**Hands-on: 13. ReactJS-HOL**

**Introduction**

In modern React applications, rendering dynamic content is a core feature. React allows developers to display different components based on conditions, render multiple components together, and efficiently display lists using keys and the map() function. Proper use of these techniques ensures better performance and more maintainable code. This section explores all the essential concepts related to conditional rendering, lists, and keys in React.

1. **Various Ways of Conditional Rendering**

React supports multiple techniques for rendering content based on conditions:

* If-else statements

if (isLoggedIn) {

return <Dashboard />;

} else {

return <Login />;

}

* Ternary operator

return isDarkMode ? <DarkTheme /> : <LightTheme />;

* Logical AND (&&) operator

{isVisible && <Tooltip />}

* Immediately Invoked Function Expression (IIFE)

{(() => {

if (status === 'error') return <Error />;

if (status === 'loading') return <Loader />;

return <Content />;

})()}

* Switch statement inside a function

function renderView(view) {

switch (view) {

case 'home': return <Home />;

case 'about': return <About />;

default: return <NotFound />;

}

1. **Rendering Multiple Components**

You can render multiple components by grouping them using:

1. React Fragments (<> </> or <React.Fragment>):

return (

<>

<Header />

<MainContent />

<Footer />

</>

);

2. Array of components:

const components = [<Header />, <Content />, <Footer />];

return <div>{components}</div>;

React allows grouping multiple elements without adding extra nodes to the DOM using fragments.

1. **Define List Component**

A List component in React is a reusable component used to display a list of items. It typically uses the map() function to iterate over an array and render elements.

* Example:

function NameList(props) {

return (

<ul>

{props.names.map((name) => <li key={name}>{name}</li>)}

</ul>

);

}

Here, NameList is a component that receives an array of names via props and displays them using <li> elements.

1. **Keys in React Applications**

Keys are special string attributes used to identify elements uniquely among their siblings in a list. They help React optimize rendering and reconciliation.

* Why keys matter:
* Help React detect which items have changed, been added, or removed.
* Improve rendering performance and prevent bugs.
* Example:

const items = ['Apple', 'Banana', 'Cherry'];

const listItems = items.map((item) => <li key={item}>{item}</li>);

Never use array index as a key unless the list is static and will never change.

1. **Extracting Components with Keys**

When building reusable components from a list, ensure the key is passed to the outermost element returned by the component.

Example:

function ListItem(props) {

return <li>{props.value}</li>;

}

function NumberList(props) {

return (

<ul>

{props.numbers.map((num) => (

<ListItem key={num.toString()} value={num} />

))}

</ul>

);

}

Here, ListItem is extracted and each instance is given a unique key in the parent component (NumberList).

1. **React Map and map() Function**

In React, the map() function is commonly used to render lists dynamically by transforming an array of data into React elements.

Example:

const fruits = ['Apple', 'Orange', 'Mango'];

const fruitList = fruits.map((fruit) => <li key={fruit}>{fruit}</li>);

* Features:
* Creates a new array of JSX elements.
* Efficient way to dynamically render components.
* Can be used inside any component's return().

**Conclusion**

Mastering conditional rendering, list rendering, and the use of keys is essential for building dynamic and efficient React applications. Whether it’s showing different content based on state or displaying hundreds of items using the map() function, React provides a flexible and powerful framework for UI logic. Keys ensure optimal performance, while component extraction encourages reusable, readable code. With these tools, developers can create scalable and responsive interfaces.