const [form, setForm] = useState({

    username: '',

    password: ''
```

```
  });

  const [submit, submitted] = useState(false);

  const printValues = e => {

    e.preventDefault();

    setForm({

      username: data.username,

      password: data.password

    });

    submitted(true);

  };

  const updateField = e => {

    setData({

      ...data,

      [e.target.name]: e.target.value

    });

  };

  return (

    <div>

      <form onSubmit={printValues}>
```

```
        <label> Username: <input value={data.username}
name="username"onChange={updateField}

          />

        </label>

        <br />

        <label> Password: <input value={data.password}name="password"
type="password" onChange={updateField}

          />

        </label>

        <br />

        <button>Submit</button>

      </form>

      <p>{submit ? form.username : null}</p>

      <p>{submit ? form.password : null}</p>

    </div>

  );

}

export default Form;
```

steps count program using useState

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

```
function Counter() {

  const [steps, setSteps] = useState(0);

 function increment() {

    setSteps(steps => steps + 1);

  }

  return (

    <div>

      Today you've taken {steps} steps!

      <br />

      <button onClick={increment}>

        I took another step

      </button>

    </div>

  );

}

export default Counter
```

Form.jsx

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

function Form() {
```

```
  const [username, setUsername] = useState('');

  const [password, setPassword] = useState('');

  const printValues = e => {

    e.preventDefault();

    console.log(username, password);

  };

  return (

    <form onSubmit={printValues}>

      <label>

        Username:

        <input value={username} onChange={event =>
setUsername(event.target.value)}

          name="username" type="text" />

      </label>

      <br />

      <label>

        Password:

        <input value={password} onChange={event =>
setPassword(event.target.value)}

          name="password" type="password" />
```

```
      </label>

      <br />

      <button>Submit</button>

    </form>

  );

}

export default Form
```

`preventDefault`

to avoid a page refresh and prints out the form values.

Contact.jsx

```
import React from 'react';

import { useForm} from 'react-hook-form';

function Contact() {

  const {register, handleSubmit} = useForm();

 const onSubmit = (data) =>{

    console.log(data);

  }

  return (

      <>
```

```
    <form onSubmit={handleSubmit(onSubmit)}>

    Firts Name  <input {...register('firstName')} /> <br />

    {/* register an input */}

     Last Name : <input {...register('lastName')} /><br />

       <input type="submit" />

    </form>

    </>

  );

}

export default Contact;
```

Image Upload (contact.jsx)

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

function Contact() {

  const [selectedFile, setSelectedFile] = useState(null);

const handleFileChange = (event) => {

    const file = event.target.files[0];

    setSelectedFile(file);

  };

const handleFileUpload = () => {
```

```
    if (selectedFile)

    {

 console.log('Selected file:', selectedFile);

}

    else

    {

      console.log('No file selected.');

    }

  };

 return (

    <div>

      <h1>Image Upload Example</h1>

      <input type="file" accept="image/*" onChange={handleFileChange}
/>

      <button onClick={handleFileUpload}>Upload Image</button>



      {selectedFile!=null && (

        <div>

         <img src={URL.createObjectURL(selectedFile)}

 alt="Selected" style={{ maxWidth: '100%' }}    />
```

```
        </div>

      )}

    </div>

  );

}

export default Contact;
```

users.jsx (class component state used)

```
import React from "react";

class User extends React.Component{

constructor()

{

  super();

  this.state={

    data:"hellllo World"

  }

}

apple()

{
```

```
  // alert ("fruit")

this.setState({data:"Rajasthan"})




}

render()

{

return(

  <>

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

<button onClick={()=>this.apple()}>Update data</button>

</>

);

}

}

export default User
```

use react-hook-form

```
npm install react-hook-form
```

React Hook Form is one such library that helps to manage complex forms. It has excellent performance, is super lightweight, has zero dependencies, can be easily integrated with different React UI.

Import *useForm* hook from react-hook-form. It will return your *register*, *handlesubmit*, *errors* methods.

- register: This is used to handle input fields. We can access the input field later using the name given to it.

```
<input {...register("firstname")} />
```

- handlesubmit: Is used to handle the form submission. It takes a custom method (eg: onSubmit). It will automatically collect field values.

```
const onSubmit = data =>  console.log(data);

<form onSubmit={handleSubmit(onSubmit)}>

     // input field 1

     // input field 2

     <input type="submit" />

</form>
```

- errors: We use this to handle errors. if we leave “firstname” field empty it will set errors.first name = true

```
<input {...register("firstname", { required: true })} />

{errors.firstname && <span>This field is required</span>}
```

Contact.jsx

```
import React from "react";

import { useForm } from "react-hook-form";

function Contact() {

    const { register, handleSubmit, formState: { errors } } =
useForm();

const onSubmit = (data) => console.log(data);

 return (

        <>

            <p className="title">Registration Form</p>

          <form className="App" onSubmit={handleSubmit(onSubmit)}>

                <input type="text" {...register("name")} />

                <input type="email" {...register("email", { required:
true })} />

                {errors.email && <span style={{ color: "red" }}>

                    *Email* is mandatory </span>}

                <input type="password" {...register("password")} />

   <input type={"submit"} style={{ backgroundColor: "#a1eafb" }} />
```

```
            </form>

        </>

    );

}

export default Contact;
```

Even odd usestate program

```
import React from "react";

import { useState } from "react";

function Counter() {

    const [click, setClick] = useState(0);

return (

    <>

        <div>

            <p>You've clicked {click} times!</p>

                <p>The number of times you have clicked

                is {click % 2 == 0 ? 'even!' : 'odd!'}</p>

<button onClick={() => setClick(click => click + 1)}>Click me</button>

        </div>

        </>

    );

}

export default Counter;
```

Update Message

```
import React from 'react';

import { useState } from 'react';

export default function Register() {

  const [message, setMessage] = useState('');

 const [updated, setUpdated] = useState(message);

const handleChange = (e) => {

    setMessage(e.target.value);

  };

const handleClick = () => {

    setUpdated(message);

  };

 return (

    <div>

      <input type="text" name="message" onChange={handleChange}
value={message}/>



    <h2>Message: {message}</h2>

    <h2>Updated: {updated}</h2> <button
onClick={handleClick}>Update</button>

</div>

  );

}
```

`Image upload`

createObjectURL() method

The createObjectURL() method creates a DOMString containing a URL representing the object given in the parameter of the method. The new object URL represents the specified File object.

Syntax:

```
const url = URL.createObjectURL(object);
```

- object: A File, image, or any other MediaSource object for which object URL is to be created.

Return value: A DOMString containing an object URL of that object.

`Contact.jsx`

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

function Contact() {
  const [selectedFile, setSelectedFile] = useState(null);
  const handleFileChange = (event) => {
    const file = event.target.files[0];
    setSelectedFile(file);
  };
  const handleFileUpload = () => {
    if (selectedFile)
    {
      console.log('Selected file:', selectedFile);
    }
    else
    {
      console.log('No file selected.');
```

```
    }
  };

  return (
    <div>
      <h1>Image Upload Example</h1>
      <input type="file" onChange={handleFileChange} />
      <button onClick={handleFileUpload}>Upload Image</button>

      {selectedFile!=null && (
        <div>

          <img
            src={URL.createObjectURL(selectedFile)} style={{ maxWidth:
'100%' }} />
        </div>
      )}
    </div>
  );
}
export default Contact;
```

Using Multiple State Variables

Declaring state variables as a pair of `[something, setSomething]` is also handy because it lets us give *different* names to different state variables if we want to use more than one:

```
function demo() {
```

```
const [age, setAge] = useState(42);

const [fruit, setFruit] = useState('banana');
```

Example

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

import { ToastContainer, toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';

export default function Login(){

const [email , setemail]= useState("");

 const emailchange=(e)=>

{      setemail(e.target.value)

    }

const show = () =>

    {

          if(email=="")

          {

               alert("Please fill your Email");

          }

          else  if (email=="varsha.shringi04@gmail.com")

          {

               alert("Hello Varsha");          }

            else{

                toast.error("pls check again")

            }

    }
```

```
return(

<>

<ToastContainer/>

<form>

      <input type= "email" placeholder="Enter your email"
onChange={emailchange} /><br/>

      <button onClick={show}>Click Here</button>

</form>

</>

    )

}

 export {Login}
```

Button Functionality (Upper ,lower ,Remove, CLR ALL)

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

import Button from 'react-bootstrap/Button';



const Textform = () => {

  const[text,settext]=useState("Enter text here");



 const handleupclick=()=>

  {

    //console.log("uppercaser here");

    let newtext= text.toUpperCase();
```

```
    settext(newtext)
  }
  const handlelwclick=()=>
  {
    let newtext= text.toLowerCase();
    settext(newtext)
  }
  const handlegap=()=>
  {
let newtext= text.replace(/\s+/g, '').trim();
    settext(newtext)
 }
  const handleclr=()=>
  {
    let newtext= "";
    settext(newtext)
  }
  const upperonchange=(e)=>
  {
    settext(e.target.value);
  }

return (
    <>
```

```
      <h1>Analyze Example </h1>

      <div className="mb-3">

        <label htmlFor="mytext" className="form-label">Enter your
text</label>

        <textarea className="form-control" id="mytext"
onChange={upperonchange} value={text} rows="8"></textarea>

        <br/>

        <Button variant="primary mx-2"
onClick={handleupclick}>UpperCase</Button>

        <Button variant="primary mx-2"
onClick={handlelwclick}>LowerCase</Button>

        <Button variant="primary mx-2"  onClick={handlegap}>Remove
space</Button>

        <Button variant="primary mx-2"
onClick={handleclr}>Clear</Button>



    </div>

       <h2>Your text summary</h2>

       <p>{text.length}</p>

</>

  )

}

export default Textform ;
```

props

Props stand for "Properties." They are read-only components. It is an object which stores the value of attributes of a tag and work similar to the HTML attributes. It gives a way to pass data from one component to other

components. It is similar to function arguments. Props are passed to the component in the same way as arguments passed in a function.

Props are immutable so we cannot modify the props from inside the component. Inside the components, we can add attributes called props. These attributes are available in the component as this.props and can be used to render dynamic data in our render method.

Table in react

```
import React from 'react';

import ReactTable from "react-table";

function Tabledata() {

return (

<div className="App">

    <table>

        <tr>

            <th>Name</th>

            <th>Age</th>

            <th>Gender</th>

        </tr>

        <tr>

            <td>Anom</td>

            <td>19</td>

            <td>Male</td>

        </tr>

        <tr>
```

```
            <td>Megha</td>
            <td>19</td>
            <td>Female</td>
        </tr>
        <tr>
            <td>Subham</td>
            <td>25</td>
            <td>Male</td>
        </tr>
    </table>
</div>
);
}


export default Tabledata;
```

React-Toastify

install

```
npm i react-toastify
```

import link

```
import { ToastContainer, toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';
```

Toast Notifications are popup messages that are added so as to display a message to a user. It can be a success message, warning message, or custom message. Toast Notification is also called Toastify Notifications.

```
import React from 'react';
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';

function Showname(){
    const notify = () => toast("Wow so easy!");

    return (
        <>
      <div>
        <button onClick={notify}>Notify!</button>
        <ToastContainer />
      </div>
      </>
    );
  }
  export default Showname;
```

toast example
Showname.jsx

```
import React from 'react';
import {ToastContainer,toast} from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';

function Showname(){
    const notify = ()=>{

        toast.warning('Danger')

        toast.success('successful')

        toast.info('Independence Day')
```

```
        toast.error('Runtime error')

        toast('Hello students')

    }
    return (
        <div className="GeeksforGeeks">
            <button onClick={notify}>Click Me!</button>
            <ToastContainer />
            </div>
    );
}
export default Showname
```

Hooks

useEffect()
useRef()
useContext()
useState()
useNavigate()
useLocation()
useMemo()
useReducer()
useCallback()
custom hooks()

`useEffect` Hooks

The `useEffect` Hook allows you to perform side effects in your components.

side effects are: fetching data, directly updating the DOM ,manually changing the DOM in React components.

The useEffect hook in React is use to handle the side effects in React such as fetching data, and updating DOM. This hook runs on every render but there is also a way of using a dependency array using which we can control the effect of rendering .

Syntax

`useEffect(()=>{},[]);`

()=>{} – Function passed to this hook

[ ] – It tells the hook when to re-render the component.

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

`useEffect` accepts two arguments. The second argument is optional.

without dependency

```
useEffect(<function>, <dependency>)
useEffect(() => {
  //Runs on every render
});
```

with dependency

```
useEffect(()->{
   // Code
}, [] )
```

```
import React, { useState, useEffect } from 'react';
function Counteffect() {
  const [count, setCount] = useState(0);
useEffect(() => {
    setTimeout(() => {
      setCount((count) => count + 1);
    }, 1000);
  });
 return (
```

```
    <div>
      <p>You clicked {count} times</p>
      {/* <button onClick={() => setCount(count + 1)}>
        Click me
      </button> */}
      <button>click me</button>
    </div>
  );
}
export default Counteffect
```

Useeffect Example

```
import { useState, useEffect } from "react";
import ReactDOM from "react-dom/client";
function Counteffect() {
  const [count, setCount] = useState(0);
  const [calculation, setCalculation] = useState(0);
 useEffect(() => {
    setCalculation(() => count * 2);
  }, [count]);
  return (
    <>
      <p>Count: {count}</p>
      <button onClick={() => setCount((c) => c + 1)}>+</button>
```

```
      <p>Calculation: {calculation}</p>
    </>
  );
}
export default Counteffect
```

The dollar ($) sign is a JavaScript identifier, which simply means that it identifies an object in the same way that a name or variable does.

useEffect Example

```
import { useState, useEffect } from "react";
const Counteffect = () => {
  const [count, setCount] = useState(0);
   useEffect(() => {
      document.title = `You clicked ${count} times`;
  }, [count]);
  return (
    <>
      <div>
          <button onClick={() => setCount((prevCount) => prevCount +
1)}>
              Click {count} times{" "}
          </button>
      </div>
```

```
</>

  )

}

export default Counteffect
```

usecontext

React Context is a way to manage state globally.

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

Context provides a way to pass data or state through the component tree without having to pass props down manually through each nested component.

Benefit

React useContext hook allows sharing data across a React component tree without passing props down through multiple levels.

Create Context

```
const UserContext = createContext()
```

Context Provider

Wrap child components in the Context Provider and supply the state value.

import

```
import { useState, createContext, useContext } from "react";
```

```jsx
import { useState, createContext, useContext } from "react";

const UserContext = createContext();

function Component1() {
  const [user, setUser] = useState("Jesse Hall");

  return (
    <UserContext.Provider value={user}>
      <h1>{`Hello ${user}!`}</h1>
      <Component2 />
    </UserContext.Provider>
  );
}


function Component2() {
  return (
    <>
      <h1>Component 2</h1>
      <Component3 />
    </>
  );
}
function Component3() {
  return (
    <>
      <h1>Component 3</h1>
```

```
      <Component4 />
    </>
  );
}

function Component4() {
  return (
    <>
      <h1>Component 4</h1>
      <Component5 />
    </>
  );
}

function Component5() {
  const user = useContext(UserContext);

  return (
    <>
      <h1>Component 5</h1>
      <h2>{`Hello ${user} again!`}</h2>
    </>
  );
}
```

```
export default Component1;
```

note:
redux is same as Context API

parent to child data access.

context API build in react . bt redux is external library .

Parent.jsx

```
import React, { createContext,useState } from 'react';
import Child from './Child';
import Superchild from './Superchild';

export const GlobalInfo = createContext();

function Parent() {
  const[color,setcolor]= useState('green');
  return (
    <>
    <GlobalInfo.Provider value={{parentColor:color}}>
          <div>
             <h1>AppParent Component </h1>
             <Child />
             <Superchild/>
          </div>
    </GlobalInfo.Provider>
    </>
  )
}
export default Parent;
```

Child.jsx

```
import React,{useContext} from 'react';
import { GlobalInfo } from './Parent';
function Child() {
```

```
    const {parentColor} = useContext(GlobalInfo)
  return (
    <>
    <div>
      <h1 style={{color:parentColor}}>hellllo Child Component</h1>
    </div>
    </>

  )
}

export default Child;
```

Superchild.jsx

```
import React,{useContext} from 'react';
import { GlobalInfo } from './Parent';
function Superchild() {
    const {parentColor} = useContext(GlobalInfo)
  return (
    <>
    <div>
      <h1 style={{color:parentColor}}>hellllo SuperChild Component</h1>
    </div>
    </>

  )
}

export default Superchild;
```

useMemo

(unwanted stop function calling)

time decrease + hang Down performance issue .

React `useMemo` Hook returns a memoized value.

useMemo is a hook used in the functional component of react that returns a memoized value.The useMemo hook is used to improve performance in our React application.

`useMemo` is a React Hook that lets you cache the result of a calculation between re-renders.

```
const cachedValue = useMemo(calculateValue, dependencies)
```

useMemo

```
import React, { useState, useMemo } from "react";

const Exmemo = () => {
  const [count, setCount] = useState(0);
  const [item, setItem] = useState(10);

   function multiCount()
   {
       console.warn("multiCount");
       return count*5;
   }

//    const multiCountMemo= useMemo(function multiCount()
//    {
//        console.warn("multiCount");
//        return count*5;
// },[count])

  return (
    <>
      <div>
        <h1>UseMemo hook used with example</h1>
        <h2>Count :{count}</h2>
        <h2>Item :{item}</h2>
```

```
        <h2>{multiCount()}</h2>
        {/* <h2>{multiCountMemo}</h2>   */}

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

        <button onClick={() => setItem(item * 10)}> Update Item
</button>
      </div>
    </>
  );
};

export default Exmemo;
```

useRef Hook

The useRef is a hook that allows to directly create a reference to the DOM element in the functional component.The useRef hook is a new addition in React 16.8.

`useRef` is a React Hook that reference a value that's not needed for rendering.

syntax

```
const ref = useRef(initialValue)
```

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

The useRef Hook allows you to persist values between renders. It can be used to store a mutable value that does not cause a re-render when updated. It can be used to access a DOM element directly.

for ex : focus , color, value handle using useRef .

`useRef()` only returns one item. It returns an Object called `current`.

When we initialize `useRef` we set the initial value: `useRef(0)`.

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

```
// function Exref() {
//     const inputRef= useRef(null);
//     const onButtonClick = () => {
//       inputRef.current.focus();
//     };
//     return (
//       <>
//         <input ref={inputRef} type="text" />
//         <button onClick={onButtonClick}>Submit</button>
//       </>
//     );
//   }
// export default Exref


// import React, {useRef} from 'react';

// function Exref() {

// const focusPoint = useRef(null);
// const onClickHandler = () => {
//  focusPoint.current.value ="Rajasthan is a beautiful state";
//  focusPoint.current.focus();
// };
// return (
//  <>
//  <div>
//      <button onClick={onClickHandler}>ACTION</button>
//  </div>

//  <br/><br/>
//  <textarea ref={focusPoint}/>
//  </>
// );
// };

// export default Exref;

// import React, { useEffect, useState } from "react";
// import { useRef } from "react";
// function Exref(){
```

```
//     const[mydata,setmydata]=useState("");
//     const[count,setcount]=useState(0);
//     //const[count,setcount]=useState();

//      const mydata = useRef(10);

//    useEffect(()=>{
//         setcount(count+1)
//        count.current = count.current + 1;
//    });
//    },[]);
//     const chnge=(e)=>{
//           setmydata (e.target.value);
//     }
//     return(
//         <>
//         <input type="text" value={mydata} onChange={chnge}/>
//         <h3>Total number of count render is {count.current}</h3>


//         </>
//     )
//  }
//  export default Exref;

// import React, { useEffect, useState } from "react";
// import { useRef } from "react";
// function Exref(){

//     const[mydata,setmydata]=useState("");

//      const inputElem = useRef("");

//      const changeStyle=()=>{
//         inputElem.current.style.backgroundColor ="red";
//         inputElem.current.focus();
//      }

//     const chnge=(e)=>{
//           setmydata (e.target.value);
//     }
//     return(
```

```
//          <>
//          <input ref={inputElem} type="text" value={mydata}
onChange={chnge}/>
//          <br/>
//          <button onClick={changeStyle}>Submit</button>



//          </>
//      )
//   }
//   export default Exref;

// import React from "react";
// import { useRef , useEffect} from "react";

// const Exref = () => {
//   const inputRef = useRef();

//   useEffect(() => {

//   console.log(inputRef.current);
//   inputRef.current.focus();
//     }, []);

//     // / Logs `undefined` during initial rendering
//    console.log(inputRef.current);

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

// export default Exref

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

// const Exref = () => {
//   const timerIdRef = useRef(0);
```

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

//  const startHandler = () => {
//    if (timerIdRef.current) { return; }
//    timerIdRef.current = setInterval(() => setCount(c => c+1), 1000);
//  };

//  const stopHandler = () => {
//    clearInterval(timerIdRef.current);
//    timerIdRef.current = 0;
//  };

//  useEffect(() => {
//    return () => clearInterval(timerIdRef.current);
//  }, []);
//   return (
//  <>
//   <div>
//       <h1>Timer: {count}s</h1>
//       <div>
//         <button onClick={startHandler}>Start</button>
//         <button onClick={stopHandler}>Stop</button>
//       </div>
//     </div>

//  </>  )
// }

// export default Exref


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

const Exref = () => {
    const countRef = useRef(0);

  const handle = () => {
    countRef.current++;
    console.log(`Clicked ${countRef.current} times`);
  };
```

```
  console.log('I rendered!');

  return (
    <div>
        <button onClick={handle}>Click me</button>;

    </div>
  )
}

export default Exref
```

```
import { useCallback, useState } from "react";
import React from "react";
import ChildBck from "./ChildBck";
```

useCallback Hook

The useCallback hook is used when you have a component in which the child is rerendering again and again without need.

The `useCallback` and `useMemo` Hooks are similar. The main difference is that `useMemo` returns a memoized *value* and `useCallback` returns a memoized *function.*

use `useCallback` is to prevent a component from re-rendering unless its props have changed.

`const cachedFn = useCallback(fn, dependencies)`

```
function Back()
{
const[add,setadd]=useState(0);
const[count,setcount]=useState(0);

const Learning = useCallback( ()=>{

},[count])

// Const Learning =()=>{
```

```
// }
    return(
        <>
        <h1>Learning Usecallbck</h1>
        <ChildBck Learning={Learning} count={count}/>
        <h1>{add}</h1>
<button onClick={()=>setadd(add+1)}>Addition </button>
        <h1>{count}</h1>
        <button onClick={()=>setcount(setcount+2)}>Count</button>

        </>
    )
}
 export default Back;
```

```
without callback ()

import React, { useState} from 'react'
const funccount = new Set();
const Back= () => {
const [count, setCount] = useState(0)
const [number, setNumber] = useState(0)
 const incrementCounter = () => { setCount(count + 1)
// }
const decrementCounter = () => {
//  setCount(count - 1)
// }

// const incrementNumber = () => {
//  setNumber(number + 1)
// }

// funccount.add(incrementCounter);
// funccount.add(decrementCounter);
// funccount.add(incrementNumber);
// alert(funccount.size);

// return (
//     <>
//     <div>Count: {count}
```

```
//  <button onClick={incrementCounter}>Increase counter</button>
//  <button onClick={decrementCounter}>Decrease Counter</button>
//  <button onClick={incrementNumber}>increase number</button>
//  </div>
//     </>

// )
// }


// export default Back;
```

Without useCallback Hook: *The problem is that once the counter is updated, all three functions are recreated again. The alert increases by three at a time .*
Without useCallback Hook: *The problem is that once the counter is updated, all three functions are recreated again. The alert increases by three at a time but if we update some states all the functions related to that states should only re-instantiated.*

```
// with callback()
import React, { useState, useCallback } from 'react'
var funccount = new Set();
const Back = () => {


const [count, setCount] = useState(0)
const [number, setNumber] = useState(0)

const incrementCounter = useCallback(() => {
setCount(count + 1)
}, [count])

const decrementCounter = useCallback(() => {
setCount(count - 1)
}, [count])

const incrementNumber = useCallback(() => {
```

```
setNumber(number + 1)
}, [number])

funccount.add(incrementCounter);
funccount.add(decrementCounter);
funccount.add(incrementNumber);
alert(funccount.size);

return (
    <>
    <div>Count: {count}
    <br/><br/>
    <button onClick={incrementCounter}>Increase counter</button>
    <button onClick={decrementCounter}>Decrease Counter</button>
    <button onClick={incrementNumber}>increase number</button>
    </div>
    </>
)
}
export default Back;
```

when we change the state 'count' then two functions will re-instantiated so the set size will increase by 2 and when we update the state 'number' then only one function will re-instantiated and the size of the set will increase by only one.

uesReducer Hook

`useReducer` Hook is similar to the `useState` Hook.

It allows for custom state logic.

alternative of usestate hook ().

Complex state management logic.

reducer is function accept two two parameter

newstate = reducer(currentState , action).

*The **useReducer** Hook is the better alternative to the **useState** 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*

useReducer(<reducer>, <initialState>)

The `reducer` function contains your custom state logic and the `initialState`can be a simple value but generally will contain an object.

The `useReducer` Hook returns the current `state`and a `dispatch`method.

The `useReducer` method gives you a state variable and a `dispatch` method to make state changes. You can define state in the following way:

```
const [state, dispatch] = useReducer(reducerMethod, initialValue)
```

```
import reducer
```

```
import { useReducer } from 'react'
```

The `dispatch` function returned by `useReducer` update the state to a different value and trigger a re-render.

The dispatch function accepts an object that represents the type of action we want to execute when it is called.

*The `dispatch` function **only updates the state variable for the next render**.*

Syntax

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

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


// const initialState=0;
// const reducer =(state , action)=>{
//          switch(action)
//          {
//              case "Increment" :
//                  return state+1
//              case "Decrement" :
//                  return state-1
//                  default:
//                      return state
//          }

// }
//  function Recounter()
//  {
//     const[count ,dispatch] = useReducer(reducer , initialState)
//     useReducer(reducer , initialState)
//     return(
//         <>
//         <div>
//             <h1>count={count}</h1>
//             <button onClick={()=>dispatch("Increment")}>
Increment</button>
//             <button
onClick={()=>dispatch("Decrement")}>Decrement</button>


//         </div>
```

```
//         </>
//     )

//  }
//  export default Recounter;

// const initialState = {count: 0};

// function reducer(state, action) {
//   switch (action.type) {
//     case 'increment':
//       return {count: state.count + 1};
//     case 'decrement':
//       return {count: state.count - 1};
//     default:
//       throw new Error();
//   }
// }

// function Recounter() {
//   const [state, dispatch] = useReducer(reducer, initialState);
//   return (
//     <>
//       Count: {state.count}
//       <br/>
//       <button onClick={() => dispatch({type:
'decrement'})}>Decrement</button>
//       <button onClick={() => dispatch({type:
'increment'})}>Increment</button>
```

```
//     </>
//   );
// }
//  export default Recounter



// 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 Recounter = () => {
    // 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 Recounter;
```

useNavigte hook()

useNavigation is a Hook available in React Router.

The **useNavigate() hook** is introduced in the React Router v6 to replace the useHistory() hook.

`useNavigation` is a hook which gives access to `navigation` object

useLocation Hook()

The useLocation Hook allows you to access the location object that represents the active URL. The value of the location object changes whenever the user navigates to a new URL. The useLocation Hook can be convenient when you have to trigger any event whenever the URL changes.

pages:

(services , tabledata, counter,demonav)

Services.jsx

```
import React, { useEffect, useState } from "react";
import {useNavigate} from "react-router-dom";
 function Services(){
   const newnav = useNavigate();
    const[name,setname]=useState("");
```

```
    const[text,settext]=useState("");
    const click=()=>{
        //settext("hello");
        newnav("/Counter",{state:{text:text}});
    }
    const chnge=(e)=>{
        setname(e.target.value);
    }
    useEffect(()=>{
     settext(name);
    })
    return(
        <>
        <label>Name:</label>
        <input value={name} onChange={chnge} placeholder="Enter Your
Name"/>
        <button onClick={click}> Click Here</button>
        </>
    )
}
 export default Services
```

Tabledata.jsx

```
import React from 'react';
import ReactTable from "react-table";
import {useNavigate} from "react-router-dom"
function Tabledata() {
    const navigate = useNavigate();
return (
```

```
<div className="App">
<h1 style={{color:"green"}}>Rajasthan State is very beautiful</h1>
     <button onClick={()=>navigate(-1)}>Go Demonav</button>
    <table>
        <tr>
            <th>Name</th>
            <th>Age</th>
            <th>Gender</th>
        </tr>
        <tr>
            <td>Anom</td>
            <td>19</td>
            <td>Male</td>
        </tr>
        <tr>
            <td>Megha</td>
            <td>19</td>
            <td>Female</td>
        </tr>
        <tr>
            <td>Subham</td>
            <td>25</td>
            <td>Male</td>
        </tr>
    </table>
</div>
```

```
);
}

export default Tabledata;
```

Demonav.jsx

```
import React from 'react';
import {useNavigate} from "react-router-dom"


function Demonav() {
    const navigate = useNavigate();
  return (
    <>
    <div>
    <h1 style={{color:"green"}}>Demo for useNvigte Hook</h1>
        <button onClick={()=>navigate("/Tabledata")}>Click me</button>
    </div>
    </>
  )
}
export default Demonav;
```

Counter.jsx

```
import React from "react";
import { useState } from "react";
import { useLocation, useNavigate } from "react-router-dom";
function Counter() {
    const loac= useLocation();
 const [click, setClick] = useState(0);
```

```
return (
    <>
        <div>
        <h1>Your Aim Is:{loac.state.text}</h1>
            <p>You've clicked {click} times!</p>
                <p>The number of times you have clicked
                is {click % 2 == 0 ? 'even!' : 'odd!'}</p>

 <button onClick={() => setClick(click => click + 1)}>Click me</button>
        </div>
        </>
    );
}
export default Counter;
```

Login page validation

install mdb-react-ui-kit

```
npm i mdb-react-ui-kit
```

```
import React, { forwardRef } from 'react';
import { useState } from 'react';
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
import Button from 'react-bootstrap/Button';
import {
  MDBBtn,
  MDBContainer,
  MDBRow,
  MDBCol,
```

```
  MDBCard,
  MDBCardBody,
  MDBInput,
  MDBIcon,
  MDBCheckbox
}
from 'mdb-react-ui-kit';
import { Link, useNavigate } from 'react-router-dom';
function Login() {
  const navigatee=useNavigate();
  const [name,setname] = useState("");
  const [pass,setpass]= useState("");
  const [email , setemail]= useState("");
  const [number,setnumber] = useState("");
  const namechange=(e)=>
    {
        setname(e.target.value)
    }
    const passchange=(e)=>
    {
        setpass(e.target.value)
    }
    const numberchange=(e)=>
    {
        setnumber(e.target.value)
    }
  const emailchange=(e)=>
    {
```

```
        setemail(e.target.value)
    }
  const navigate = useNavigate()
  const Forward = () => navigate(`signup`)
   const Show = () =>
     {
       if (name=="")
          {
             alert("please fill name");
          }
          else if(!isNaN==name)
          {
           alert("please fill only alphabat fields");
          }
          else if(name.length<5 || name.length>15)
          {
           alert("name length will between 5 to 15");
          }
          else if (pass=="")
          {
             alert("please fill password");
          }
          else if(pass.length<=5 || pass.length>10)
          {
           alert("password length will between 5 to 10");
          }
          else if(email=="")
         {
```

```
                alert("Please fill your Email");
        }
        else if(number=="")
        {
                alert("please fill your mobile number");
         }
        else if(number.length !=10)
        {
                alert("number must be 10 digits");
        }
        else
        {
          toast.success("congratulations.........");
          setTimeout(() => {
         navigatee("/Textform")
          }, 3000);
        }
 return(
  <>
  <ToastContainer/>
    <MDBContainer fluid>


      <MDBRow className='d-flex justify-content-center
align-items-center h-100'>
        <MDBCol col='12'>


          <MDBCard className='bg-white my-5 mx-auto'
style={{borderRadius: '1rem', maxWidth: '500px'}}>
            <MDBCardBody className='p-5 w-100 d-flex flex-column'>
```

```
                <h2 className="fw-bold mb-2 text-center">Login Page</h2>
                <p className="text-white-50 mb-3">Please enter your login
and password!</p>
<MDBInput wrapperClass='mb-4 w-100' label='Name'
id='formControlLg'onChange={namechange} type='text' size="lg"/>
                <MDBInput wrapperClass='mb-4 w-100' label='Password'
id='formControlLg' onChange={passchange} type='password' size="lg"/>
                <MDBInput wrapperClass='mb-4 w-100' label='Email address'
id='formControlLg'onChange={emailchange} type='email' size="lg"/>
                <MDBInput wrapperClass='mb-4 w-100' label='Number'
id='formControlLg' onChange={numberchange} type='number' size="lg"/>
                <MDBCheckbox name='flexCheck' id='flexCheckDefault'
className='mb-4' label='Remember password' />

                <MDBBtn size='lg' onClick={Show} >
                  Login
                </MDBBtn>
<span class="forget">
      <a href="forgotten Password">Forgotten Password</a>
</span>
<hr/>
<Button type="button" class="btn btn-primary"
onClick={Forward}>Signup</Button>
 <hr className="my-4" />
</MDBCardBody>
          </MDBCard>
 </MDBCol>
      </MDBRow>
</MDBContainer>
    </>
```

```
  );
}

export default Login;
```

props:

React allows us to pass information to a Component using something called props (which stands for properties). Props are objects which can be used inside a component.

They are read-only components. It is an object which stores the value of attributes of a tag and work similar to the HTML attributes. It gives a way to pass data from one component to other components. It is similar to function arguments. Props are passed to the component in the same way as arguments passed in a function.

Props are immutable so we cannot modify the props from inside the component. Inside the components, we can add attributes called props.

parse() function

The parse() function takes the argument of the JSON source and converts it to the JSON format.

because most of the time when you fetch the data from the server the format of the response is the string

API

API stands for Application Programming Interface. It is a medium that allows different applications to communicate programmatically with one another and return a response.

React utilizes API calls to connect with external services to receive or send data. They allow React application to interact with other systems and exchange information with them.

Fetch *allows you to send or get data across networks.*

The fetch() method in JavaScript is used to request data from a server. The request can be of any type of API that returns the data in JSON or XML. The fetch() method requires one parameter, the URL to request, and returns a promise .

syntax

```
fetch('url')          //api for the get request
```

```
.then(response => response.json())
.then(data => console.log(data));
```

Return Value: *It returns a promise whether it is resolved or not. The return data can be of the format JSON or XML*

then() *method in JavaScript has been defined in the Promise API and is used to deal with asynchronous tasks such as an API call.*

. then() in javascript takes two parameters, ***the callback function for the success and the callback function for failure cases of the Promise****. .Then() in JavaScript returns the callback function based on whether the Promise is fulfilled or rejected.*

async await

The await keyword is used inside the async function to wait for the asynchronous operation.

Async functions always return a promise.

The keyword 'async' before a function makes the function return a promise, always. And the keyword await is used inside async functions, which makes the program wait until the Promise resolves.

api url

fakestoreapi.com/products

```
https://jsonplaceholder.typicode.com/users
```

Apirender.jsx

```
import Button from 'react-bootstrap/Button';
import { useState, useEffect } from 'react';
import { useCart } from 'react-use-cart';
import { Card } from 'react-bootstrap';
import Dcrd from './Dcrd';
function Apirender() {
  const { addItem } = useCart();
```

```
//    const[Datacard,setcard]=useState([]);


//    const getapi=async()=>{
//    const result = await fetch("https://fakestoreapi.com/products");
//    // console.log(result);
//    setcard(await result.json());


// }
// useEffect(()=>{
//    getapi();
// },[])
  return (
     <div className='flex-card'>
       {
         Dcrd.map((items=>{
           return(
             <>
               <div className='cardcard' key={items.id}>
               <Card>
       <Card.Img  variant="top" src={items.image} width="100" />
       <Card.Body>
         <Card.Title>{items.title}</Card.Title>
         <Card.Text>
         {items.text}
         </Card.Text>
         <h5>{items.price}</h5>
         <Button  onClick={() => addItem(items)} variant="primary">ADD
TO CART</Button>
       </Card.Body>
```

```
    </Card>
                </div>
                </>
                )
            }))
        }
    </div>
  );
}
export default Apirender;
```

Dcrd.jsx

```
const Dcrd=[
    {
        id:1,
        // img:`${myimage}`,
        url:"/Card",
         title:"Nature Image",
         // update:"Last updated 3 mins ago",
        // text:"This card has supporting text below as a natural
lead",
    },
    {
        id:2,
        url:"/Card",
        // img:`${myimage}`,
         title:"Water Image",


        // update:"Last updated 3 mins ago",
```

```
        // img:"img1.jpg",
        // text:"This card has supporting text below as a natural
lead",
    },
    {
        id:3,

        url:"/Card",
        title:"Mountain Image",

        // img:"img1.jpg",
        update:"Last updated 3 mins ago",
        text:"This card has supporting text below as a natural lead",
    },
    {
        id:4,
        img:"",
        title:"Mountain Image",
        url:"/Card",
        // img:"img1.jpg",
        update:"Last updated 3 mins ago",
        text:"This card has supporting text below as a natural lead",
    },
    {
        id:5,
        img:"",
        title:"Mountain Image",
        url:"/Card",
        // img:"img1.jpg",
```

```
        update:"Last updated 3 mins ago",
        text:"This card has supporting text below as a natural lead",
    },
    {
        id:6,
        img:"",
        title:"Mountain Image",
        url:"/Card",
        // img:"img1.jpg",
        update:"Last updated 3 mins ago",
        text:"This card has supporting text below as a natural lead",
    },
    {
        id:7,
        img:"",
        title:"Mountain Image",
        url:"/Card",
        // img:"img1.jpg",
        update:"Last updated 3 mins ago",
        text:"This card has supporting text below as a natural lead",
    },
    {
        id:8,
        img:"",
        title:"Mountain Image",
        url:"/Card",
        img:"img1.jpg",
        update:"Last updated 3 mins ago",
```

```
        text:"This card has supporting text below as a natural lead",
    },
    {
        id:9,
        img:"",
        title:"Mountain Image",
        url:"/Card",
        img:"img1.jpg",
        update:"Last updated 3 mins ago",
        text:"This card has supporting text below as a natural lead",
    }

]
export default Dcrd;
```

```
// import Button from 'react-bootstrap/Button';
// import { useState, useEffect } from 'react';
// import { useCart } from 'react-use-cart';
// import { Card } from 'react-bootstrap';
// import Dcrd from './Dcrd';



// function Apirender() {
//   const { addItem } = useCart();



//   const[Datacard,setcard]=useState([]);
```

```
//   const getapi=async()=>{


//   const result = await fetch("https://fakestoreapi.com/products");
//   // console.log(result);
//     setcard(await result.json());


// }
// useEffect(()=>{
//   getapi();
// },[])
//   return (
//     <div className='flex-card'>
//       {
//         Dcrd.map((items=>{
//           return(
//             <>
//             <div className='cardcard' key={items.id}>
//             <Card>
//       <Card.Img  variant="top" src={items.img} width="100" />
//       <Card.Body>
//         <Card.Title>{items.title}</Card.Title>
//         <Card.Text>
//         {items.text}
//         </Card.Text>
//         <h5>{items.price}</h5>
//         <Button  onClick={() => addItem(items)}
variant="primary">ADD TO CART</Button>
//       </Card.Body>
//     </Card>
```

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

// export default Apirender;


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

// const Apirender = () => {
//   const [users, setUsers] = useState([])

//   const fetchUserData = () => {
//     fetch("https://jsonplaceholder.typicode.com/users")
//       .then(response => {
//         return response.json()
//       })
//       .then(data => {
//         setUsers(data)
//       })
```

```
//   }


//   useEffect(() => {
//     fetchUserData()
//   }, [])


//   return (
//     <div>
//       {users.length > 0 && (
//         <ul>
//           {users.map(user => (
//             <h1><li key={user.id}>{user.name}</li></h1>
//           ))}
//           {users.map(user => (
//             <li key={user.id}>{user.username}</li>
//           ))}
//         </ul>
//       )}
//     </div>
//   );
// }


// export default Apirender;




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

```
function Apirender() {
  const url = "https://jsonplaceholder.typicode.com/users";
  const [data, setData] = useState([]);


  const fetchInfo = () => {
    return fetch(url)
      .then((res) => res.json())
      .then((d) => setData(d))
  }



  useEffect(() => {
    fetchInfo();
  }, []);


  return (
    <div className="App">
      <h1 style={{ color: "green" }}> FETCH API</h1>
      <center>
        {data.map((dataObj, index) => {
          return (
            <div
              style={{
                width: "15em",
                backgroundColor: "#35D841",
                padding: 2,
                borderRadius: 10,
                marginBlock: 10,
```

```
                }}
                >
                <p style={{ fontSize: 20, color: 'white'
}}>{dataObj.name}</p>
                </div>
            );
        })}
      </center>
    </div>
  );
}


export default Apirender;
```

Custom Hooks

Hooks are reusable functions.

When you have component logic that needs to be used by multiple components, we can extract that logic to a custom Hook.

Custom React hooks are an essential tool that let you add special, unique functionality to your React applications.

A custom hooks is like java script () start with "use" .

hooks like useState, useEffect are reusable components. Somtimes we make components that we have to reuse again and again in the application. In this case we can convert the component to hooks by extracting logic from it.

When we want to share the logic between other components, we can extract it to a separate function

Custom hooks is to maintain the concept of DRY(Don't Repeat Yourself) in your React apps.

To remove the duplicated logic in component and extract that logic in hooks .

hooks like useState, useEffect are reusable components. Somtimes we make components that we have to reuse again and again in the application. In this case we can convert the component to hooks by extracting logic from it.

There are several advantages to using Custom Hooks:

- **Reusability — we can use the same hook again and again, without the need to write it twice.**
- **Clean Code — extracting all the component logic into a hook will provide a cleaner codebase.**
- **Maintainability — easier to maintain. if we need to change the logic of the hook, we only need to change it once.**

Note: It is important to name your custom hooks starting with "use", because without it React can't realize that it is a custom hook and therefore can't apply the rules of hooks to it. So, you should name it starting with "use".

Cnthk.jsx

```
import React from 'react';
```

```jsx
import { useState , useEffect } from 'react';
import useTitle from './useTitle';
function Cnthk() {
const[counter,setcounter]=useState(0)
useTitle(counter)
// useEffect(()=>{
//     document.title=counter+"hits on button"
// })


  return (
    <>
    <div>
    <h1>{counter}</h1>
      <button onClick={()=>setcounter(counter+1)}>Click Me</button>
          </div>
      </>
    )
}
export default Cnthk;
```

useTitle.jsx (custom hooks)

```jsx
import {useEffect} from 'react'
function useTitle(counter) {
    useEffect(()=>{
        document.title=counter+"hits on button"
    })
  }
export default useTitle
```

created a new file called `useFetch.js` containing a function called `useFetch` which contains all of the logic needed to fetch our data.

We removed the hard-coded URL and replaced it with a `url` variable that can be passed to the custom Hook.

custom hooks URL

[15 Useful React Custom Hooks That You Can Use In Any Project - DEV Community]

[How to Build Your Own React Hooks: A Step-by-Step Guide (freecodecamp.org)]

Controlled Component

A controlled component is bound to a value, and its changes will be handled in code by using event-based callbacks. Here, the input form element is handled by the react itself rather than the DOM. In this, the mutable state is kept in the state property and will be updated only with setState() method.

Controlled components have functions that govern the data passing into them on every onChange event occurs. This data is then saved to state and updated with setState() method. It makes component have better control over the form elements and data.

Uncontrolled Component

It is similar to the traditional HTML form inputs. Here, the form data is handled by the DOM itself. It maintains their own state and will be updated when the input value changes. To write an uncontrolled component, there is no need to write an event handler for every state update, and you can use a ref to access the value of the form from the DOM.

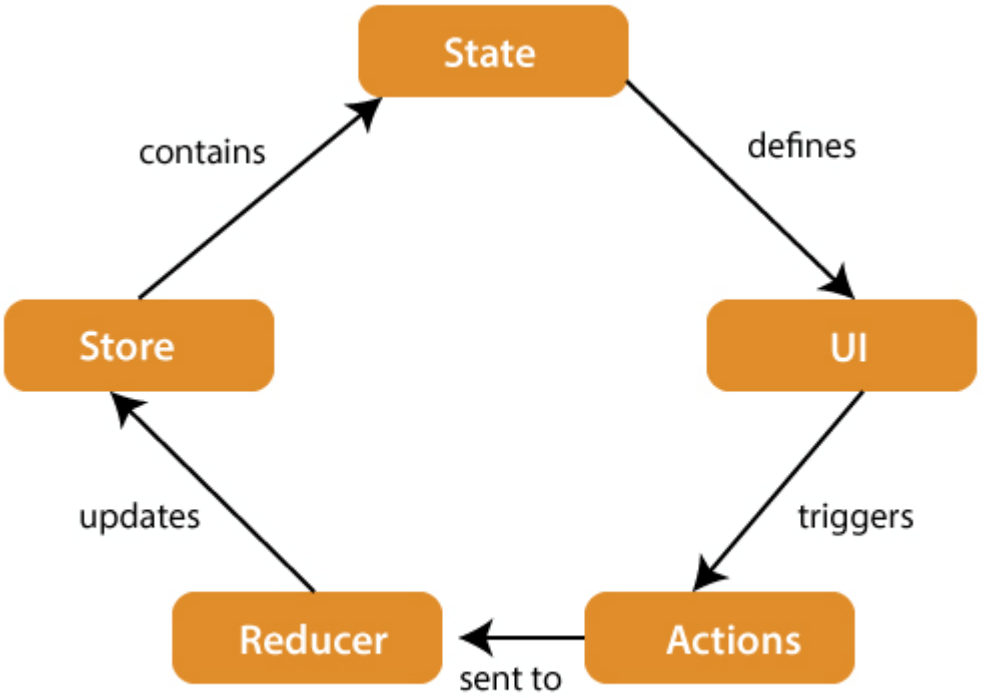
no	controlled	uncontrolled
1.	It does not maintain its internal state.	It maintains its internal states.
2.	Here, data is controlled by the parent component.	Here, data is controlled by the DOM itself.
3.	It accepts its current value as a prop.	It uses a ref for their current values.
4.	It allows validation control.	It does not allow validation control.
5.	It has better control over the form elements and data.	It has limited control over the form elements and data.

React Redux

Redux is an open-source JavaScript library used to manage application state. React uses Redux for building the user interface. It was first introduced by Dan Abramov and Andrew Clark in 2015.

React Redux is the official React binding for Redux. It allows React components to read data from a Redux Store, and dispatch Actions to the Store to update data. Redux helps apps to scale by providing a sensible way to manage state through a unidirectional data flow model. React Redux is conceptually simple. It subscribes to the Redux store, checks to see if the data which your component wants have changed, and re-renders your component.

React Redux is the official UI bindings for React applications that provides a predictable and centralized way to manage the state of your application.



***STORE**: A Store is a place where the entire state of your application lists. It manages the status of the application and has a dispatch(action) function. It is like a brain responsible for all moving parts in Redux.*

***ACTION**: Action is sent or dispatched from the view which are payloads that can be read by Reducers. It is a pure object created to store the information of the user's event. It includes information such as type of action, time of occurrence, location of occurrence, its coordinates, and which state it aims to change.*

***REDUCER**: Reducer read the payloads from the actions and then updates the store via the state accordingly. It is a pure function to return a new state from the initial state.*

create new project

npx create-react-app learn

cd learn

npm start

install redux

npm install redux react-redux

preventDefault()

The preventDefault() method is used to prevent the browser from executing the default action of the selected element. It can prevent the user from processing the request by clicking the link.

Syntax:

```
event.preventDefault()
```

The preventDefault() method ***cancel the event if it is cancelable,*** *meaning that the default action that belongs to the event will not occur.*

preventDefault() prevents the default browser behavior for a given element.

dispatch()

dispatch is a function of the Redux store. You call store. dispatch to dispatch an action. This is the only way to trigger a state change. With React Redux, your components never access the store directly - connect does it for you.

Dispatching actions in Redux is ***the fundamental method of updating a Redux store's state*** *. Actions are used to store relevant information for the state , and they reach the store through the dispatch() method available on the store object.*

state()

A state in Redux is ***a JavaScript object, where the internal state of the application is stored.***

redux project

npx create -react-app todo-react-redux

cd todo-react-redux

npm start

install external library // heading

npm install bootstrap react bootstrap redux react-redux

bootstrap then cdn CSS link copy

index.js

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

src —--- component (folder) – 3 files created

AddTodo.jsx

DisplayTodos.jsx

DisplayCount.jsx

then rafc in pages

call in app.js file

open AddTodo.jsx

create form using title and description fields

```
<Container>
        <Row>
            <Col md={12}>
                <Card className='shadow-sm'>
                    <Card.Body>
                        <h3>Add todo here</h3>
                        <DisplayCount />
                        <Form onSubmit={handleSubmit}>
                            <Form.Group>
                                <Form.Label>Todo Title</Form.Label>
                                <Form.Control type='text'
placeholder='Enter here todo'
                                value={todo.title}

onChange={event=>setTodo({...todo,title:event.target.value})}></Form.Co
ntrol>
                            </Form.Group>
```

```
                            <Form.Group className='mt-3'>
                                <Form.Label>Todo
Description</Form.Label>
                                <Form.Control as={'textarea'}
type='text'
                                placeholder='Enter here Description'
                                value={todo.description}

onChange={event=>setTodo({...todo,description:event.target.value})}
// all data todo ... mean
                                ></Form.Control>
                            </Form.Group>
                            <Container>
                                <Button type='submit'
variant='primary'>
                                    Add Todo
                                </Button>
                            </Container>
                        </Form>
                    </Card.Body>
                </Card>
            </Col>
        </Row>
    </Container>
```

then create useState for title and desc

```
    const[todo,setTodo]=useState(
        {
            title:'',
            description:'',
```

```
            id:''


        })
```

change in field thn create onchange event

form submit event call

```
const handleSubmit=(event)=>{
            event.preventDefault()
            console.log(todo)
            setTodo({
                title:'',
                description:''
            })
        }
```

DisplayTodos.jsx

```
  return (
<Container>
    <Row>
        <Col>
        <Card className='mt-3 shadow-sm'>
            <Card.Body>
                <h3>All todos are here</h3>
                <DisplayCount />
                <ListGroup>
                    {
                        todos.map((todo,index)=>(
                        <ListGroup.Item key={index}>
```

```
                                <h4>{todo.title}</h4>
                                <p>{todo.description}</p>
                                <Button
onClick={event=>deleteTodo(todo.id)} variant='danger' size="sm">
                                    Delete
                                </Button>
                            </ListGroup.Item>
                            ))
                        }
                    </ListGroup>
                </Card.Body>
            </Card>
            </Col>
        </Row>
</Container>
  )
```

create useState

```
function DisplayTodos() {
    const[todos,setTodos]=useState([
        {
            title:'First Name',
            description:'First desc'
        },
        {
            title:'Second Name',
            description:'Second desc'
        }
```

```
    ]);
```

create card body

```
            <Card.Body>
                <h3>All todos are here</h3>
                <DisplayCount />
                <ListGroup>
                    {
                        todos.map((todo,index)=>(
                        <ListGroup.Item key={index}>
                            <h4>{todo.title}</h4>
                            <p>{todo.description}</p>
</ListGroup.Item>
                        ))
                    }
                </ListGroup>
            </Card.Body>
```

open DisplatCount.jsx

```
import React from 'react'
import { Card } from 'react-bootstrap'
const DisplayCount = ({todos}) => {
  return (
    <Card className='shadow-sm border border-0'>
        <Card.Body>
            <h4>Number of Todos : {todos} </h4>
        </Card.Body>
    </Card>
```

```
  )
}
```

src - create new folder redux

in redux create new folder

1. *actions*
2. *reducers*

In actions

new file create

1. *action-types.js*
2. *todo.js*
3. *user.js*

actions-type.js file

create constant type for action (add , update ,delete)

action-types.js

```
export const ADD_TODO="ADD_TODO"

export const DELETE_TODO="DELETE_TODO"

export const UPDATE_TODO="UPDATE_TODO"
```

todo.js

```
import { ADD_TODO, DELETE_TODO, UPDATE_TODO } from
"./action-types";


//add todo

export const addTodo=(todo) =>(

    {

            type:ADD_TODO,

            payload:todo

    })
```

```
//delete todo
export const deleteTodo=(id)=>({
    type:DELETE_TODO,
    payload:id
})

//update todo
export const updateTodo=(todo ,id)=>({
    type:UPDATE_TODO,
    payload:{todo:todo,todoId:id}
})
```

click on redux - open reducers

reducer create new file

todo_reducer.js

```
import { ADD_TODO, DELETE_TODO, UPDATE_TODO } from
"../actions/action-types"



const initialState=[]
export default(state=initialState, action)=>{

    switch(action.type)
    {
        case ADD_TODO:
            //add todo
            return [...state, action.payload]
```

```
        case DELETE_TODO:
            //delete todo
            const newState
=state.filter((todo)=>todo.id!==action.payload)
            return newState
        case UPDATE_TODO:
            //update todo
            state.map(todo=>{
                if(todo.id==action.payload.todoId)
                {
                    todo.title=action.payload.todo.title
todo.description=action.payload.todo.description
                }
                return todo

            })
          // return updateState
        default:
            return state
    }

    // if(action.type===ADD_TODO)
    // {

    // }
    // else if(action.type===DELETE_TODO)
    // {
```

```
    // }
    // else if(action.type===UPDATE_TODO)
    // {

    // }
    // else
    // {
    //     //return initial state
    // }

}
```

in redux create new file store.js

store.js

```
import {createStore, combineReducers} from 'redux';
import todoReducer from "./reducers/todo_reducer";

const rootReducer = combineReducers({todoReducer,})

const store = createStore(rootReducer)
export default store
```

App.js

used provider

(provider - react redux)

provider - sare store ko hierarchy m lega .

store used in all component

```
return (
  <Provider store={store}>
```

```
        <Container className='py-3'>
                <AddTodo />
                <DisplayTodos />

          </Container>

  </Provider>
      );
```

complete App.js file code

```
import './App.css';
import {Container} from 'react-bootstrap';
import AddTodo from './components/AddTodo';
import DisplayTodos from './components/DisplayTodos';
import { Provider } from 'react-redux';
import store from './redux/store';

function App() {
  return (
  <Provider store={store}>
        <Container className='py-3'>
                <AddTodo />
                <DisplayTodos />

          </Container>

  </Provider>
      );
}
```

```
export default App;
```

In Add todo:

```
const AddTodo = ({addTodo}) => {
```

addTodo pass

addtodo call

```
const handleSubmit=(event)=>{
            event.preventDefault()
            addTodo(todo) // calling addTodo
           // console.log(todo)
            setTodo({
                title:'',
                description:''
            })
        }
```

props k andr addtodo ko reducer() s map krna h

addtodo.jsx

```
const mapStateToProps=(state)=>({})
const mapDispatchToProps=(dispatch)=>({
    addTodo:(todo)=>(dispatch(addTodo(todo)))
})
```

dispatch s reducer() ko call krna h .

Connect redux to react

```
import {connect } from 'react-redux'
```

connect()

isme pass `mapStateToProps,mapDispatchToProps)`

```
export default
connect(mapStateToProps,mapDispatchToProps)(AddTodo)
```

AddTodo.jsx complete code

```
import React, { useState ,useEffect } from 'react'
import { Card, Container, Row ,Col, Form, Button} from
'react-bootstrap'
import DisplayCount from './DisplayCount'
import { addTodo } from '../redux/actions/todo'
import {connect } from 'react-redux'
import {v4} from 'uuid'


const AddTodo = ({addTodo}) => {
    const[todo,setTodo]=useState(
        {
            title:'',
            description:'',
            id:''

        })

const handleSubmit=(event)=>{
            event.preventDefault()
            addTodo({...todo,id:v4()})
          // console.log(todo)
```

```
            setTodo({
                title:'',
                description:''
            })
        }



  return (
    <Container>
        <Row>
            <Col md={12}>
                <Card className='shadow-sm'>
                    <Card.Body>
                        <h3>Add todo here</h3>
                        <DisplayCount />
                        <Form onSubmit={handleSubmit}>
                            <Form.Group>
                                <Form.Label>Todo
Title</Form.Label>
                                <Form.Control type='text'
placeholder='Enter here todo'
                                value={todo.title}

onChange={event=>setTodo({...todo,title:event.target.value})}></F
orm.Control>
                            </Form.Group>

                            <Form.Group className='mt-3'>
                                <Form.Label>Todo
Description</Form.Label>
```

```
                                    <Form.Control as={'textarea'}
type='text'
                                    placeholder='Enter here
Description'
                                    value={todo.description}

onChange={event=>setTodo({...todo,description:event.target.value}
)}
                                    ></Form.Control>
                                </Form.Group>
                                <Container>
                                    <Button type='submit'
variant='primary'>
                                        Add Todo
                                    </Button>
                                </Container>
                            </Form>
                        </Card.Body>
                    </Card>
                </Col>
            </Row>
        </Container>
    )
}


const mapStateToProps=(state)=>({})
const mapDispatchToProps=(dispatch)=>({
    addTodo:(todo)=>(dispatch(addTodo(todo)))
})
```

```
export default
connect(mapStateToProps,mapDispatchToProps)(AddTodo)
```

DisplayTodos.jsx

```
const mapStateToProps=(state)=>{
    //console.log(state.todoReducer)
    return{todos:state.todoReducer}
}

const mapDispatchToProps=(dispatch)=>({})
```

then connect()

```
export default
connect(mapStateToProps,mapDispatchToProps)(DisplayTodos)
```

DisplayTodos.jsx

```
import React, { useState } from 'react'
import { Button, Card, Col, Container, ListGroup, ListGroupItem, Row }
from 'react-bootstrap'
import DisplayCount from './DisplayCount'
import { connect } from 'react-redux'
import { deleteTodo } from '../redux/actions/todo'

function DisplayTodos({todos,deleteTodo}) {

    // const[todos,setTodos]=useState([
    //     {
    //         title:'First Name',
    //         description:'First desc'
```

```
    //     },
    //     {
    //         title:'Second Name',
    //         description:'Second desc'
    //     }


    // ]);
  return (
<Container>
    <Row>
        <Col>
        <Card className='mt-3 shadow-sm'>
            <Card.Body>
                <h3>All todos are here</h3>
                <DisplayCount />
                <ListGroup>
                    {
                        todos.map((todo,index)=>(
                        <ListGroup.Item key={index}>
                            <h4>{todo.title}</h4>
                            <p>{todo.description}</p>
                            <Button
onClick={event=>deleteTodo(todo.id)} variant='danger' size="sm">
                                Delete
                            </Button>
                        </ListGroup.Item>
                        ))
                    }
                </ListGroup>
```

```
            </Card.Body>
        </Card>
        </Col>
    </Row>
</Container>
  )
}
const mapStateToProps=(state)=>{
    console.log(state.todoReducer)
    return{todos:state.todoReducer}
}

const mapDispatchToProps=(dispatch)=>({
    deleteTodo:(id)=>(dispatch(deleteTodo(id)))
})

export default
connect(mapStateToProps,mapDispatchToProps)(DisplayTodos)
```

DisplayCount.jsx

```
import React from 'react'
import { Card } from 'react-bootstrap'
import { connect } from 'react-redux'

const DisplayCount = ({todos}) => {
  return (
    <Card className='shadow-sm border border-0'>
```

```
        <Card.Body>
            <h4>Number of Todos : {todos} </h4>
            {/* <h4>Number of Todos : 45 </h4> */}
        </Card.Body>
    </Card>
  )
}

const mapStateToProps=(state)=>({todos:state.todos})
const mapDispatchToProps=(dispatch)=>({})
export default connect(mapStateToProps,mapDispatchToProps)
(DisplayCount)
```

delete operation

URL [***react-uuid - npm (npmjs.com)***](https://www.npmjs.com/package/react-uuid)

install `npm i react-uuid`

```
import {v4} from 'uuid'
```

addTodo.jsx

```
            addTodo({...todo,id:v4()})
```

DisplayTodos.jsx

```
const mapDispatchToProps=(dispatch)=>({
    deleteTodo:(id)=>(dispatch(deleteTodo(id)))
})
```

🔥 [***Managing State using Redux in ReactJS | Complete Redux in one video in Hindi - YouTube***](https://www.youtube.com/watch?v=93p3LxR9xfM) ***(with durgesh)***

[***React redux tutorial in Hindi | Full Course - YouTube***](https://www.youtube.com/watch?v=DJekVo9-bSA) ***—- code step by step***

n ECMA Script6 also called ES6 and ECMAScript 2015 various new features were added:

The features are listed below:

- *The let keyword*
- *The const keyword*
- *Arrow functions*
- *Classes*
- *Modules*
- *The Rest Parameters*
- *The Spread Operator*
- *Destructuring Assignment*
- *The Object Destructuring Assignment*
- *Default values for Function Parameters*
- *Template Literals*
- *The for...of Loop*

What is Babel

Babel is a free and open-source JavaScript Transcompiler that is mainly used to convert ES6 (2015) and above code into a backward compatible code that can be run by older JS engines. Babel deploys a technique called Polyfill which basically means to fill many areas.

npm

npm is an abbreviation used for the node package manager. It is a package manager for JavaScript. It is the default package

manager that comes with NodeJS when you install it. It consists of a command-line interface and an online database of public packages and private packages that are paid which is called the npm Registry.