**PROPS**

**WHAT IS ‘PROP DRILLING’ ?**

**->**

Child C

Child B

Child A

Parent

\*\* (view useContext\_concepts folder)

**PROPS DRILLING CAN BE OVERCOME USING ‘CONTEXT API’ AND ‘USECONTEXT()’ BUT HOW?**

**->**

* **‘CONTEXT API’**

1. Context API is used to make a direct connection between two componets without passing data through every components present there. Means, to overcome the passing of data through nested components unnecessarily, we use **Context API.**
2. Basic three terms have to remind. They are **Create, Provider** and **Consumer.**
3. Firstly, creation of context data. (const data = createContext())
4. Sometimes contextAPI becomes problematic when data is transmitted in a nested format. So, to solve this p[roblem we use the ‘useContext()’ hook.

{/\* provider in contextAPI \*/}

<data.Provider value={name}>

<ChildC/>

</data.Provider>

{/\* consumer in contextAPI \*/}

<data.Consumer>

{(name) => { //function definition

return <h1>My name is {name}</h1>

}}

</data.Consumer>

* **USECONTEXT HOOK ()**
* useContext() Hook can be used together with the useState Hook to share state between deeply nested components more easily than with useState alone.
* Basic three terms have to remind. They are **Create, Provider** and **useContext.**
* 'useContext()' is used in React to avoid "prop drilling" — the process of passing props through multiple levels of components. Instead of passing data down manually through props from a parent to a deeply nested child component, useContext() allows you to share values (like state, functions, or constants) directly across various components without having to pass them down each time through the component tree.
* 'useContext()' is a simpler alternative to managing global state than using more complex state management libraries like Redux for smaller-scale state sharing.
* Using thisd hook we can fetch data from parent component to child componrnt and vice versa.
* How useContext() Works

<---------------------------------------->

**Create a Context**: ==>> You first create a context using React.createContext() and set a default value for it.

**Provide Context Data:** ==>> Use the <Context.Provider> component at a higher level in the component tree to provide the value that all components within the tree can consume.

**Consume Context Data:** ==>> In any component, you can use the useContext() hook to access the value provided by the nearest <Context.Provider>.

**STATE**

**CLASS COMPONENT**

**STATE**

**HOOKS**

* The state is a build-in react object that is used to contain data or information about the component.
* A state can be modified based on user action or network changes.
* After the state changes, React re-render the component to the browser accordingly.
* We can’t direclyt update any cariable / object of the React component, thats why state is used to update or modify the data / object.

**LIFECYCLE COMPONENTS:**

REDER() METHOD

COMPONENTDIDMOUNT() METHOD

COMPONENTDIDUPDATE() METHOD

COMPONENTWILLUNMOUNT() METHOD

**HOOKS**

* Must be written inside function component.
* Must be imported in the component.
* Must be called at the top level of the component.
* Cannot be conditional.
* all hooks must be started with ‘use’

-(\* useContext Hook already discussed\*)**useContext**

useContext

useEffect

useState

useReducer

useCallback

useRef

custom hooks

useMemo

* If familiar with React class lifecycle, **useEffect() hook** can be used as **ComponentDidMount**, **ComponentDidUpdate**, and **ComponentWillUnMount** in combine.
* Tow components are there in **UseEffect hook.** One is a function and another is a variable / optional.

useEffect(() => {

console.log("component mounted");

//page loading everytime, then this cmd rendered or shown in console.

});

In this case the console will show the “console.log” wheneverpage is reloaded along with any activity done or button clicked.

useEffect(() => {

console.log("component mounted");

//page loading everytime, then this cmd rendered or shown in console.

},[]);

In this case the console will show the “console.log” only in that case when the page is reloaded.

This prevents the page to update the change everytimes when the pageis loaded.

useEffect(() => {

console.log("component mounted");

//page loading everytime, then this cmd rendered or shown in console.

}, [data]);

const upDateData = () => {

setData("Haldar")

console.log("Haldar")

};

In this case, the page will be updated once after reloading for a particular state, here that is ‘data’.

* **UseRef () hook**  is used to manipulate dthe DOM (Document Object Model) in Javacript or React Js.
* This hook is used when we are unable to use state or props directly.

const refEle = useRef("");

console.log(refEle);

Returning value “current” in console.

Any value is given as input will be stored in “current.value” in console.

* Memo-Js concept (Memorization) is used to increase application performance.
* **UseMemo() hook** memorizes the same method or value, thats why for same or repeat usage it doesn’t calculate.
* It also takes one function and one dependency just like useEffect hook.
* Thus the memory speed increases and storage & time consuming becomes lesser.

function multi() {

console.log("\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

return add \* 10

}

//in this case for both add & sub buttons this o/p is shwon in console. That's a issue

//To solve it we use useMemo hook

const multiply = useMemo(

function multi() {

console.log("\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

return add \* 10

}, [add]

//it is written for that, only for add function this function will be called.

)

* **useCallback() hook** is also similar to useMemo() hook.
* Controlled and Uncontrolled components in React.