



Handling Complex API Filter Queries in ASP.NET Core



Gowtham K

Aug 07, 2023



17k



3



1



Introduction

A generic composite filter functionality to handle the client-side complex API filter queries is always a necessary component in API development. In this article, I'm going to explain the generic composite filter which I have developed for the ASP.NET Core web API application and published on [GitHub](#).

Composite Filter

If you check the Utility folder from the GitHub repository, you will find two cs files.

1. CompositeFilter.cs
2. RootFilter.cs

RootFilter.cs

It is a model class that we used to deserialize the filter query coming from the client.

```
1 private static Expression public class RootFilter
2 {
3     public List<Filter> Filters { get; set; }
4     public string Logic { get; set; }
5 }
6
7 public class Filter
8 {
9     public string Field { get; set; }
10    public string Operator { get; set; }
11    public object Value { get; set; }
12    public string Logic { get; set; } // This is the nested "logic" property for the filter
13    public List<Filter> Filters { get; set; } // Nested filters array
14 }
```

Assume we have the following filter query for the API from the client.

```
1 https://localhost:44360/api/values?filter={"filters":[{"field":"Name","operator":"contains"}]}
```

The above RootFilter model will be used to deserialize the filter query by using the Newtonsoft.Json library.

```
1 string filter = HttpContext.Request.Query["filter"];
2 if (!string.IsNullOrEmpty(filter))
3 {
4     filterResult = JsonConvert.DeserializeObject<RootFilter>(filter);
5 }
```

The above code will deserialize the filter query from the client request.

CompositeFilter.cs

The composite filter file consists of four methods.

1. BuildFilterExpression

```
1 private static Expression BuildFilterExpression(Filter filter, ParameterExpression parameter)
2 {
3     if (filter.Filters != null && filter.Filters.Any())
4     {
5         if (filter.Logic?.ToLower() == "and")
6         {
7             var andFilters = filter.Filters.Select(f => BuildFilterExpression(f, parameter));
8             return andFilters.Aggregate(Expression.AndAlso);
9         }
10        else if (filter.Logic?.ToLower() == "or")
11        {
12            var orFilters = filter.Filters.Select(f => BuildFilterExpression(f, parameter));
13            return orFilters.Aggregate(Expression.OrElse);
14        }
15    }
16
17    if (filter.Value == null || string.IsNullOrWhiteSpace(filter.Value.ToString()))
18        return null;
19
20    var property = Expression.Property(parameter, filter.Field);
21    var constant = Expression.Constant(filter.Value);
22
23    switch (filter.Operator.ToLower())
24    {
25        case "eq":
26            return Expression.Equal(property, constant);
27        case "neq":
28            return Expression.NotEqual(property, constant);
29        case "lt":
```



```

34         return Expression.GreaterThan(property, constant);
35     case "gte":
36         return Expression.GreaterThanOrEqual(property, constant);
37     case "contains":
38         var containsMethod = typeof(string).GetMethod("Contains", new[] { typeof(string) });
39         return Expression.Call(property, containsMethod, constant);
40     case "startswith":
41         var startswithMethod = typeof(string).GetMethod("StartsWith", new[] { typeof(string) });
42
43         // Convert the constant value to lowercase for case-insensitive comparison
44         var constantLower = Expression.Call(constant, typeof(string).GetMethod("ToLower", new[] { typeof(string) }));
45
46         return Expression.Call(property, startswithMethod, constantLower, Expression.Constant(""));
47
48     // Add more operators as needed...
49     default:
50         throw new ArgumentException($"Unsupported operator: {filter.Operator}");
51     }
52 }

```

This function will build the fundamental filter expression with different operators like eq, neq, lt, lte, gt, gte, contains, and startswith.

2. GetAndFilterExpression

```

1 private static Expression<Func<T, bool>> GetAndFilterExpression(List<Filter> filters)
2 {
3     if (filters == null || !filters.Any())
4         return null;
5
6     var parameter = Expression.Parameter(typeof(T), "x");
7     Expression andExpression = null;
8
9     foreach (var filter in filters)
10    {
11        var filterExpression = BuildFilterExpression(filter, parameter);
12        if (filterExpression != null)
13        {
14            if (andExpression == null)
15            {
16                andExpression = filterExpression;
17            }
18            else
19            {
20                andExpression = Expression.AndAlso(andExpression, filterExpression);
21            }
22        }
23    }
24
25    if (andExpression == null)
26    {
27        // Return default expression that always evaluates to false
28        andExpression = Expression.Constant(false);
29    }
30
31    return Expression.Lambda<Func<T, bool>>(andExpression, parameter);
32 }

```

This function will help to build the filter expression for AND logic with a nested filter option.

3. GetOrFilterExpression

```

1 private static Expression<Func<T, bool>> GetOrFilterExpression(List<Filter> filters)
2 {
3     if (filters == null || !filters.Any())
4         return null;
5
6     var parameter = Expression.Parameter(typeof(T), "x");
7     Expression orExpression = null;
8
9     foreach (var filter in filters)
10    {
11        var filterExpression = BuildFilterExpression(filter, parameter);
12        if (filterExpression != null)
13        {
14            if (orExpression == null)
15            {
16                orExpression = filterExpression;
17            }
18            else
19            {
20                orExpression = Expression.OrElse(orExpression, filterExpression);
21            }
22        }
23    }
24
25    if (orExpression == null)
26    {
27        // Return default expression that always evaluates to false
28        orExpression = Expression.Constant(false);
29    }
30
31    return Expression.Lambda<Func<T, bool>>(orExpression, parameter);
32 }

```

```
29     }
30
31     return Expression.Lambda<Func<T, bool>>(orExpression, parameter);
32 }
```

This function will help to build the filter expression for OR logic with a nested filter option.

4. ApplyFilter

```
1 public static IQueryable<T> ApplyFilter(IQueryable<T> query, RootFilter filter)
2 {
3     if (filter == null || filter.Filters == null || !filter.Filters.Any())
4         return query;
5
6     Expression<Func<T, bool>> compositeFilterExpression = null;
7
8     if (filter.Logic?.ToLower() == "and")
9     {
10         compositeFilterExpression = GetAndFilterExpression(filter.Filters);
11     }
12     else if (filter.Logic?.ToLower() == "or")
13     {
14         compositeFilterExpression = GetOrFilterExpression(filter.Filters);
15     }
16
17     return compositeFilterExpression != null
18         ? query.Where(compositeFilterExpression)
19         : query;
20 }
```

This function is exposed for the call from another file,

```
1 sessionsQuery = CompositeFilter<Product>.ApplyFilter(sessionsQuery, filterResult);
```


Where sessionQuery will be the list of product records.

Here is the complete action written in the API Controller.

```
1 public async Task<IActionResult> Get()
2 {
3     dynamic sessions = new ExpandoObject();
4     string filter = HttpContext.Request.Query["filter"];
5     var filterResult = new RootFilter();
6
7     if (!string.IsNullOrEmpty(filter))
8     {
9         filterResult = JsonConvert.DeserializeObject<RootFilter>(filter);
10    }
11    else
12    {
13        filterResult = new RootFilter();
14    }
15
16    var productList = new List<Product>();
17    for (int i = 0; i < 100; i++)
18    {
19        productList.Add(new Product { ProductID = i, Name = "Product " + i, Price = i });
20    }
21
22    var sessionsQuery = productList.AsQueryable();
23
24    if (filterResult.Filters != null)
25    {
26        sessionsQuery = CompositeFilter<Product>.ApplyFilter(sessionsQuery, filterResult);
27    }
28
29    sessions.records = sessionsQuery.ToList();
30
31    return Ok(sessions.records);
32 }
```

Let's test API with some complex filter queries from Postman.

```
1 https://localhost:44360/api/values?filter={"filters":[{"field":"Name","operator":"cont
```

 postman testing

Added more complexity(logic grouping) in the filter query.

```
1 https://localhost:44360/api/values?filter={"filters":[{"field":"Name","operator":"cont
```

Summary

We have seen how to perform the composite filter in ASP.NET Core API based on the filter queries from the client, and we also see the generic filter functions are also capable of handling complex filter queries.

Happy coding!!!

RECOMMENDED FREE EBOOK

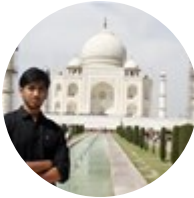


Coding Principles

Download Now!

SIMILAR ARTICLES

- [Action Filters in ASP.NET Core](#)
- [SQL Subqueries: Correlated, Scalar, and EXISTS/NOT EXISTS](#)
- [Handling Concurrency In ASP.NET Core 2.0 Web API](#)
- [Filtering In ASP.NET Core 2.0 Web API](#)
- [ASP.NET Core – Exception Handling](#)



Gowtham K *TOP 100*

58 9.5m 8 MVP 6

Gowtham K is awarded as MVP by Microsoft for his exceptional contribution in Microsoft technologies under the category "Developer Technologies" for the year 2016, 2017 and 2018. He has more than 5 years of expe... [Read more](#)

<https://www.c-sharpcorner.com/members/gowtham-k3>

3

[View All Comments](#)



Type your comment here and press Enter Key (Minimum 10 characters)

