**LHLAPI**

**LIST & KEYS: -**

1. **How do you render a list of items in React? Why is it important to use keys when rendering lists?**

* **How to Render a List of Items in React:**

1. **Prepare the Data**: Use an array to store the list of items you want to display.
2. **Iterate Over the Array**: Use the. map() method to iterate over the array.
3. **Return JSX Elements**: For each item in the array, return a JSX element representing how you want it displayed.

* **Why are Keys Important?**

1. **Efficient Updates**:
   * React uses keys to identify which items in the list have changed, been added, or removed.
   * This helps React optimize rendering by updating only the elements that have changed, instead of re-rendering the entire list.
2. **Avoid Unexpected Behaviour**:
   * Without keys, React may incorrectly reuse elements during updates, causing bugs such as:
     + Losing input focus.
     + Incorrect animations.
     + Retaining state of a wrong component.
3. **Console Warnings**:
   * If you don’t provide a key, React logs a warning:  
     "Warning: Each child in a list should have a unique "key" prop."
4. **What are keys in React, and what happens if you do not provide a unique key?**
   * Keys in React are special attributes used to uniquely identify elements within a list of items. They help React track and manage changes to the DOM efficiently when the list is updated, such as during adding, removing, or reordering items.
   * Keys are typically unique values, like IDs or other identifiers, that remain stable across renders.

* **If You Do Not Provide a Unique Key?**

1. **Performance Issues**:
   * React cannot efficiently determine which elements have changed, leading to unnecessary re-renders of the entire list.
   * For example, if an item is removed, React might re-render the whole list instead of just removing the relevant DOM node.
2. **Unexpected UI Behaviour**:
   * React may reuse DOM elements incorrectly, leading to bugs. For instance:
     + Inputs might lose focus.
     + Animation effects might break.

**HOOKS: -**

1. **What are React hooks? How do useState() and useEffect() hooks work in functional components?**

* React Hooks are special functions that let you "hook into" React's features (e.g., state and lifecycle methods) in functional components.
* Introduced in React 16.8, they allow you to manage state, handle side effects, and perform other component-level operations without using class components.
* **How useState() Works: -**
* The useState() hook allows you to add state to a functional component. It returns:

1. The current state value.
2. A function to update the state.

**Syntax** = const [state, setState] = useState(initialState);

* **How useEffect() Works**
* The useEffect() hook allows you to handle side effects in functional components. Side effects include:
* Data fetching.
* Subscriptions.
* Manual DOM manipulations.

**Syntax** = useEffect(() => {

// Side effect logic

return () => {

// Cleanup (optional)

};

}, [dependencies]);

1. **What problems did hooks solve in React development? Why are hooks considered an important addition to React?**

* **Problems Solved by Hooks in React Development**

1. **Complex State Logic in Class Components**:
   * Managing state logic across multiple lifecycle methods (componentDidMount, componentDidUpdate, componentWillUnmount) made components harder to read and maintain.
   * Hooks like useState and useReducer simplify state management within functional components.
2. **Code Reusability**:
   * Sharing logic between components in class-based systems required higher-order components (HOCs) or render props, which could lead to "wrapper hell."
3. **Better Readability**:
   * Functional components with hooks are generally more concise and easier to understand than class components with lifecycle methods.

* **Why Hooks are Considered an Important Addition to React**

1. **Functional Component Empowerment**:
   * Before hooks, functional components were stateless and used primarily for rendering UI. Hooks transformed functional components into fully capable components that can manage state, side effects, and context.
2. **Simplified Codebase**:
   * Hooks eliminate the need for verbose class components and lifecycle methods, leading to cleaner, more maintainable code.
3. **What is useReducer ? How we use in react app?**

* The useReducer hook is an alternative to useState for managing complex state logic in React applications. It is particularly useful when:

1. The state depends on multiple values or actions.
2. The state transitions are complex or interrelated.
3. You want to organize state logic in a more predictable, centralized way.

* **How useReducer Works: -**

1. Define a **reducer function**:
   * A pure function that takes the current state and an action, then returns the updated state.
2. **What is the purpose of useCallback & useMemo Hooks?**

* useCallback and useMemo are React hooks designed to optimize performance by preventing unnecessary re-renders or computations in your React applications.
* While they have different purposes, they are often used together for performance optimization.
* **useCallback Hook**

**Purpose:**

* The useCallback hook is used to **memoize a function** so that it is not re-created on every render.
* This is especially useful when passing functions as props to child components, preventing those child components from unnecessarily re-rendering.
* **useMemo Hook**
* **Purpose:**
* The useMemo hook is used to **memoize the result of a computation** so that it is not recalculated on every render. It is especially useful for expensive or intensive calculations that don't need to run every time the component re-renders.

1. **What’s the Difference between the useCallback & useMemo Hooks?**

* **UseCallback: -**
* Memoizes a function to avoid re-creation.
* Returns a memoized function reference.
* revent child components from re-rendering due to new function references.
* Prevent child components from re-rendering due to new function references.
* To prevent unnecessary re-creation of functions (e.g., event handlers or props passed to children).
* **UseMemo: -**
* Memoizes the result of an expensive computation.
* Returns a memoized value (result of computation).
* To optimize expensive computations or derived state.
* Recomputes the value only when dependencies change.

1. **What is useRef ? How to work in react app?**

* **useRef is a React hook used to:**

1. **Access DOM elements directly** (e.g., focus an input field, scroll, or modify element properties).
2. **Store mutable values** that persist across renders (e.g., store timers or counters) without causing re-renders.

* **How useRef Works**

1. Returns a **mutable object**: {current: initialValue }.
2. The current property can be updated without triggering re-renders.