**Base Filter Algoritmasi**

İlkin olaraq client-dan gələn request modelinə uyğun təyin edilmiş filter model təyin edilməli və doldurulmalıdır. Filter model təyin edilən zaman property məlumatları Entity-dəki filter ediləcək property adları ilə **MÜTLƏQ** şəkildə eyni olmalıdır. Əlavə olaraq Filter model daxilində hər bir property-ni xarakterizə edən və “{**propertyName**}**FilterType**” adlandırma standartı tətbiq edilən “**FilterType**” property-lər yaradılmalıdır. “**FilterType**” property yaratmaq üçün eyni adlı Enum yaradılaraq içərisinə Expression yaradılan zaman hansı şərt ilə yaradılmasını təmin edən açar sözlər əlavə edilir (*Default olaraq filter modeldəki filtertype-lara dəyərlər təyin edilir. Əlavə olaraq əgər bu dəyərlər sonradan dəyiştirilmək istəyərsə modelin referansı yaradılan zaman dəyişdirilə bilər.*). Yuxarıda sadalanlara uyğun olaraq **Enum**, **Request** model və ona uyğun **Filter** modellər əks edilmişdir.

**Enum:**

public enum FilterType

{

Equal,

Contains,

NotEqual,

GreaterThan,

LessThan

}

**Request Model:**

public class CasesDangerousRequestModel

{

public string Voen { get; set; } = null;

public string FirmName { get; set; } = null;

public string UnicalCode { get; set; } = null;

public string ManufacturerCompany { get; set; } = null;

public CasesDangerousCountry Country { get; set; } = new CasesDangerousCountry();

public DangerousSituationRequestModel DangerousSituation { get; set; } = new DangerousSituationRequestModel();

}

**Filter Model:**

public class BaseFilterForCDModel

{

public CasesDangerousFilterModel CasesDangerousFilter { get; set; } = null;

public CasesDangerousCountryFilterModel Country { get; set; } = null;

public DangerSituationFilterModel DangerSituation { get; set; } = null;

}

public class CasesDangerousFilterModel

{

public string Voen { get; set; } = null;

public string FirmName { get; set; } = null;

public string UnicalCode { get; set; } = null;

public string ManufacturerCompany { get; set; } = null;

public FilterType VoenFilterType { get; set; } = FilterType.Equal;

public FilterType FirmNameFilterType { get; set; } = FilterType.Contains;

public FilterType UnicalCodeFilterType { get; set; } = FilterType.Equal;

public FilterType ManufacturerCompanyFilterType { get; set; } = FilterType.Contains;

}

Növbəti mərhələdə filter alqoritmasını təmin edən sinif (BaseFilterAlgorithm) və method-lar əlavə edilir. Əlavə edilmiş public olan GenerateFilterExpression adlı method generic tipdədir və Expression yarada bilməsi üçün TEntity tipinində Entity modeli və həmin modeli xarakterizə edən TFilter modeli göndərilməlidir. Doğru məlumatlar göndərildikdən sonra cari method daxilində private olan iki fərqli methoda AddFilterExpressions (*filter modeldəki property adına uyğn olan Entity modelinin property-si üçün Expression yaradır*) və GetFilterType (*Filter property-dəki FilterType-a görə şərtin necə olacağı təyin edilir*) müraciət edir. Method-lar və uyğun kodlar aşağıdakı kimidir.

public static class BaseFilterAlgorithm<TEntity> where TEntity : BaseDAO

{

// Filtr modeli (TFilter) istifadə edərək (TEntity) tipində filtr ifadəsi yaradır.

public static Expression<Func<TEntity, bool>> GenerateFilterExpression<TFilter>(TFilter filterModel)

{

// TEntity tipi üçün parametr ifadəsi yaradılır.

ParameterExpression parameter = Expression.Parameter(typeof(TEntity), "entity");

// Filtr ifadəsinə ilkin olaraq null dəyəri mənimsədilir.

Expression filterExpression = null;

// Filtr modelinə uyğun mövcud field-lər tapılır.

PropertyInfo[] filterProperties = typeof(TFilter).GetProperties()

.Where(p => p.PropertyType != typeof(FilterType))

.ToArray();

// hər bir filed-ə uyğun filtr təyin edilir.

foreach (var property in filterProperties)

{

string propertyName = property.Name;

string propertyValue = property.GetValue(filterModel)?.ToString();

if (!string.IsNullOrEmpty(propertyValue))

{

filterExpression = AddFilterExpressions(parameter, propertyName, propertyValue, filterExpression, filterModel);

}

}

// Filtr ifadəsi null deyilsə lambda ifadəsinə çevreilir.

if (filterExpression != null)

{

Expression<Func<TEntity, bool>> lambda = Expression.Lambda<Func<TEntity, bool>>(filterExpression, parameter);

return lambda;

}

return null;

}

private static FilterType GetFilterType<TFilter>(string propertyName, TFilter filterModel)

{

//Property ilə eyni adla adlandırılmış və propertini xarakterizə edən porperty-ni təyin edir.

PropertyInfo filterTypeProperty = typeof(TFilter).GetProperty($"{propertyName}FilterType");

if (filterTypeProperty != null)

{

var filterTypeValue = filterTypeProperty.GetValue(filterModel);

if (filterTypeValue != null && filterTypeValue is FilterType filterType)

{

return filterType;

}

}

// Default to FilterType.Equal if the property is not found or invalid

return FilterType.Equal;

}

private static Expression AddFilterExpressions<TFilter>(ParameterExpression parameter, string propertyName, string propertyValue, Expression filterExpression, TFilter filterModel)

{

//Entity parametri ve Filterin eyni adli propName-ə əsasən prop yaradılır.

MemberExpression property = Expression.Property(parameter, propertyName);

MethodInfo containsMethod = typeof(string).GetMethod("Contains", new[] { typeof(string) });

Expression equality = null;

FilterType filterType = GetFilterType<TFilter>(propertyName, filterModel);

if (property.Type == typeof(string))

{

switch (filterType)

{

case FilterType.Equal:

equality = Expression.Equal(property, Expression.Constant(propertyValue));

break;

case FilterType.NotEqual:

equality = Expression.NotEqual(property, Expression.Constant(propertyValue));

break;

case FilterType.Contains:

equality = Expression.Call(property, containsMethod, Expression.Constant(propertyValue));

break;

default:

throw new NotSupportedException($"Filter type '{filterType}' is not supported for string properties.");

}

}

else if (property.Type == typeof(DateTime))

{

DateTime propertyDateTimeValue;

if (DateTime.TryParse(propertyValue, out propertyDateTimeValue))

{

propertyDateTimeValue = propertyDateTimeValue.Date; // Extract the date part

switch (filterType)

{

case FilterType.Equal:

equality = Expression.Equal(

Expression.Property(property, "Year"),

Expression.Constant(propertyDateTimeValue.Year)

);

equality = Expression.AndAlso(

equality,

Expression.Equal(

Expression.Property(property, "Month"),

Expression.Constant(propertyDateTimeValue.Month)

)

);

equality = Expression.AndAlso(

equality,

Expression.Equal(

Expression.Property(property, "Day"),

Expression.Constant(propertyDateTimeValue.Day)

)

);

break;

case FilterType.NotEqual:

equality = Expression.NotEqual(

Expression.Property(property, "Year"),

Expression.Constant(propertyDateTimeValue.Year)

);

equality = Expression.OrElse(

equality,

Expression.NotEqual(

Expression.Property(property, "Month"),

Expression.Constant(propertyDateTimeValue.Month)

)

);

equality = Expression.OrElse(

equality,

Expression.NotEqual(

Expression.Property(property, "Day"),

Expression.Constant(propertyDateTimeValue.Day)

)

);

break;

case FilterType.GreaterThan:

equality = Expression.GreaterThan(

Expression.Property(property, "Value"),

Expression.Constant(propertyDateTimeValue.Date)

);

break;

case FilterType.LessThan:

equality = Expression.LessThan(

Expression.Property(property, "Value"),

Expression.Constant(propertyDateTimeValue.Date)

);

break;

default:

throw new NotSupportedException($"Filter type '{filterType}' is not supported for DateTime properties.");

}

}

}

else if (property.Type == typeof(DateTime?))

{

DateTime? propertyDateTimeValue = null;

if (DateTime.TryParse(propertyValue, out DateTime parsedValue))

{

propertyDateTimeValue = parsedValue.Date;

}

if (propertyDateTimeValue != default(DateTime))

{

switch (filterType)

{

case FilterType.Equal:

equality = Expression.Equal(

Expression.Property(Expression.Property(property, "Value"), "Year"),

Expression.Constant(propertyDateTimeValue.GetValueOrDefault().Year)

);

equality = Expression.AndAlso(

equality,

Expression.Equal(

Expression.Property(Expression.Property(property, "Value"), "Month"),

Expression.Constant(propertyDateTimeValue.GetValueOrDefault().Month)

)

);

equality = Expression.AndAlso(

equality,

Expression.Equal(

Expression.Property(Expression.Property(property, "Value"), "Day"),

Expression.Constant(propertyDateTimeValue.GetValueOrDefault().Day)

)

);

break;

case FilterType.NotEqual:

equality = Expression.NotEqual(

Expression.Property(Expression.Property(property, "Value"), "Year"),

Expression.Constant(propertyDateTimeValue.GetValueOrDefault().Year)

);

equality = Expression.OrElse(

equality,

Expression.NotEqual(

Expression.Property(Expression.Property(property, "Value"), "Month"),

Expression.Constant(propertyDateTimeValue.GetValueOrDefault().Month)

)

);

equality = Expression.OrElse(

equality,

Expression.NotEqual(

Expression.Property(Expression.Property(property, "Value"), "Day"),

Expression.Constant(propertyDateTimeValue.GetValueOrDefault().Day)

)

);

break;

case FilterType.GreaterThan:

equality = Expression.GreaterThan(

Expression.Property(Expression.Property(property, "Value"), "Date"),

Expression.Constant(propertyDateTimeValue.GetValueOrDefault().Date)

);

break;

case FilterType.LessThan:

equality = Expression.LessThan(

Expression.Property(Expression.Property(property, "Value"), "Date"),

Expression.Constant(propertyDateTimeValue.GetValueOrDefault().Date)

);

break;

default:

throw new NotSupportedException($"Filter type '{filterType}' is not supported for DateTime properties.");

}

}

}

else if (property.Type == typeof(decimal))

{

decimal propertyDecimalValue;

if (decimal.TryParse(propertyValue, out propertyDecimalValue))

{

switch (filterType)

{

case FilterType.Equal:

equality = Expression.Equal(property, Expression.Constant(propertyDecimalValue));

break;

case FilterType.NotEqual:

equality = Expression.NotEqual(property, Expression.Constant(propertyDecimalValue));

break;

case FilterType.GreaterThan:

equality = Expression.GreaterThan(property, Expression.Constant(propertyDecimalValue));

break;

case FilterType.LessThan:

equality = Expression.LessThan(property, Expression.Constant(propertyDecimalValue));

break;

default:

throw new NotSupportedException($"Filter type '{filterType}' is not supported for decimal properties.");

}

}

}

else

{

throw new NotSupportedException($"Filtering is not supported for properties of type '{property.Type}'.");

}

return filterExpression == null ? equality : Expression.AndAlso(filterExpression, equality);

}

}

Method və modellər əlavə edildikdən sonra olan son mərhələ Filter alqoritmasına hər hansı bir yerdən müraciətin edilməsidir. Aşağıdakı kodlarda bunu ətraflı görmək mümkündür.

Expression<Func<RegisterOfCasesDangerousDAO, bool>> CDFilter = BaseFilterAlgorithm<RegisterOfCasesDangerousDAO>.GenerateFilterExpression(model.CasesDangerousFilter);

Expression<Func<CountryDAO, bool>> CountryFilter = BaseFilterAlgorithm<CountryDAO>.GenerateFilterExpression(model.Country);

Expression<Func<DangerousSituationDAO, bool>> DangerSiation = BaseFilterAlgorithm<DangerousSituationDAO>.GenerateFilterExpression(model.DangerSituation);

if (CDFilter == null && CountryFilter == null && DangerSiation == null)

return new ErrorDataResult<List<CasesDangerousModel>>(Messages.NotFilter);

using (QueryContext context = new QueryContext())

{

var country = CountryFilter == null ? context.Country.AsQueryable() : context.Country.Where(CountryFilter).AsQueryable();

var dangerousSituation = DangerSiation == null ? context.DangerousSituation.AsQueryable() : context.DangerousSituation.Where(DangerSiation).AsQueryable();

var query = CDFilter == null ? context.RegisterOfCasesDangerous.AsQueryable() : context.RegisterOfCasesDangerous.Where(CDFilter).AsQueryable();

query = query.Join(country, cd => cd.CountryId, c => c.Id, (cd, c) => cd);

query = query.Join(dangerousSituation, cd => cd.DangerousSituationsCollectionId, ds => ds.CollectionId, (cd, ds) => cd);

…

}