AeroAspire - SDE Intern
Gokul Krishna S

Week 2 – Day 3 (October 1)

Task/Assignment :

- TaskCard component;
- Render list of dummy tasks via props;

Questions/Reflections:

1. Explain how props are passed from parent to child;what happens if props change?
    - Props in React are used to pass data and functionality from a parent component to its child components, and they play a critical role in how React apps manage and update their UIs. Here's a clear, human-friendly explanation based entirely on reliable documentation sources.

      Passing Props from Parent to Child :
    - Props are like function parameters for components—they let a parent component send information or behavior (like event handlers) down to its children. In JSX, props are inserted within a component tag as attributes:

      <MyButton count={count} onClick={handleClick} />

    - Here, count and onClick are props being passed from the parent (MyApp) to the child (MyButton). Inside MyButton, these props are accessed as function arguments:

      function MyButton({ count, onClick }) {
      return (
      <button onClick={onClick}>Clicked {count}
      times</button>
      ); }

- Whenever the parent changes the value of a prop—for example, if a state variable in the parent (like count) updates—React will re-render the child component(s) with the new prop values.

If Props Change: What Happens Next?

- When the parent updates a prop (say, after a button click increases count), React automatically re-invokes the child's function and returns a fresh render with the updated data. Old prop values are replaced by the new ones. The component's display updates to reflect the latest prop values, making the UI stay in sync with data.

2. What is the virtual DOM in React and how does re-render happen when props change?
    - React uses a "virtual DOM"—an in-memory representation of the actual DOM. When props change, React doesn't immediately touch the real DOM. Instead:
    - React re-runs the affected component's function.
    - The new JSX is converted to a virtual DOM tree.
    - React compares ("diffs") this new virtual DOM with the previous one to spot what's changed.
    - Only the differences are applied to the real DOM, making updates efficient.
    - This process makes React super fast: it avoids full page refreshes and instead updates only what's needed.

3. How to avoid unnecessary re-renders?
    - Frequent, unnecessary re-renders can slow down an app. Here are effective strategies to keep renders efficient:
    - Only update state when needed: Avoid setting state to the same value—if nothing changed, don't update.
    - Use React.memo: This higher-order component memorizes the result of function components. If props haven't changed, React skips re-rendering the child.
    - Use stable functions/objects as props: Changing functions/objects (creating new ones each render) will trigger

re-renders. Use useCallback or useMemo to avoid this if needed.

- Split components wisely: Smaller, focused components re-render less often if props don’t change.