Conceptual Questions

a. What are React hooks?

React hooks are special functions that let you add features like state (memory) and effects (actions) to your simple function components.

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

To optimize rendering performance in a React app with a long list, you can use techniques such as:

- Virtualization: Use libraries like react-window or react-virtualized to render only the visible items in the list.

- Memoization: Use React.memo to prevent unnecessary re-renders of components that do not change.

- Key Prop: Ensure each list item has a unique key to help React identify which items have changed.

- Lazy Loading: Load data in chunks as the user scrolls down the list.

- Avoid Inline Functions: Define functions outside of the render method to prevent re-creation on each render.

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

My preferred approach to managing form state and validation in React is to use controlled components along with a state management library like Formik or React Hook Form. These libraries provide built-in support for handling form state, validation, and error messages. I also use Yup for schema-based validation, which allows me to define validation rules in a clear and reusable way. This approach helps keep the form logic organized and makes it easier to manage complex forms with multiple fields.

Debugging Code:

I created a new filteredJobs array by using the .filter() method on the original jobs prop. This new array only contains jobs that match the current search term, making the search functional. Additionally, I added a unique key={job.id} prop to each <li> element inside the .map() loop to satisfy React's requirement for list rendering, which improves performance and stability.

```
function JobList({ jobs }) {
  const [search, setSearch] = useState('');

  const filteredJobs = jobs.filter(job =>
    job.title.toLowerCase().includes(search.toLowerCase()) ||
    job.company.toLowerCase().includes(search.toLowerCase())
  );

  return (
    <div>
      <input
        placeholder="Search jobs..."
        value={search}
        onChange={e => setSearch(e.target.value)}
      />
      <ul>
        {filteredJobs.map(job => (
          <li key={job.id}>
            {job.title} at {job.company}
          </li>
        ))}
      </ul>
    </div>
  );
}
```

4. Performance Scenario

a. Two ways to make a long list (500+ items) faster:

Virtualization: Only show the items currently on the screen. As you scroll, it loads more, so the app never feels slow.

Pagination: Break the long list into small pages (e.g., Page 1, Page 2). This way, you only load a few items at a time.

b. How to stop a job card from re-rendering unnecessarily:

Use React.memo: Wrap your component in React.memo to tell it, "Don't re-render if the information you receive is exactly the same as before."

Use useCallback: Use this for any functions you pass to your component. It prevents the function from being recreated on every render, which helps React.memo do its job.