React js: It is a library developed by facebook.

⇒ To create the react js project run a node cmd

" npm create vite@latest "

⇒ Vite is a build tool which is used to build the application.

⇒ Once we created the react app we will see some folders & files

React App
- node_modules → provides the info & req files for react
- public → folder where we save all files to access
- src → will develop our react inside this
- .gitignore → will ignore when uploading in git
- package.json → defines the dependencies & version
- readme.md → contains the cmd & docs links of react

Component :- Group of elements is called as Component.

⇒ Components are reusable in react js.

Date:.....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

⇒ Components are 2 types → functional Components (latest) ✓
→ Class Components (older) ✗

⇒ To create a Component we use extension of .jsx.
↓
javascript + XML

Example:-

```
function functionName ( ) {
        return (  html code )
}
export default functionName ;
```

→ Component Name.

⇒ In react we write html code inside the javascript.

```
return (
    <h1> Hello World </h1>
);
```

⇒ Whenever you want to use the react js Component we have to import it inside that particular a React js file.

```
import ComponentName from "path" ;
```

⇒ To get the boilerplate code just use the rfce.

Note:- Component can return one html element at a time; So because of it we always need to write it inside .

Date:.....

Rules we need to follow when creating a Component :-

(1) Component name should always start with Uppercase Alphabet.

(2) When returning multiple elements from a Component always wrap it around a div tag or empty brackets <> </>

(3) When importing Component inside another jsx file always use import statement.

(4) When accessing an Component inside another Component always use the below syntax.

Syntax:

```
import SampleComponent from "./xyz/SampleComponent.jsx"

function AnotherComponent () {
    return (
        <SampleComponent />
            (or)
        <SampleComponent> </SampleComponent>
    );
}
```

Note:- To stop the server we have to use Command Ctrl+C

Date:.....

Importing css :-

⇒ By default we will have two css files — App.css, index.css

⇒ We can use any of css file to style the App.js or index.css

⇒ If we want to use seperate css of another css file for Component, we can do that by using a css file.

⇒ We have to import the css file in that component file.

Syntax:

```
import "/css path/file";
```

⇒ Also we can use different css frameworks to style in react.

placeholder:-

⇒ Here in react js we are writing the html code inside javascript because of it we are able to access variables of js inside html code.

⇒ To access the js data (variable) inside html we use the place holder ({ }).

Syntax:-

```
Const name = "myz";
<h1> I am {name} </h1>
```

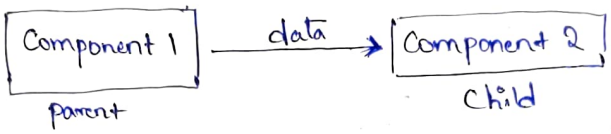
Date:.....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

Props:-

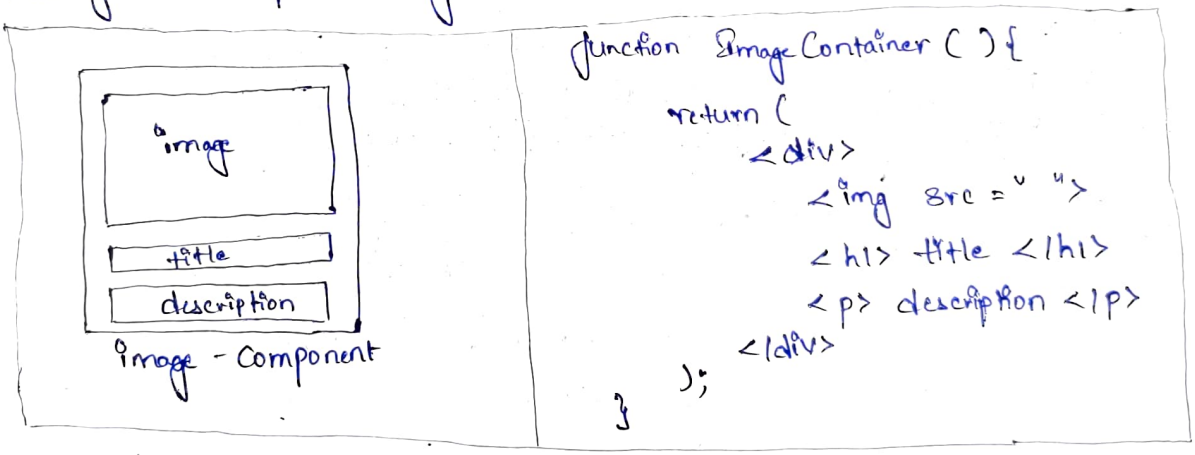
⇒ Props are nothing but properties. It is an object.

⇒ Props are mainly used to pass / send data between props.



⇒ Here the data Communication is one-way, where data will be passed from parent to child. where child cannot edit the passed data.

⇒ It is mainly used to reuse the Component. Imagine we are having a Component layout as shown below.



⇒ The Component which we shown above are not reusable as we are fixing the Content, but when we are using props, we can send the data which want to display.

⇒ To use props we have to write the parameter inside the function

Date:.....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

⇒ Let's say we are having a parent Component which is using the child Component.

```
function ParentComponent (){
    < ImageContainer
        source = "img path"
        title = "title"
        desc = "description"
    />
    :
    :
```

properties that are passing to child Component

⇒ Now let's see how to accept the data at the child side.

```
function ImageContainer (props) {
    return (
        <div>
            <img src = {props.source} />
            <h1> {props.title} </h1>
            <p> {props.desc} </p>
        </div>
    );
}
```

⇒ We can destructure the object using the spread operator

Date:.....

+91 9696963446/8186944555 ✉ Digitalbrolly@gmail.com 🌐 https://digitalbrolly.com/

Props Destructuring :-

⇒ In props we can use the keys to destructure the prop.

Way 1 :-

```
function ImageContainer (props) {
        { source, title, desc } = props
        ===
        ===
}
```

Way 2 :-

```
function ImageContainer ( { source, title, desc } ) {
        ===
        ===
}
```

Destructuring Nested props :-

```
function ImageContainer ( { user : { source, title, desc } } ) {
        ===
        ===
}
```

⇒ We also pass data blw Component we call it Children. Every Component is going to have only one children in prop.

↓

default keyword.

Date: .....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

State Managment :-

⇒ Used to manage the data inside the variables.

⇒ We use the hooks to manage the data.

↓

built-in functions.

useState:-

⇒ We use the useState hook to manage the state of a variable.

Syntax:

```
Const [variableName, setVarName] = useState()
```

⇒ Always write the useState at the start of function.

⇒ Whenever we are using & want to update the value inside the variable then we have to use the "set function".

Ex:

```
Const [name, setName] = useState("");
setName("Sample");
Console.log(name); //output: Sample
```

useEffect:-

⇒ manages the effects, In general if we try to change the state of a variable without a condition it will fall under infinite loop → leads to an error.

Date:.....

NOTE:- When we are inside a function, until we complete it, it will not get update.

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

⇒ But in useEffect we will have the tracker which will avoid the infinite loop.

Syntax:-

```
useEffect (callback function , [dependency]);
```

⇒ useEffect will track the changes in the variable / dependency which we mention inside the array.

Ex:-

```
useEffect ( () => {
        if (name == "") {
            setName ("Sample");
        } else {
            setName ("Mike");
        }
    }, [name] );
```

Event Listeners :-

⇒ These will be triggered when we are making any changes in our webpage either by mouse & keyboard.

⇒ There are 3 types of event listeners.

1. mouse Events
2. keyboard Events
3. Form Events

Date:.....

① mouse Events
- Click
- dbClick
- mouseOver
- mouseEnter
- ~~mouseExit~~
- mouseLeave
- mouseUp
- mouseDown

② keyboard Events
- keyDown
- keyPress
- keyUp

③ Form Events
- Submit
- reset
- focus
- blur
- Change
- input

⇒ We can use addEventListener & removeEventListner just like in js.

⇒ But here we use them 'inside' useEffect()'.

NpM:-

⇒ NpM means Node Package Manager.

⇒ Here we can get a lot of features (3rd party) to use it in react but to get it we need to install it inside the project.

Date:.....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

Fetching Data from API :-

→ To fetch the data from API we use the async & await as they are asynoronus code handlers.

⇒ When we are fetching the data from API the syncronus code will become Asynoronus.

⇒ To fetch data from API we use the below syntax.

Syntax:-

```
async function userHandler() {

}
```

Steps to fetch the API :-

① First, we have to get the 'url', where we get the API from & store it inside a variable.

② Then we have to create a function → async

③ Use the fetch method to get data from the given url.

Syntax:-

```
async function fetchData() {
        Const response = await fetch('url');
        Const data = await response.json();
}
```

Date:.....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

⇒ Displaying the info of API is very easy all we need to do is to return the html code.

Syntax:-

```
return (
    <div className = "parent-Containg" >
        { data . map ( (item) => {
            return (
                <div className = "usrSection" >
                    { item.key }
                </div>
            )
        }
    </div>
);
```

⇒ To send a request of data to backend we use the below approch.

Syntax:-

```
async function SendData ( ) {
    Const response = await fetch ( url, {
                                    method : "POST",
                                    headers : {
                                        "Content-Type" : "Application/json"
                                    },
                                    body : JSON.Stringify (data)
    } );
}
```

Date:.....

⇒ Always use the useEffect() with empty array. to fetch the data.

Error handling:-

⇒ It is used to handle errors in react js.

⇒ TO handle error we use try and catch keywords.

Syntax:-

```
try {
    Code trying / Checking
    if it is error & not
}
Catch (error) {
    Console.log(error);
}
```

DOM:-

⇒ It is known as Document Object Model.

⇒ It is generally known as Browser programming interface & Browser API

⇒ In javascrit we covered this topic

⇒ In react we work with Virtual DOM.

Date:.....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

Virtual DOM :-

⇒ Virtual DOM is a feature of react js.

⇒ When ever we are adding component to the DOM. It will create a copy of DOM.

⇒ That copy is called as Virtual DOM.

⇒ With the help of Virtual DOM our application speed increases

⇒ We cannot see the virtual DOM physically.

Short circuit Evaluation / Using Condition to display Info :-

⇒ It is used to display content based on boolean values.

⇒ Syntax :-

```
Const [val, setVar] = useState (true);

return ( <div>
          { Var && <h1> Hi </h1>
        </div>
```

Date: .....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

use Reducer Hook:-

⇒ It is an alternative & advanced option of use State.

⇒ Here in use Reducer we can update multiple states at a time.

Syntax:-

```
Const [state, dispatch] = use Reducer (function, initial State);
```

⇒ It is used to manage the complex state logic.

⇒ Let's Consider Creating a small application where we are increasing & decreasing a value.

Ex:-

```
function Reducer Example () {
    Const initial State = { count : 0 };
    function reducer Handler (State , action) {
        Switch (action.type) {
            Case 'Increment':
                return { count : state.count + 1 };
            Case 'Decrement':
                return { count : State.Count - 1 };
            default
                return state ;
        }
    }
```

Date:.....

Date:.....

```
function Counter () {

    const [ state , dispatch ] = useReducer (reducerHandler,
                                    initialState ) ;

    return (
        <div>
            <p> count : { state . count } </p>
            <button onClick = { () => dispatch (
                                { type : 'decrement' } ) } >
                decrement
            </button>
            <button onClick = { () => dispatch (
                                { type : 'increment' } ) } >
                increment
            </button>
        </div>
    ) ;
```

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

React Router Dom :- Browser Router
Hash Router
Memory Router

⇒ React router DOM will helps us to handle navigation. → install - react router dependency

⇒ Steps to implement React Router DOM.

① Install react router DOM → npm i react-router-dom
② Use the <BrowserRouter> tag to wrap the Content.
③ Define the routes in routes.
④ Link elements.

⇒ We use the BrowserRouter to wrap the '<App />' Component.

Syntax:

```
<BrowserRouter>
    <App />
</BrowserRouter>
```

⇒ After wrapping the the App Component with BrowserRouter, we have to use the link and routes to create a routes.

⇒ It is always better to create a seperate Component for routes.

⇒ If we use the NavLink it will add a active class to the anchor tag.

Date:.....

⇒ To handle the junk routes we will use '*' path.

⇒ To redirect / navigate from one page to another we can use - useNavigate hooks → returns function

Syntax: functionName("/path");

↓
use it handle the nav part

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

Example:-

```
function App() {
    return (
        <nav>
            <Link to="/endpoint"> Option </Link>
            :
        </nav>
        <Routes>
            <Route path="/endpoint" element={
                <Component />} />
            :
        </Routes>
    );
}
```

page-reloading will happen

routes path

(OR)

```
<a href="/path">
```

Use Context Hook:-

⇒ It is used to send the data throughout the DOM Tree by using a provider.

⇒ Basically, we have pass the data for each & every component in a DOM Tree when we are sending it to the grand child.

Date:.....

⇒ To avoid it we use the useContext().

+91 9696963446/8186944555 ✉ Digitalbrolly@gmail.com https://digitalbrolly.com/

Steps to implement use Context :-

1. Create a Context
2. use provided function
3. Create a Custom Component
4. use custom hook to send the data.

⇒ let's understand a case here. Imagine we are having 3 Components

- Component 1
- Component 2
- Component 3

⇒ Component 3 is inside Component 2 --->

```
function Component2 (prop){
    return (
        < Component 3 />
    );
}
```

⇒ Component 2 is inside Component 1

```
function Component1 (prop){
    return (
        < Component 2 data={prop.data} />
    );
}
```

```
function Component3 (prop){
    return (
        <h1>
            {prop.data}
        </h1>
    );
}
```

⇒ Here in the above example I am trying to pass data to Component 3 from Component 1. But as 'Component 3' is inside the Component 2

Date: .....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

⇒ First, I am passing data to Component 2 through props & then passing it to Component 3 through props.

⇒ The flow looks like this Component 1 → Component 2 → Component 3.

⇒ To simple this long process we are using useContext hook.

Creating a Custom Component :-

(1) Create a custom Component file.

(2) Use createContext() to create a new Content in Component.

Syntax:-

customComponentName

```
Const  ContentData = createContext();
```

(3) After creating a Content create a function to accept the Component in it. to create a provider

Example:-

```
function  DataProvider ( {Children} ) {
        Const [Data, SetData] = useState ("Data");
        return (
            <ContentData.provider  value = {{Data, SetData}}
                {Children}
            </ContentData.provider>
        );
}
```

Date:.....

Note:- Always declare Content outside the function.

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

④ After creating the provider we need to create a custom hook.

Syntax:-

(optional)

```
function useData(){
    return useContext(customContextName)
}
```

(or)

```
const customHookName = () => useContext(CCName);
```

Note:- Create custom hook in the same jsx file where we created the context. Also make sure to export both the Context & hook.

⑤ When using the context we have to import both Context & customHook.

Example:-

```
function App(){
    return (
        <DataProvider>
            <Component1 />
        </DataProvider>
    );
}
```

⇒ we are not sending any data through prop right now.

⇒ coz, we are having the useContext which is holding the data.

⇒ We have to access the data in Component 3 using custom hook

Date:.....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

(6) To use the data which is in useContext we import custom hook in Component 3.

Example:-

```
function Component3 ( ) {
    Const { Data } = useData ( );
    return (
        <h1>
            { Data }
        <|h1>
    );
}
```


Date:.....

Date:.....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

History of React js:-

⇒ In 2011 - Jordon Walke developed React js

⇒ Initially it is used to solve the performance issues in facebook UI.

⇒ It is initally called as "FaxJs".

⇒ In 2013 it was offically released open source.

⇒ There they introduced "virtual DOM" and "Component - architecture".

⇒ In 2015 - react native was launched. → develops Andriod & IOS Apps

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

UseEffect:-

It will re-renders (Component) when we make any changes to the tracka-ble element.

⇒ There are 2 rules in UseEffect.

① UseEffect always get Executed after Component Completes rendering.

② UseEffect will have two parameters,

- function → defines
  - ↓
  - Calls when we render Component
- dependency Array (Optional)
  - ↓
  - ⇒ If we didn't define this, then the UseEffect will always Executed when we re-renders Component
  - ⇒ If we define a value / a property inside it then it will get re-render whenever we are changing a particular property value

Syntax:-

```
useEffect(
        function,
        []
);
```

⇒ If function is having another function in it then we call it cleaning function.

↓

will get execute for previous values & it will be executed even before useEffect function.

Date: 28.-V-22

Promises:-

Used to handle the asynchronous statements of flow of Request. It will either provide the data or else it will provide error.

⇒ We have to use the exceptional handling to handle the errors in promise.

⇒ The promise will provide a response for us, then convert to json data.

———— ✕ ————

React Routing:-

⇒ If you are having any parameters in the routes then we use

`:id` → To get Id → url: `localhost:5173/path/1`

↓

Syntax: for dynamic path routing:-

```
<Route path = "/pathName/:id" element = {<Comp />} />
                          ↓
                      Parameter
```

⇒ To capture the parameter we can use useParam() hook, returns the parameter mentioned in the url.

↓

(object)

Date:.....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

⇒ Inte use `useSearchParams()` to help to read the query params.

↓

returns 2 values — parameters used for search
— update function.

⇒ In `useSearchParams()` data will store in form of map.

Navigation Layout :-

⇒ If at any point you want to have sub-navigation then we have to create a navigation layout.

Syntax:

```
<Route path="/path" element={<Comp />}>
    <Route path=":id" element={<Comp />} />
    :
</Route>
```

(main path; layout Component; subpath; Child Comp)

⇒ when we are rendering we need to define any one of the Child Component as index

⇒ Also to render the navigation layout along with child component we have to use 'Outlet' component in layout Component.

Syntax:-

```
function LayoutComp() {
    return (
        <NavBarComp />
        <Outlet />
    );
}
```

Date:.....

⇒ When you are having some common data that you want to share with child components in any level, then we use **Context**.

⇒ If we want to share data b/w layouts we have to do as shown below.

Syntax:-

```
function childComp() {
    let a = {
        key1: value,
        key2: value
    }
    return (
        <Outlet context={a} />
    );
}
```

On parent side

On child comp side:-

```
function subChildComp() {
    let obj = useOutletContext();
    :
}
```

Date:.....

⇒ here the outletContext hook will help us to get the hook from parent component.

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

Accepting form Data in react :-

To accept form data we have to follow some steps as mentioned below.

Steps :-

① Create a form and write the required form elements in it

② Create a function to handle the form data.

Syntax :-

```
function handleForm(event) {
    event.preventDefault(); // Stops page reload
    ...
}
```

③ Create a onChange event listener to the form & provide function.

Syntax :-

```
<form onChange={handleForm}>
    ...
</form>
```

④ To handle the form data create an object to maintain the data.

Syntax :-

```
Const [formData, setFormData] = useState(
    {
        name : 'default value',
        :
    });
```

Date:.....

+91 9656963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

⑤ To insert the values we have to use object destructuring.

Example of object destructuring :-

```
let a = {
    name : "Tony Stark",
    age : 30,
    state : "State"
};

let {name, age, state} = a;
```

⑥ But as we are using set method to assign data we have to do the below option.

Syntax :-

```
function handleForm (event){
    event.preventDefault();
    setFormData (
        (prevFormData) => {
            const name = event.target.name;
            return ({
                ...prevFormData,
                [name]: event.target.value
            });
        }
    );
}
```

Date :.....

⑦ When we are using / accepting data from any radio button & Checkbox & ~~textbox~~ textarea

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

Syntax:-

```
<input type="radio" name="gender" value="male"
id="male" onchange={handleFormData} >
<label for="male" > Male </label>
```

Syntax for Checkbox:-

```
<input type="Checkbox" name="term" onchange={han-
-dleFormData} id="terms" >
<label for="terms" > Terms & Conditions </label>
```

```
function handleFormData(event){
    event.preventDefault();
    setFormData(
        (prev) => {
            name = event.target.name;
            return ({
                ...prev,
                [name] : (name == "terms")?
                    event.target.checked :
                    event.target.value ;
            })
        }
    );
}
```

Date:.....

AXIOS :-

It is used to handle the API's. It's a 3rd panty lib which is used as an alternative to fetch.

⇒ run Command `npm install axios`

Working with axios :-

⇒ Use a create() method to call API, inside it we have to mention 'baseURL'

Syntax :-

```
Const api = axios.create({
              baseUrl: "url here"
            });
```

⇒ Declare methods to work with the api. Ex:- get, update, delete, insert

Syntax to getData :-

```
export const getData = (url) => {
                         return api.get(url);
                       }
```

Syntax to insertData / saveData :-

```
export Const saveData = (url, payload) => {
                          return api.post(url, payload);
                        }
```

Date:.....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

Syntax for updateData :-

```
export const updateData = (url, payLoad) => {
    return api.put(url, payLoad);
}
```

Syntax for deleteData :-

```
export const deleteData = (url) => {
    return api.delete(url);
}
```

React Hook Form :-

⇒ It is a lib which makes the form data acceptance easy.

⇒ Follow the below steps to work with react hook form.

① install 'React Hook Form'.

② To use the react hook form import "useForm" hook

③ React Hook form will provide the useForm → object

④ we have to register the fields to accept data.

Syntax :-

```
const { register } = useForm();

<form>
    <input type="text" {...register("name")} />
    Date:.....
</form>
```

⑤ That's it, react will read the data which we enter in that input field.

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

Validating data in form :-

(1) To add the basic validation of "required" we can actually use hook-form.

Syntax:-

```
<form>
    <input type="text" {...register("field-name",
        {required: true})} />
</form>
```

(2) If you want to validate data based on pattern it's super easy

Syntax:-

```
<form>
    <input type="text" {...register("name",
        {required: true,
        pattern: regex})} />
</form>
```

(3) we can set the min & max characters that will allow in a form.

Syntax:-

```
<input type="text" {...register("name",
    {required: true,
    pattern: regex,
    minLength: size
})};
```

Date: .....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

Handling errors :-

(1) To handle errors in forms we are having "errors" in useform hook. → destructure it along with register

Syntax :-

```
const { register, formstate: {errors} } = useForm();
```

(2) 'errors' is an object which contains the field name and message.

(3) we have to provide the error message to display.

Syntax for Basic req. validation :-

```
<input type="text" {...register("name",
                    {required: "error msg"})} />
```

Syntax for validation :-

```
<input type="text" {...register("name",
                    {required: "errormsg",
                     pattern: {
                         value: regex,
                         message: "error msg"
                     }})} />
```

Syntax for minLength :-

Date:.....

```
<input type="text" {...register("name",
                    {minLength: [
                        value: len,
                        message: "msg"
                    ]}
                    })} />
```

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

To handle async process:-

(1) we are having isSubmitting hook which will return boolean value.

(2) we can use the asyn functions to handle async tasks.

To set custom errors :-

(1) In hook form we are having setError helps us to handle any custom errors.

(2) To work with it just wrap the code in try & catch block

Syntax :-

```
try {
    //input fields
} catch (error) {
    setError ("name", {
        message: "error msg"
    })
}
```

Date:.....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

To auto populate values in input fields :-

(1) In react we can auto populate the values

Syntax:-

```
const { register, formState: { errors } } = useForm ({
    defaultValues: {
        fieldname: "value",
        ...
    }});
```

Custom hooks ():-

⇒ In simple hooks are functions which are mainly used to reuse the logic / code.

⇒ So to create custom hooks all we need to do is to create a custom component and use name as 'use Hook Name'.

⇒ Also it is suggestable to place all the files seperatley.

Syntax:-

```
export function useHookName ( ) {
    // reusable logic
}
```

hook comp.

Date:.....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

useRef Hook:-

⇒ It will provide us a mutable object without re-rendering the react DOM.

⇒ At the same time we can access the DOM elements without re-rendering.

⇒ Syntax:

~~export~~ ~~const~~ ~~val~~ = ~~useRef (init-value);~~

~~function increase () {~~

~~return~~

Syntax:-

```
function Compt ( ){
    const var = useRef (0);
    const [data, setData] = useState (0);
    function increase ( ) {
        var = var +1 ; // increases the value of val
    }                  // but will not be re-renders
                       page.
    function handle handleSubmit (event){
        event.preventDefault ();
        SetData (val);
    }
    return (
        <form onSubmit = {handleSubmit}>
        {data}
        <button onclick = {increase}>
        increase
        </button>
```

Date:.....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

```
        <button> onclick Submit </button>
    </form>
  );
}
export default Comp1();
```

Redux

⇒ It is a Centralized State Containing. works with js lib.

⇒ It is used to avoid prop-drilling

⇒ To avoid we are using Context Api.

Date:.....

⇒ But when we are having more common data then managing the states will become more complex.

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

How redux works:-

⇒ It will have a store init which will store all the states.

⇒ Those states are only "Read-only" → Can't change directly.

⇒ We have to use action → describes what happened
↓
plain js code

⇒ Then we have to dispatch action.

⇒ After dispatching "reducer" will take care of it. & then re-renders the page.
↓
term & not a hook.

Working with Redux:-

Date:.....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

⇒ let's work with redux, to work with redux we can install redux.

```
npm install react-redux
```

⇒ Redux working implementation:-

(1) To work with redux we have to use a reducer which will update the state.

(2) create a reducer function as shown below.

Syntax:-

```
Const reducerName = (state = {key : init-value}, action) => {
    return {
        key : state.key  //updater
    }
};
```

(3) Now register a store by using react-redux hook - createStore.

Syntax:-

```
Const storeName = createStore (reducerName);
```

(4) Now we had a store with the inital value in it. Now we have to create a subscribe to help with the action.

Date:.....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

Syntax:-

```
Const subscribeName = () => {
    Const latestState = storeName.getState();   // default method
    Console.log(latestState); // invokes when state changes
};
```

⇒ The subscribe function will automatically get invoked by the reducer whenever there is an action performed on the key.

(5) To make it happen we have to tell these subscribe to the redux store

Syntax:-

```
store.subscribe(subscribeName);
```

(6) We need to perform an action to update the state in a store. To do that we use dispatch method.

Syntax:-

```
store.dispatch(obj);   // obj -> { type: "type-of-action" }
```

(7) Once we are done with the configuration we have to use "provider" component from react-redux to make the store available for other components.

Note:- Always use provider on top-level component.

Date:.....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

Syntax:-

```
<Provider store = {reduxStoreName} >
    <App />
</Provider>
```

(5) To get data from the store we use useSelector() hook.

Syntax:-

```
const val = useSelector ( state => state.key );
```

NOTE:- Whenever we are changing the state the old state will always be replaced by new one.

Ex:-
```
const init = {
    key1: "value1"
    key2: "value2"
}
```
→ we update key1 value to 'update1' & not returning 2nd state.

⇓

```
const init = {
    key1: "update1"
}
```
→ key2 of 2nd state was removed as we are not returning key2 with key1.

Date:.....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

Payload for the action:-

⇒ When we are sending a dispatch request we are just passing type so far. Now to send more data/info then we use 'payload'.

⇒ We can give any name to the payload & the reducer end the name have to be same.

Syntax:-

```
dispatch ({type: "val", payloadName = "name"});
```

Handling multiple states in redux:-

⇒ Working with multiple states are easy as we just need to declare them in same object.

Syntax:-

```
Const reduxStore = (State = {st1: val1, st2: val2, action){
    ...
};
```

⇒ remember when we are handling the state of one the other one also should pass as well.

Date:.....

Syntax:-

```
if (action.type == "...")
    return {
        st1: st1.val //updated state
        st2: st2...
    }
```

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

Redux Toolkit:-

⇒ It is just an extension of redux, to help us by making the redux more simple.

⇒ To work with redux toolkit follow the below steps.

Steps:-

(1) import createSlice() from toolkit.

Syntax:-

```
Const name = createSlice() {
        name : "sliceName",
        initialState : { key:val, key2:val },
        reducers : {
            methods() → will be triggered based on action
        }
}
```

(2) create methods inside reducer, the method will recieve the state and action as an args. Also data returing the data is simple as all we need to do is to write directly & in mutable way.

Syntax:-

```
methodName ( state , action (op)) {
        return updated State Value;
}
```

Date:.....

Note:- The Toolkit will take care of creating a Copy of the state & managing the updated state.

③ To registre the slice we have to use Configure Store ( ).

↓

Combines all the reducers in diff slices into one.

Syntax for Single reducer:-

```
Const StoreName = configureSlice ( {
                    reducer : sliceName .reducer
                  } );
```

Syntax for multiple reducers :-

```
Const StoreName = configureSlice ( {
                    reducer : {
                      redeName : sliceName .reducer,
                      :
                    }
                  } );
```

④ To dispatch the action we have to use action creators (identif. -eg.

↓

Created by toolkit with names of reducer methods.

⑤ Export the actions, so that we can use them in desired location.

Syntax:-

```
export const actionName = sliceName . actions;
```

Date:.....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

(6) To use those action methods import them in a Component & use inside dispatch.

Syntax:-

```
dispatch (actionName.method());
```

Syntax to send payload:-

```
dispatch (actionName.method(value));
(or)
dispatch (actionName.method({obj}));
```

NOTE:- When passing the extra value it will always be assigned to "payload" by toolkit

---

NOTE:- we can't use the http request inside reducers.

Working with async Code:-

=> In react to work with async code we use action creators.

=> we use thunk to handle async function.

=> To do that follow the steps.

Date:.....

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

① Using Redux-Thunk as middleware :-

(1) Create a custom action creator. which returns a function.

Syntax :-

```
export Const funName = () => {
    return async (dispatch) => {
        dispatch ({type: "name"});
        try {
            Const res = await fetch (url);
            Const data = await res.json();
            dispatch ({type: "name", payload: data});
        }
        Catch (error){
            dispatch ({type: "name", payload: error});
        }
    };
```

(2) Create a reducer to handle the dispatch.

Syntax :-

```
function reducerName (state = {...}, action){
    =
    =
```

Date: ..... }

+91 9696963446/8186944555 Digitalbrolly@gmail.com https://digitalbrolly.com/

Using redux toolkit :-

① we use createAsyncThunk from toolkit.
→ needs 2 args — name
— async function

Syntax:-

```
export const funName = createAsyncThunk(
    name,  → custom name
    async () => {
        const resp = await fetch(url);
        return resp.json()
    }
);
```

② when we are calling this function toolkit will provide 3 actions

① name / pending (before start)
② name / fulfilled (after success)
③ name / error (if any error)

③ we need to create a slice to handle these actions.

Syntax:-

```
const sliceName = createSlice({
    name : 'name',
    initialState : {
        ==
    },
    reducers : { }
    extraReducers : (builder)
    => {
```

Date:.....

```
        builder.
        addCase(thunkFunName.pending, (state) => {
                state.status = 'loading';
            }).
        addCase( ...
            =
    });
  },
});
```

④ Using it inside a component is also easy. Call the dispatch inside useEffect.

Syntax:-

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

Date:.....