Eliassen - Search Extensions

Summary

The intention of this library is to provide a common means to define endpoints for the searching, filtering, sorting and paging of data. As a mean to define a common interface and allow for faster development it was determined to create a common design pattern for all data query screens.

There was a preference to use existing off-the-shelf protocols and libraries as a means to provide this data querying functionality. Before designing this feature existing protocols were reviewed as such as OData and GraphQL. Various limitations such as complexity in implementation and potential exposure personally identifyable information it was determined an in-house solution was required.

Design Goals and Use Cases

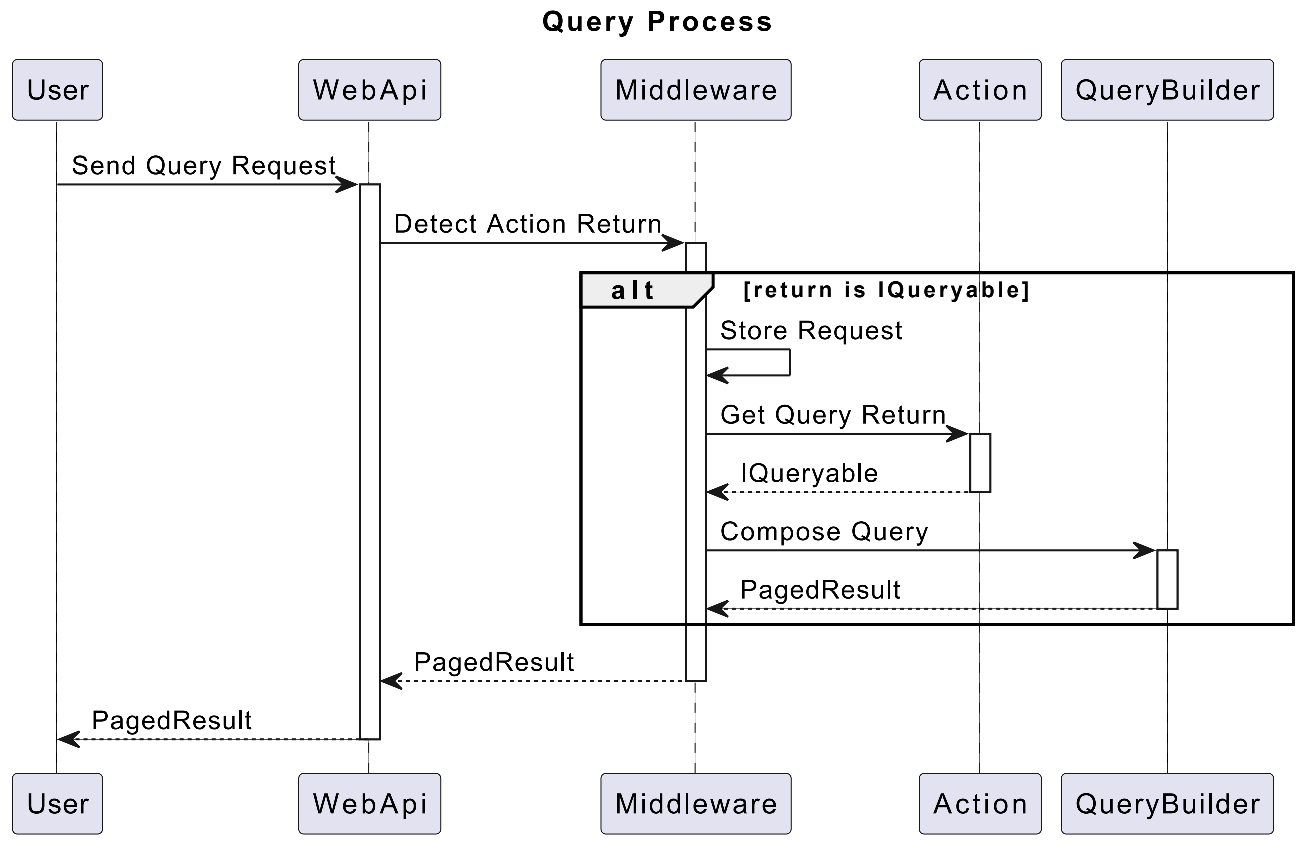
- Create a query model that allows for an easy to follow and reusable pattern
- Data selection patterns
  - The dataset in particular will be provided via an HTTP/RESTful endpoint
  - Part of the URI path may be used to select the named endpoint
  - Search term may be used to select rows from multiple columns within the database
  - Filter rules may be used to select rows based on a particular field within the dataset
    - equal, not equal, wildcard (starts with, ends with, contains)
    - unbounded and bounded ranges
    - set value matching
- Multiple field sorting with the ability to select direction and priority order
- Data paging with the ability to change page size or disable paging to return all rows
- Make it easy to implement for developers

Technical Requirements

- Define Query Request Model
- Define Paged Result Models
- Define Patterns for Filter
- Define Wildcard expressions
- Define Patterns for Sort Order
- Define Patterns for Paging Request
- Extend ASP.Net Middleware to intercept and extends `IQueryable<>` returns from Controller/Actions
- Extend OpenAPI to present notes on available fields for request and response
- Create Expression Tree based framework to convert request models into LINQ queries
- HTTP Request type and formatting is detected from the related `Method` and `Content-Type`

Architecture

HTTP Data Query



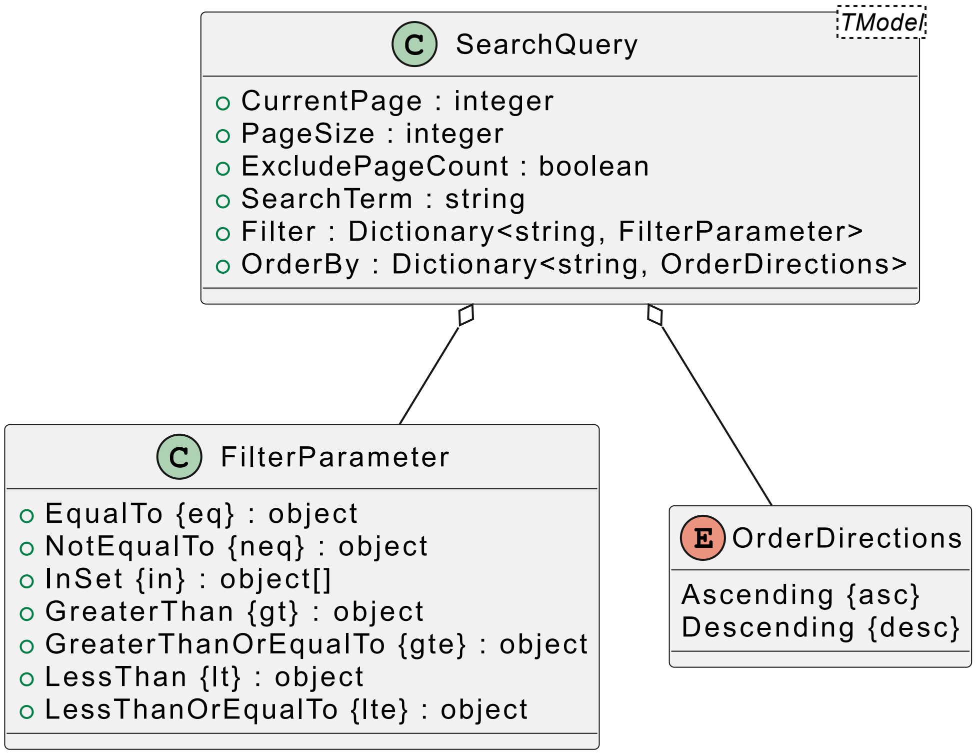
API Template

Query

Models

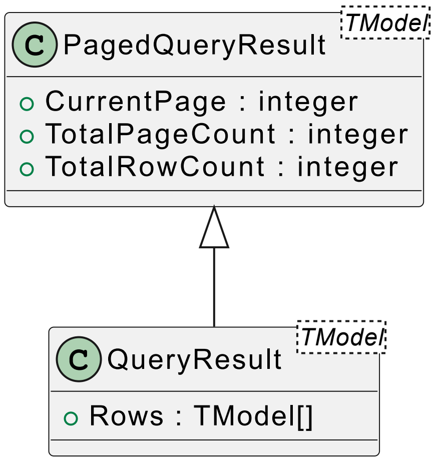
Request

Note: the key values for `Filter` and `OrderBy` are based on fields from the `TModel` type


TModel
C SearchQuery
o CurrentPage : integer
o PageSize : integer
o ExcludePageCount : boolean
o SearchTerm : string
o Filter : Dictionary<string, FilterParameter>
o OrderBy : Dictionary<string, OrderDirections>
C FilterParameter
o EqualTo {eq} : object
o NotEqualTo {neq} : object
o InSet {in} : object[]
o GreaterThan {gt} : object
o GreaterThanOrEqualTo {gte} : object
o LessThan {lt} : object
o LessThanOrEqualTo {lte} : object
E OrderDirections
Ascending {asc}
Descending {desc}

Response

Note: `QueryResult` is the unpaged result while `PagedQueryResult` is only a single page of possible results. `QueryResult` with all matched records will be returns if the `PageSize` is equal to `-1` on the request. Paging will still occur but total counts will not be calculated if `ExcludePageCount` is `true` on the request.



Technical Considerations

Security

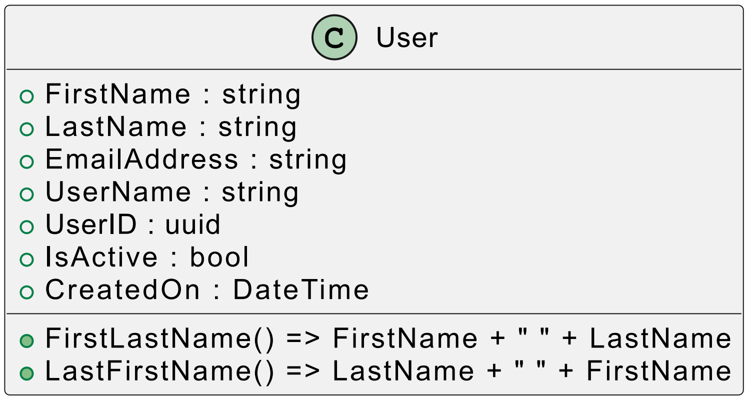
Authentication and authorization follow standard ASP.Net MVC configurations.

Performance

To improve performance and processing queries will be composed by LINQ to the related query provider allowing for data server processing including.

Examples

Data Model



Request - Starts with

Query first page where first name starts with `jim` and sort the results by `LastName " " FirstName`

```
{
  "currentPage": 0,
  "filter": {
    "FirstName": {
      "eq": "jim*"
    }
  },
  "orderBy": {
    "lastFirstName" : "desc"
  }
}
```

Request - Any field Contains

Query first page where any field contains `jim` with the default sort.

```
{
  "currentPage": 0,
  "searchTerm": "*jim*"
}
```