**CLASS-10-Prop Drilling in React**

**What are Props?**

When React recognizes an element as a user-defined component, it sends the component’s JSX properties and children as a single object. This item is known as **props**.

For instance, the following code displays Hello, Hulk on the page :

function Welcome(props) {

return <h1>Hello, {props.name}</h1>

}

const root = ReactDOM.createRoot(document.getElementById('root'))

const element = <Welcome name="Hulk" />

root.render(element)

## Introduction to Prop Drilling

Data is frequently transferred amongst components in a standard React app using props. Sharing this information manually might be difficult, especially when it involves several nested components. Additionally, it may be difficult to share data between two child components. As a result, global state management is required.

**Prop drilling** in react is the process of passing data from one component via several interconnected components to the component that needs it.

The name drilling refers to this process of forcing these components to take in unnecessary data and pass it on to the following component, which in turn sends it on to the next component, and so on until it reaches its intended destination. The reusability of components and app performance may suffer as a result in a significant way.

It is not a good idea to pass data via several components when building neat, reusable, and DRY code.

Prop drilling in react is sometimes advantageous for smaller apps since there are lesser components and conditions to control.

## Why We shouldn’t Use Prop Drilling?

You must manually send the state and data through all of the intermediate levels that do not need it to update the state of the component lower down in the tree. The result is long and challenging to maintain code.

Furthermore, there are greater possibilities for mistakes like renaming the props midway by mistake, refactoring some data's structure, props being forwarded more often than is necessary, using default props unfairly, or using default props unfairly or insufficiently.

Additionally, there are numerous other circumstances in which prop drilling in react can be quite frustrating during the maintenance and refactoring process, this could become even more complicated in large-scale projects.

## Creating React Application

Use the following command to create the React Application.

npx create-react-app ReactApp

Then run the following command to move to the ReactApp folder.

cd ReactApp

### Example - 1 : With Using Prop Drilling

**propDrilling.js :**

import React, { useState } from 'react'

function Parent() {

const [title, setTitle] = useState('Scaler Topics')

return (

<>

<div>This is a Parent component</div>

<br />

<ChildA title={title} />

</>

)

}

function ChildA({ title }) {

return (

<>

Prop is received from the Parent component and passed on to the ChildB component

<br />

<ChildB title={title} />

</>

)

}

function ChildB({ title }) {

return (

<>

Prop is received from the ChildA component and passed on to the ChildC component

<br />

<ChildC title={title} />

</>

)

}

function ChildC({ title }) {

return (

<>

Prop is received from the ChildB component

<br />

<h3> Welcome to</h3>

<h4>{title}</h4>

</>

)

}

export default Parent

In this code, the title prop is passed from the Parent component and then to the ChildA, then to ChildB, then finally to the ChildC component.

**App.js :**

import Parent from './propDrilling'

export default function App() {

return (

<div className="App">

<Parent />

</div>

)

}

## Developing an Application of Prop Drilling with Several Levels of Nesting

Let us consider an app where the user is welcomed by their name upon logging in to the application. The structure and the hierarchy of the app are mentioned in the following images :

app

mainpage navbar

content

messege

We are having trouble because the component generating the message of welcome is deeply buried inside our application, while the user object holding the name of the user is only accessible at the level where the root component is available. This implies that we must somehow transmit this user object to the part that generates the message of welcome.

The user object prop is shown by the blue arrows as it descends from the root component that is App, via numerous intermediate components, onto the specific Message component that requires it. Finally, it displays the welcoming message along with the name of the user who has signed in.

This is an example of a prop drilling situation. To get around this seeming issue, developers frequently turn to the Context API without carefully considering the various issues that could result.

import { useState } from 'react'

function App() {

const [user, setUser] = useState({ name: 'Aegon' })

return (

<div>

<Navbar />

<MainPage user={user} />

</div>

)

}

export default App

// Navbar Component

function Navbar() {

return <nav style={{ background: '#10ADDE', color: '#fff' }}>Demo App</nav>

}

//MainPage Component

function MainPage({ user }) {

return (

<div>

<h3>Main Page</h3>

<Content user={user} />

</div>

)

}

// Content Component

function Content({ user }) {

return (

<div>

<Message user={user} />

</div>

)

}

//Message Component

function Message({ user }) {

return <p>Welcome {user.name}</p>

}

To avoid any external imports and for the sake of simplicity, we have declared different components in a single file rather than splitting them.