Assignemnt of Chapter - 8

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

For making nested routes, you need to provide path object in the children of the children as relative path like below:

```
const appRouter = createBrowserRouter([
    {
        path: "/",
        element: <AppLayout />,
        errorElement: <ErrorComponent />,
        children: [
            {
                path: '/about',
                element: <About />,
                children: [
                    {
                        path: 'profile',
                        element: <Profile />
                    }
                ]
            },
        ]
    }
]);
```

Now, you can access the routes on /about/profile -> you will get Profile component rendering in the outlet of About component.

2. What is createHashRouter and createMemoryRouter?

createHashRouter: This 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". Note: Using hash URLs is not recommended.

createMemoryRouter: 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.

```
const router = createMemoryRouter(routes, {
    initialEntries: ["/", "/events/123"],
    initialIndex: 1,
  });
```

Type Definition:

```
function createMemoryRouter(
  routes: RouteObject[],
  opts?: {
    basename?: string;
    initialEntries?: InitialEntry[];
    initialIndex?: number;
    window?: Window;
  }
): RemixRouter;
```

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

Ans. It follows the below order:

When only single component:

---------- Render Phase

1.Constructor

2.Render

---------- Commit Phase

3.ComponentDidMount

4.Re-render if any state change

5.ComponentDidUpdate

6.Re-render if any state change

7.ComponentWillUnmount

When Parent Child component are present:

---------- Render Phase

1.Parent constructor

2.Parent render

3.Child constructor

4.Child render

---------- Commit Phase

5.Child componentDidMount

6.Parent componentDidMount

When Multiple Child component are present:

---------- Render Phase

1.Parent constructor

2.Parent render

3.Child constructor 1

4.Child render 1

5.Child constructor 2

6.Child render 2

---------- Commit Phase

7.Child componentDidMount 1

8.Child componentDidMount 2

9.Parent componentDidMount

Because above react want to finish the render phase for all the children so after child render 1 it will show child constructor 2/

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

---------- Render Phase

1.Parent constructor

2.Parent render

3.Child constructor

4.Child render

---------- Commit Phase

5.Parent componentDidMount

6.Child componentDidMount

7.Child re renders

why this happens because child have an api call and it needs to fetch the data and update DOM, where fetching is asynchronous

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

4. Why do we use componentDidMount?

componentDidMount gets called after the first render in the react lifecycle and hence its becomes perfect place to call or fetch data from API.(Application Programming Interface).

5. Why do we use componentWillUnmount?

componentWillUnmount gets called just before the component getting unmounted or removed from the DOM. It can be useful for clearing up the data similar to how we used to clear the setInterval in the Profile component.

```
import React from "react";

export default class Clock extends React.Component {
   constructor(props) {
      console.log("Clock", "constructor");
      super(props);
      this.state = {
         date: new Date()
      };
   }
   tick() {
      this.setState({
         date: new Date()
      });
   }
   // These methods are called "lifecycle hooks".
   componentDidMount() {
      console.log("Clock", "componentDidMount");
      this.timerID = setInterval(() => {
         this.tick();
      }, 1000);
   }
   // These methods are called "lifecycle hooks".
   componentWillUnmount() {
      console.log("Clock", "componentWillUnmount");
      clearInterval(this.timerID);
   }
   render() {
      return (
            <div>It is {this.state.date.toLocaleTimeString()}.</div>
      );
   }
}
```

6. Why do we use super(props) in constructor?

super() will call the constructor of its parent class. This is required when you need to access some variables from the parent class. In React, when you call super with props, React will make props available across the component through this.props. This is how our custom component gets props variable inherited from React.Component which we are extending from React library.

7. Why should we don't have the callback function of useEffect async?

React's useEffect hook expects a cleanup function returned from it which is called when the component unmounts. Using an async function here will cause a bug as the cleanup function will never get called.
Instead, you should use either immediately invoked functions or named function in such scenarios
IIF

```
useEffect(() => {
  (async () => {
    const users = await fetchUsers();
    setUsers(users);
  })();
    return () => {
         // this now gets called when the component unmounts
    };
}, []);
```

Named function

```
useEffect(() => {
  const getUsers = async () => {
    const users = await fetchUsers();
    setUsers(users);
  };

  getUsers();

  return () => {
    // this now gets called when the component unmounts
  };
}, []);
```