**Hands-on: 14. ReactJS-HOL**

**Introduction**

As React applications grow in complexity, managing state across deeply nested components can become challenging. The React Context API solves this problem by providing a way to share data globally without having to manually pass props through every level. Additionally, React Router enables navigation between components in single-page applications. This section covers the Context API, how to use createContext(), and different types of Router components.

1. **Need and Benefits of React Context API**

* Why React Context API is Needed:
* In large applications, "prop drilling" becomes a problem—passing data from parent to child through multiple intermediate components.
* Components that don’t directly need the data still have to forward it down the tree.
* Benefits:
* Global State Management: Share state like themes, authentication, user data across components.
* Cleaner Code: Reduces the need for props chaining.
* Improved Readability: Logic is centralized, not scattered across deeply nested components.
* Reusable: Multiple components can consume the same context.

1. **Working with createContext()**

The createContext() function is used to create a context object.

* Basic Steps:

1. Create the Context:

import React, { createContext } from 'react';

const ThemeContext = createContext();

1. Create a Provider:

function App() {

return (

<ThemeContext.Provider value="dark">

<Toolbar />

</ThemeContext.Provider>

);

}

1. Consume the Context:

import { useContext } from 'react';

function ThemedButton() {

const theme = useContext(ThemeContext);

return <button className={theme}>Click Me</button>;

}

* Key Concepts:
* Provider: Wraps around the components that need access to context.
* value: The data you want to make available to children.
* useContext(): A hook used to access the context inside functional components.

1. **Types of Router Components (React Router v6+)**

React Router is used for navigating between components in a React application without full page reloads. It offers various components for different routing needs.

1. BrowserRouter

* Uses the HTML5 history API.
* Most commonly used in web apps.

import { BrowserRouter } from 'react-router-dom';

2. HashRouter

* Uses the URL hash (e.g., #) to simulate routing.
* Useful when the server doesn't support deep links.

import { HashRouter } from 'react-router-dom';

3. MemoryRouter

* Stores the history in memory.
* Ideal for testing or non-browser environments like React Native.

import { MemoryRouter } from 'react-router-dom';

4. StaticRouter

* Used for server-side rendering (SSR).
* Doesn’t change the URL or manage history.

import { StaticRouter } from 'react-router-dom/server';

5. NativeRouter *(React Native only)*

* Used with React Native for mobile routing (via react-router-native).

**Conclusion**

The React Context API is a powerful solution for managing shared data without prop drilling, making your application more scalable and maintainable. Using createContext() and useContext(), developers can efficiently pass data to deeply nested components. On the other hand, React Router provides a smooth navigation experience in single-page applications, with multiple router components suited for different platforms and use cases. Together, these tools enhance both the functionality and structure of React apps.