React Take-Home Assignment

1. Conceptual Questions

a. What are React hooks, and how do they improve component logic compared to class-based components?

answer: React hooks are special functions like useState and useEffect that let you use state and side effects in functional components. They make code easier to write and understand compared to class components. They also make it easier to reuse logic across components.

b. Explain how you would optimize rendering performance in a React app with a long list (e.g., hundreds of job postings).

answer: If I had to render hundreds of job postings, I would use list virtualization so that only the jobs currently visible on the screen are rendered. This way, the app doesn't try to load all the items at once, which makes it faster. I'd also give each job a proper unique key so React can handle updates efficiently. This keeps the list smooth and easy to scroll even when it's very long.

c. Describe your preferred approach to managing form state and validation in React.

answer: I usually use controlled inputs with useState for simple forms. For bigger forms, I would use something like useReducer. For validation, I like keeping rules simple and showing errors near each input. I check inputs as the user types or on submit, so the form feels responsive and clear.

3. Debugging Exercise

a. What is wrong or could be improved?

1. `useState` is used but not imported from React.
2. Each list item inside `.map()` should have a unique `key` prop to avoid React warnings.

b. Provide your improved code and explain your changes.

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>
  );
}
```

Explanation:

- I imported useState from React so it works.
- I added key={job.id} inside the <li> so React can render the list efficiently without warnings.

4. Performance Scenario

a. Two techniques to improve performance of a job list with 500+ items:

One technique is called **Pagination**, where the list is broken into smaller pages so the user only loads and views a limited number of jobs at a time. Another technique is **List Virtualization** where only the jobs that are currently visible on the screen are rendered, while the rest are loaded as the user scrolls.

b. How to avoid unnecessary re-renders in a job card component:

To avoid extra re-renders, I would use **React.memo**, which ensures the job card only updates when its data changes. I would also give each job card a **stable key** (like a unique `id` that does not change) so React can correctly track items between renders. A stable key helps React know which card stayed the same, so it doesn't redraw it again unnecessarily

Notes on Improvements / Future Work

- **Error Handling:** Improve error messages to be more specific
- **Persistent Saved Jobs:** Store saved jobs in localStorage or a backend so they remain after refresh.
- **UI Enhancements:** Add sorting, filtering by location or company, and improve the job card design.
- **Scalability:** Use pagination or list virtualization to handle very large job lists efficiently.