Architectural Design Record (ADR)

Title: Client-Side Grid Filtering, Sorting, and Pagination with Backend Record Limit Enforcement
Date: 2025-06-16
Status: Accepted

Context

This decision addresses the approach to managing **grid-level data interactions** — specifically filtering, sorting, and pagination — for data-driven UI components.

Two architectural options were considered for handling these features:

1. **Client-Side Handling** – Load the dataset once from the API and allow all interactions (filtering, sorting, pagination) to occur entirely in the browser.
2. **Server-Side Handling** – Rely on backend API endpoints to execute filtering, sorting, and pagination for every user interaction.

The system in question needs to handle datasets that may be large in some cases, but are typically manageable in size (<500 records after filtering).

Decision

The architecture will implement **client-side filtering, sorting, and pagination** for grid components.

To support this:

- The UI will include **filter inputs** and a **"Search" button** that users must use to initiate data retrieval.
- Upon clicking "Search", the frontend will send a request to the API including any provided filter criteria.
- The API will return a dataset (up to 500 records), which will be loaded into the client.
- All grid operations (filtering, sorting, pagination) will then be handled **entirely client-side** using the retrieved data.

In addition:

- The **API will implement standardized filtering and sorting behavior** for all relevant endpoints. This ensures consistency and reusability of query logic across the application.
- Every API endpoint will support:
    - **Sorting** via query parameters in ascending/descending order.
    - **Filtering** on text fields (equality) and numeric fields (equality, greater than, less than, and range).

This approach ensures that users have a mechanism to narrow their result sets before the client attempts to render or operate on the data, helping avoid hitting the 500-record API response limit, while also providing a robust and extensible backend interface.

Reasoning

Performance and User Experience

- Interactions like filtering and sorting are executed instantly in the browser, eliminating round trips to the server and improving responsiveness.
- For most use cases, the amount of data returned will remain within a tolerable size (500 records or fewer).

Simplicity

- Reduces complexity in the frontend-to-backend interaction model.
- Decreases backend logic for grid state handling.
- Minimizes API call frequency, especially during repeated user interactions like re-sorting and incremental filtering.

API Record Limit Handling

- The REST API will enforce a **default maximum return size of 500 records** per request.
- If a client request results in more than 500 records being returned:
  - The API will **truncate the dataset to the first 500 records**.
  - A **warning response** will be included, indicating that the full dataset was not returned and suggesting the need for more precise filters.

REST API Capabilities

Even though client-side grid operations are being used, the REST API will still support **standardized query options** to enable reusable and predictable behavior across endpoints:

Filtering

- **Text fields**: Supports equality match (`eq`)
- **Numeric fields**: Supports:
  - Equality (`eq`)
  - Greater than (`gt`)
  - Less than (`lt`)
  - Range filtering (e.g., `min`/`max` parameters or range object)

Sorting

- Supports sorting on eligible fields in both **ascending** and **descending** order (e.g., `?sort=name.asc`, `?sort=createdAt.desc`)

Pagination

- Optional pagination parameters (`limit`, `offset`) will be accepted but capped at a **maximum of 500 records per response**.

These capabilities will be beneficial for more advanced use cases or if a future move to full server-side grid interaction is needed.

Consequences

Advantages

- Fast and fluid user interactions without needing repeated API calls.
- Lower backend load due to fewer calls.
- Enables consistent client-side logic and simpler debugging.
- Provides foundational backend filter/sort capabilities that can be reused elsewhere or scaled later.

Trade-Offs

- Large unfiltered datasets may result in incomplete results due to the 500-record API limit.
- Client memory usage may increase with larger datasets.
- Users must apply meaningful filters to ensure they are working with the full intended dataset.

Framework & Stack Notes

The following technologies are being used to implement this architecture:

- **Frontend**: React with Ant Design Table for UI grid components.
- **Backend**: RESTful API implemented with NestJS.
- **Communication**: JSON-based REST over HTTPS, with standard query parameters for filtering, sorting, and pagination.