React Take-Home Assignment

1. Conceptual Questions

A) React Hooks simplify component logic by enabling state, side effects, and other features in functionalcomponents
without using classes. They enhance code readability, reusability, and maintainability through cleaner syntax and custom hooks.
React Hooks improve component logic by allowing functional components to manage state and side effects without using class syntax,
resulting in cleaner, simpler code. They eliminate the need for `this`, enable better code reuse through custom hooks, allow related
Logic to be grouped together,and make components easier to read, test, and maintain.

B) There are several methods used to optimize rendering performance in a React app, especially when dealing with large lists such as

. List Virtualization – displays only the things that are visible, greatly reducing the number of DOM elements shown.
. Memoization with React.memo – Prevents unnecessary re-renders of list items when props remain unchanged.
. Using useCallback and useMemo – increases efficiency by avoiding the need to recreate values and functions on each render.

c) My preferred approach to managing form state and validation in React are ,

. Controlled Components for Simple Forms :UseState ensures real-time synchronisation between the form and component state
for tiny forms by managing the value of each input.
. React Hook Form for Complex Forms :React Hook Form can be used for larger forms because it is lightweight,
minimises re-renders, and makes state management easier.
. Clean and Scalable Code : keeps the code organized, improves performance, and ensures better scalability and
maintainability for real-world applications.

---

3. Debugging Exercise

A) . useState Import is missing :Here useState is used ,but not imported from react, which will cause a runtime error.
Include -import React, { useState } from 'react';

. Missing key prop in .map() : Each child in a list should have a unique key prop. React uses this for efficient re-rendering
Include -unique key like job.id

B) Improved Code

```
import React, { useState } from 'react';
function JobList({ jobs }) {
const [search, setSearch] = useState('');
return (
  <div>
    <input value={search} onChange={e => setSearch(e.target.value)} />
    <ul>
     {jobs.map(job => (
      <li key={job.id}>{job.title} at {job.company}</li>
    ))}
    </ul>
  </div>
 );
}
```

---

4. Performance Scenario

A) . Pagination or Infinite Scrolling : Reduce the initial rendering load by showing a small number of items
(for example, 20 per page) and loading more as the user scrolls or navigates.
. Code Splitting and Lazy Loading Components : Use React.lazy and Suspense to load job-related components only
needed. This reduces the initial bundle size and speeds up the initial load.

B) We can avoid unnecessary re-renders in a JobCard component in the following 3 steps:

. Use React.memo :wrapping the JobCard component with React.memo to ensure it only re-renders when its props change.
. Memoize event handlers with useCallback : using useCallback for handlers like onClick, especially when passed down
to child components.
. Avoid inline objects and functions as props : by defining them outside the render or memoizing them.