**NAMASTE REACT-AKSHAY SAINI**

**CHAPTER-8 –LET’S GET CLASSY THEORY NOTES**

1. How do you create nested routes using react router dom configuration?

Ans:-

import { createBrowserRouter, RouterProvider } from "react-router-dom";

const appRouter = createBrowserRouter([

{

path:"/",

element:<AppLayout/>,

errorElement:<ErrorPage/>,

children:[

{

path:"/",

element:<Body/>

},

{

path:"/about",

element:<About/>,

children:[

{

path:"profile",

element:<Profile/>

}

]

},

{

path:"/contact",

element:<Contact/>

},

{

path:"/restaurant/:resId",

element:<RestaurantMenu/>

}

]

},

])

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(<RouterProvider router={appRouter}/>);

1. What is createHashRouter and createMemoryRouter?

Ans:-

**createHashRouter** router is useful if you are unable to configure your web server to direct all traffic to your React Router application. Instead of using normal URLs, it will use the hash (#) portion of the URL to manage the "application URL".

Using hash URLs is not recommended.

**createMemoryRouter**

1. Instead of using the browsers history a memory router manages it's own history stack in memory. It's primarily useful for testing and component development tools like Storybook, but can also be used for running React Router in any non-browser environment.

For example: -

import {

RouterProvider,

createMemoryRouter,

} from "react-router-dom";

test("event route", async () => {

const FAKE\_EVENT = { name: "test event" };

const routes = [

{

path: "/events/:id",

element: <CalendarEvent />,

loader: () => FAKE\_EVENT,

},

];

const router = createMemoryRouter(routes, {

initialEntries: ["/", "/events/123"],

initialIndex: 1,

});

render(<RouterProvider router={router} />)

1. What is the order of lifecycle method in Class Based component?

Ans:-

* constructor >> render >> componentDidMount >> componentDidUpdate >> componentWillUnmount(after leaving the component)
* if one parent component has two child component calling like then life cycle will call like below:-

Parent constructor

parent render

first child constructor

first child render

second child constructor

second child render

First child componentDidMOunt

Second child componentDidMOunt

Parent Child componentDidMOunt

So first it get renders(Render phase) and then updates(Commit phase).

##### When Parent Child component are present where child having api call as well:

Render Phase--

Parent constructor

Parent render

Child constructor

Child render

Commit Phase--

Parent componentDidMount

Child componentDidMount (consist of fetch api call)

Child re renders

This happened because child have an api call and it needs time to fetch the data and update DOM, where fetching is asynchronous thus it loads Parent DidMount first.

Refer: <https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/>

1. Why do we use componentDidMount?

Ans:-

componentDidMount gets called after the initial render in the react lifecycle and hence its becomes perfect place to call or fetch data from API.

1. Why do we use componentWillUnmount? Show with example.

Ans:-

We use it to cleanup any timers , canceling network requests, or cleaning up any subscriptions that were created in componentDidMount() , which may affect performance of our project.

For eg: clearning setInterval or any counter which needs to be clear after leaving the component.

componentDidMount() {

this.timer = setInterval(() => {

console.log("Namaste React OP");

}, 1000);

console.log("Child Component DidMount");

}

componentWillUnmount() {

clearInterval(this.timer);

console.log("Component will Unmount");

}

1. Why do we use super(props) in Constructor?

Ans:-

class MyComponent extends React.Component {

constructor(props) {

super(props);

}

}

When implementing the constructor() function inside a React component, super() is a requirement. Keep in mind that your MyComponent component is extending or borrowing functionality from the React.Component base class.

This base class has a constructor() function of its own that has some code inside of it, to setup our React component for us.

When we define a constructor() function inside our MyComponent class, we are essentially, overriding or replacing the constructor() function that is inside the React.Component class, but we still need to ensure that all the setup code inside of this constructor() function still gets called.

So to ensure that the React.Component’s constructor() function gets called, we call super(props). super(props) is a reference to the parents constructor() function, that’s all it is.

We have to add super(props) every single time we define a constructor() function inside a class-based component.

If we don’t we will see an error saying that we have to call super(props).

The entire reason for defining this constructor() funciton is to initialize our state object.

So in order to initialize our state object, underneath the super call I am going to write:

class App extends React.Component {

constructor(props) {

super(props);

this.state = {};

}

// React says we have to define render()

render() {

return <div>Hello world</div>;

}

};

So we have defined our constructor() method, initialized our state object by creating a JavaScript object, assigning a property or key/value pair to it, assigning the result of that to this.state. Now of course this is just an example here so I have not really assigned a key/value pair to the state object, its just an empty object.

Refer- https://stackoverflow.com/questions/30571875/whats-the-difference-between-super-and-superprops-in-react-when-using-e

1. Why can't we have the callback function of useEffect async ?

Ans:-

You cannot directly make the callback function supplied to the useEffect hook async because: async functions implicitly return a promise, and useEffect expects its callback to either return nothing or a clean-up function.