Design and Implementation Documentation

This document is an overview of the design and implementation details of the assignment "Inventory and Manufacturing Management System". The primary components involved are:

- InventoryList
- OrderList
- OrderDetails

which manage the items, orders and details of orders respectively.

Technologies / Tools Used:

- React + Next.js (JavaScript)
- TailwindCSS (UI)
- React Hot Toast (Notifications)

Design Choices:

1. State Management - `useState` hook is used in this assignment to manage state of various components. It manages list of items and orders, filter and search states, modal states, loading states, pagination states, and form input.
2. Data Fetching and Mutations - Data mutations i.e. CRUD operations (add, edit, delete) are simulated using `setTimeout` improving user experience with loading indicators.
3. Notifications - I've implemented the toast notifications via the NPM library `react-hot-toast` for displayig success and error messages, it provides immediate feedback to user actions like, adding, updating, or deleting items / orders.
4. Pagination - The inventory and order list pages are paginated from client side, with `currentPage` and `itemsPerPage` state variables.
5. Sorting and Filtering
    a. Orders can be sorted by customer name and item count by clicking on the up and down icons on column headers.
    b. Users can filter items by stock availability and order status by using dropdown menus.
6. Modal windows - Added modal windows for adding, editing, deleting items / orders, which enhances the UX.

Implementations Details:

InventoryList Component

State Variables

- items: Holds the list of inventory items.
- filter: Controls the filter state for stock status.
- newItem: Manages the state of the new item form inputs.
- editItem: Holds the item being edited.
- isModalOpen, isDeleteModalOpen: Manage the visibility of add/edit and delete confirmation modals.
- itemToDelete: Stores the ID of the item to be deleted.
- loading: Indicates loading state during data mutations.
- currentPage, searchTerm: Manage pagination and search functionality.

Key Functions

- handleDelete(id): Initiates the delete process for an item.
- confirmDelete(): Confirms and processes the deletion of an item.
- handleFilterChange(e): Updates the filter state based on user selection.
- handleSearchChange(e): Updates the search term state.
- handleAddItem(e): Handles the addition of a new item.
- handleEditItem(id): Initiates the edit process for an item.
- handleSaveEdit(e): Saves the edited item.
- paginate(pageNumber): Updates the current page for pagination.

Array and Object Manipulations

- Adding Items: Uses setItems([...items, newItemData]) to add a new item to the list.
- Editing Items: Uses setItems(items.map(item => item.id === editItem.id ? editItem : item)) to update an item.
- Deleting Items: Uses setItems(items.filter(item => item.id !== itemToDelete)) to remove an item.

OrderList Component

State Variables

- orders: Holds the list of orders.
- filter: Controls the filter state for order status.
- sortKey, isAscending: Manage sorting of orders.
- isDeleteModalOpen: Manages the visibility of the delete confirmation modal.
- orderToDelete: Stores the ID of the order to be deleted.
- loading: Indicates loading state during data mutations.
- currentPage, searchTerm: Manage pagination and search functionality.

Key Functions

- handleDelete(id): Initiates the delete process for an order.
- confirmDelete(): Confirms and processes the deletion of an order.
- handleFilterChange(e): Updates the filter state based on user selection.
- handleSearchChange(e): Updates the search term state.
- handleSort(key): Updates the sort key and toggles sort order.
- paginate(pageNumber): Updates the current page for pagination.

Array and Object Manipulations

- Deleting Orders: Uses setOrders(orders.filter(order => order.id !== orderToDelete)) to remove an order.
- Sorting Orders: Uses sortedOrders.sort((a, b) => ...) to sort orders based on customer name or item count.

OrderDetails Component

State Variables

- order: Holds the detailed information of a specific order.
- loading: Indicates loading state while fetching order details.
- error: Holds any error messages encountered during data fetching.
- editItem: Manages the state of an item being edited within the order.
- isEditModalOpen: Manages the visibility of the edit modal.
- isDeleteModalOpen: Manages the visibility of the delete confirmation modal.
- itemToDelete: Stores the ID of the item within the order to be deleted.

Key Functions

- fetchOrderDetails(orderId): Fetches the detailed information of an order by its ID.
- handleEditItem(itemId): Initiates the edit process for an item within the order.
- handleSaveEdit(e): Saves the edited item within the order.
- handleDeleteItem(itemId): Initiates the delete process for an item within the order.
- confirmDeleteItem(): Confirms and processes the deletion of an item within the order.

Array and Object Manipulations

- Fetching Order Details: Uses useEffect to fetch and set order details based on the order ID.
- Editing Order Items: Uses setOrder({ ...order, items: order.items.map(item => item.id === editItem.id ? editItem : item) }) to update an item within the order.
- Deleting Order Items: Uses setOrder({ ...order, items: order.items.filter(item => item.id !== itemToDelete) }) to remove an item within the order

*Caution Points:

- The default colour theme is dark. It may vary according to your system mode (dark or white mode). Executing this project in the dark mode system settings is recommended.
- You might see the error "*Unhandled Runtime Error: Hydration failed because the initial UI does not match what was rendered on the server*", it occurs when the HTML structure rendered by the server differs from what React expects on the client side during hydration. I attempted to resolve the issue, but it resulted in unintended changes to the current UI.
- I've added demo data in order to test the pagination, and the data provided by you has been commented out in the *data.js* file.

Conclusion

This design and implementation choices prioritize user experience, efficient state management, and clear feedback mechanisms. By utilizing React's state management hooks and providing visual feedback through modals and toast notifications, the system creates a smooth and responsive user interface for managing inventory items, orders, and detailed order information.