My personalised react note

I learn that the if(loop) can be optimise further more-

If there is only if and no else then && is better

1. The &&

Without using the if loop we can directly write

<button onClick={()=> counter<20 && setcounter(counter+1)}>Add</button>

Here see the second condition will only run if the first one is statisfying

1. The conditional

If there is if else, we can go with condition? true: false

**React virualDOM**

The virual dom is created using the ReactDOM.createRoot

Especially the createRoot behind the scene it create a Dom like structure. It then compare the main dom and its dom and update only the thing that are updated in the UI

But what the browser do is it removes the whole dom and repaint the dom which we see it in the from of browser reload

Virtual DOM is updated with the alogo of **react fiber**

**Reconciliation** – is a differentiation algorithm uses to diff one tree with another to determine which parts need to be changed.Reconciliation is the algorithm behind what is popularly understood as the "virtual DOM."

REACT PROPS PROBLEM THAT I SAW AND FIX

**OPTIONAL CHAINING**

In react when we try to pass a props of object

<Card  channel="tenzdelek"/>

 <Card someobj={myobj} channel="monalm"/>

Here one card is not passing the object so in the app.jsx it will throw error

To handle that we have **?.** which makes it optional to use

<h1 >{someobj?.name || "default header using obj"}</h1>

Now it will work properly

Remember that normal strings and all doesn’t need it

**Context API**

**THE CONCEPT OF PROPS DRILLING-**

Back then when the concept of context Api was not familiar what we use to do is we have a app.jsx file and that is sending a data to a card component. But here is the tricky part, the card component that is going to receive the data is in another component call dashboard and in dashboard there are two side left component and right component. In the right component there are two more component top and bottom and finally inside the top component the card component is there.

So now for card component to get the data send by the app component the props need to be received by the upper component like the dashboard then right one, then the top one then finally the card one.

But the dashboard and the right one and the top one doesn’t need that prop only the card component needed that prop. So that method was highly not optimized.

Then there was a hunch about making one global file {name: “tenzin”} then only the one who needs that data can go there and access it. No use case of props. The data that is send by the app.jsx is now send as a object not props to that file and other components access it if they need it

**THIS WAS CALL CONTEXT API**

BUT the context Api is only associated with react. Then there is REDUX state management which is associated with many other frameworks. For react the redux used is call **react-redux.**

There is a easier version of redux which is call **redux-toolkit (RTK)**

There are basically four steps

1. Creating context
2. Creating the provider
3. Sending the values
4. Receiving the values

**Creating my first context**

Create a file name context and inside that create a file name usercontext.js. inside it declare the context and export it.

The context act as a provider and a wrapper. every component that is wrap around the context has the access to the global context created.

import React from "react";

const UserContext =React.createContext()

export default UserContext;

after the context is created now we have to define the provider and the children to whom we wanted to wrap

so for that we create a new jsx not js file in context folder and put this

import React from 'react'

import UserContext from './UserContext'

const UserContextProvider = ({children}) => {

    const [user,setUser]=React.useState(null)

  return (

    <UserContext.Provider value={{user,setUser}}>

        {children}

    </UserContext.Provider>

  )

}

export default UserContextProvider

here the children is similar to what we learn about outlet. The components that is wrap in a div is pass as a children

now we will see how can we send the data first in the context

create a component like login

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

import UserContext from "../context/UserContext";

const Login = () => {

  const [username, setusername] = useState("");

  const [password, setpassword] = useState("");

  //remember the setuser is made in contextprovider file

  const {setuser}=useContext(UserContext)

  const handlesubmit = (e) => {

    e.preventDefault();

    setuser({username,password})

  };

  return (

    <div>

      <h2>Login</h2>

      <input

        type=" text"

        value={username}

        onChange={(e) => setusername(e.target.value)}

        placeholder="username"

      />

      <input

        type=" text"

        onChange={(e) => setpassword(e.target.value)}

        value={password}

        placeholder="password"

      />

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

    </div>

  );

};

export default Login;

now see here we first create two input where we take name and password where the values are made using the usestate and onchange the new input is set. Now to send the data. When we submit the button it triggers a handlesubmit button, firstly prevent the default behavior. Then with the use of usecontext , destructure the setuser state which was pass from the provider. Then in the function

setuser the username and the password.

Now here is the receiver

import React,{useContext} from 'react'

import UserContext from '../context/UserContext'

const Profile = () => {

    const {user}=useContext(UserContext)

  if(!user)

  {

    return <div>pls login</div>

  }

  return <div>Welcome {user.username}</div>

}

export default Profile

REDUX

STEP1 CREATE THE STORE (EVERY APP HAS 1 STORE)

STEP2 CREATING REDUCER (SLICE) THE SLICE NEED 3 THINGS

NAME, INITIAL STATE AND THE REDUCER, REDUCER IS JUST THE object of FUNCTION

Every reducer have two para- state and action

Then we either dispatch or selector

BLOG

.env FILE- SHOULD BE IN ROOT FOLDER (WHERE README AND OTHER ARE IN)

.env should be added to gitignore

To access it we have many ways depending on which platform we deploy if you have created using create react app then you use REACT\_APP\_THE\_NAME AND ACESS IT USNG clg(process.env. REACT\_APP\_THE\_NAME)

The REACT\_APP prefix is mandatory and the process.env is the way

But in vite we have different way, the variable should be label VITE\_APPWRITE\_URL instead of REACT\_APP\_APPWRITE\_URL.

AND ACCESS IT USING

onsole.log(import.meta.env.VITE\_APPWRITE\_URL)

VITE\_APPWRITE\_URL="https://cloud.appwrite.io/v1"

VITE\_APPWRITE\_PROJECT\_ID=""

VITE\_APPWRITE\_DATABASE\_ID=""

VITE\_APPWRITE\_COLLECTION\_ID=""

VITE\_APPWRITE\_BUCKET\_ID=""

The appwrite url is the end point copyed from the appwrite