**WEEK-7 REACT HO-13**

**1. Various Ways of Conditional Rendering in React**

React lets you choose what to display based on state or props. Common patterns:

* **if / element variable** (decide before return):
* let content;
* if (isLoggedIn) {
* content = <UserDashboard />;
* } else {
* content = <GuestView />;
* }
* return <div>{content}</div>;
* **Ternary operator** (inline two-way choice):
* {isLoggedIn ? <UserDashboard /> : <LoginPrompt />}
* **Logical AND** (render only if truthy):
* {showBanner && <Banner />}
* **Early return inside a component** (prevent rendering):
* function Promo({ active }) {
* if (!active) return null;
* return <div>Special offer!</div>;
* }
* **switch / chained conditionals** (multiple exclusive cases):
* switch (view) {
* case 'books':
* return <BookDetails />;
* case 'blogs':
* return <BlogDetails />;
* default:
* return <DefaultView />;
* }
* **Dynamic component selection**:
* const Component = isAdmin ? AdminPanel : UserPanel;
* return <Component />;

**2. How to Render Multiple Components**

You can render more than one component in several ways:

* **Multiple JSX children**:
* return (
* <div>
* <Header />
* <MainContent />
* <Footer />
* </div>
* );
* **Array of components** (each needs a key if siblings):
* return [<First key="1" />, <Second key="2" />, <Third key="3" />];
* **Composition**: pass components as props or children:
* function Layout({ sidebar, content }) {
* return (
* <div>
* <aside>{sidebar}</aside>
* <main>{content}</main>
* </div>
* );
* }
* <Layout sidebar={<Nav />} content={<Article />} />
* **Conditional multiple rendering**:
* return (
* <div>
* {showHeader && <Header />}
* <Content />
* {showFooter && <Footer />}
* </div>
* );

**3. Define List Component**

A list component accepts an array of data and renders a corresponding set of child items, usually via map. Example:

function BookList({ books }) {

return (

<div>

{books.map(book => (

<div key={book.id}>

<h4>{book.title}</h4>

<p>by {book.author}</p>

</div>

))}

</div>

);

}

This abstracts the “list” behavior so you can reuse BookList with different data.

**4. Keys in React Applications**

* **Purpose:** Keys give React a stable identity for each element in a list so it can efficiently diff and update only what changed.
* **Requirements:**
  + Must be unique among siblings (e.g., book.id).
  + Should be stable over time; avoid using array indices if the list can reorder or mutate.
* **Usage example:**
* {items.map(item => (
* <ItemCard key={item.id} item={item} />
* ))}

Without proper keys, React may reuse DOM nodes incorrectly, causing bugs or unexpected behavior.

**5. How to Extract Components with Keys**

When you extract a list item into its own component, the key belongs on the element in the list iteration, not inside the child component’s props (React uses it internally):

function BookItem({ book }) {

return (

<div>

<strong>{book.title}</strong> — {book.author}

</div>

);

}

// In parent:

{books.map(b => (

<BookItem key={b.id} book={b} />

))}

If you wrap the child in another element, place the key on the outermost element returned in the map:

{books.map(b => (

<div key={b.id} className="wrapper">

<BookItem book={b} />

</div>

))}

**Bonus: Combining Concepts**

Example combining conditional rendering, list, and keys:

function FeaturedBooks({ books, showOnlyFeatured }) {

const filtered = showOnlyFeatured ? books.filter(b => b.featured) : books;

return (

<div>

{filtered.length === 0 ? (

<p>No books to show.</p>

) : (

filtered.map(b => <BookItem key={b.id} book={b} />)

)}

</div>

);

}

Anshika Srivastava  
superset id: 6387029