Case Study 3: Task Management Application with Drag and Drop

1. How would you implement drag-and-drop functionality in a React component?

A. To add drag-and-drop, we can use libraries like react-beautiful-dnd. It makes things easier by giving you components to handle the dragging and dropping. we wrap your task list in a component that lets you drag items around and another one that lets you drop them in the right place.

2. Describe how to manage the state of tasks and handle CRUD operations.

A. We can use React's state management (e.g., useState or useReducer) to store the list of tasks. CRUD operations (Create, Read, Update, Delete) can be managed by updating the state. For instance, adding a new task appends to the task array, while drag-and-drop can reorder the task list within the state.

3. How can you use local storage to persist the state of the tasks across page reloads?

A. We can use localStorage to store tasks in the browser. we can store the tasks when changes occur (using useEffect for example) and retrieve them on page load. This ensures the state is saved and remains across page reloads.

4. Explain the steps to ensure smooth and responsive drag-and-drop interactions.

A. Minimize re-renders by managing state efficiently, and update the UI only when necessary. We can draggable components lightweight to ensure smooth interaction. Use performance optimizations like React.memo where appropriate.

5. What are the best practices for handling large lists of tasks in terms of performance?

A. **Virtualization:** Use libraries like react-window or react-virtualized to only render visible items, reducing memory usage.

Efficient State Updates: Update only the necessary parts of the task list to avoid expensive re-renders.

Debouncing: If drag operations trigger heavy updates, debounce those updates to avoid performance bottlenecks