Architectural Design Record (ADR)

Title: Client-Side Grid Filtering, Sorting, and Pagination with Backend Offset-Based Pagination
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 after user-driven 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 **page of data** (default size of 500 records or fewer) along with **offset-based pagination metadata**.
- 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).

If the total number of matching records exceeds the initial page size (default limit = 500), the API will not return an error. Instead, it will return a `nextPageUrl` that can be used to fetch additional pages of data.

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.
- Users control when and how data is retrieved, reducing backend load and improving perceived speed.

Simplicity

- Reduces complexity in the frontend-to-backend interaction model.
- Decreases backend logic for grid state handling.
- Encourages standardization of query patterns across endpoints.

Scalable Pagination

- Using `offset` and `limit` enables scalable data retrieval while avoiding hard failures due to record caps.
- Providing metadata like `nextPageUrl` and `totalRecords` gives clients full control if they need to fetch more data explicitly.

REST API Capabilities

Even though grid operations are handled client-side after data retrieval, the REST API will support consistent and standardized pagination, filtering, and sorting behavior:

Filtering

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

Sorting

- Query parameter format: `?sort=field.asc` or `?sort=field.desc`
- Multiple sort keys may be supported where applicable.

Pagination (Offset-Based)

- Default `limit`: 500
- Supports `limit` and `offset` parameters
- The API response includes:

```
{
  "data": [ /* array of records */ ],
  "offset": 0,
  "limit": 500,
  "totalRecords": 1342,
  "hasMore": true,
```

```
  "nextPageUrl": "/api/items?offset=500&limit=500",
  "previousPageUrl": null,
  "filtersApplied": {
    "status": "active"
  },
  "sort": "createdAt.desc"
}
```