mcnkbr / **GenericRepository.cs**

Last active 9 months ago • Report abuse

Embed <script src="https://gist. Download ZIP

Generic Repository and AutoMapper for .NET Entity Framework

GenericRepository.cs

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Linq.Expressions;
using Common.Extensions;
using AutoMapper;

namespace Repository
{
    public abstract class GenericRepository<TC, TM, TD> : IGenericRepository<TM, TD>
        where TM : class
        where TD : class
        where TC : Entity.AeeeEntities, new()
    {

        private TC _entities = new TC();
        public TC Context
        {
            get { return _entities; }
            set { _entities = value; }
        }

        public virtual List<TD> GetAll(bool mapReset = true)
        {
            var ent = _entities.Set<TM>();
            var query = ent.ToList();

            if (mapReset)
                Mapper.Reset();
            var list = MapToDtoList<TM, TD>(query).ToList();
            return list;
        }

        public List<TD> FindBy(Expression<Func<TM, bool>> predicate, bool mapReset = true)
        {
            var ent = _entities.Set<TM>();
            var query = ent.Where(predicate).ToList();

            if (mapReset)
                Mapper.Reset();

            var list = MapToDtoList<TM, TD>(query).ToList();
            return list;
        }

        public TD SingleOrDefault(Expression<Func<TM, bool>> predicate)
        {
            var ent = _entities.Set<TM>();
            var query = ent.SingleOrDefault(predicate);

            if (query == null)
                return null;

            Mapper.Reset();
            Mapper.CreateMap<TM, TD>();
            var item = Mapper.Map<TM, TD>(query);

            return item;
        }

        public bool Any(Expression<Func<TM, bool>> predicate)
        {
            var result = _entities.Set<TM>().Any(predicate);
            return result;
        }

        public virtual void Add(TD entity)
        {
            Mapper.Reset();
            Mapper.CreateMap<TD, TM>();
            var item = Mapper.Map<TD, TM>(entity);

            _entities.Set<TM>().Add(item);
            Save();
        }

        public virtual int Add(TD entity, bool returnId, string returnName)
        {
            Mapper.CreateMap<TD, TM>();
            var item = Mapper.Map<TD, TM>(entity);

            _entities.Set<TM>().Add(item);
            Save();
            return returnId ? (int)item.GetType().GetProperty(returnName).GetValue(item, null) : 0;
        }

        public virtual void Add(TM entity)
        {
            _entities.Set<TM>().Add(entity);
            Save();
        }

        public virtual TM AddGetId(TD entity)
        {
            Mapper.CreateMap<TD, TM>();
            var item = Mapper.Map<TD, TM>(entity);

            _entities.Set<TM>().Add(item);
            Save();
            return item;
        }

        public virtual void Delete(Expression<Func<TM, bool>> predicate)
        {
            _entities.Set<TM>().RemoveRange(_entities.Set<TM>().Where(predicate));
            Save();
        }

        public virtual void Edit(TD entity, bool hasMap = false)
        {
            if (!hasMap)
                Mapper.CreateMap<TD, TM>();
            var participationDto = Mapper.Map<TD, TM>(entity);

            _entities.Set<TM>().Attach(participationDto);
            _entities.Entry(participationDto).State = EntityState.Modified;

            Save();
        }

        public virtual void Save()
        {
            _entities.SaveChanges();
        }

        public IQueryable<TM> List(Expression<Func<TM, bool>> filter = null, Func<IQueryable<TM>,
            IOrderedQueryable<TM>> orderBy = null, List<Expression<Func<TM, object>>> includeProperties = null,
        int? page = null, int? pageSize = null)
        {
            IQueryable<TM> query = _entities.Set<TM>();

            if (includeProperties != null)
                includeProperties.ForEach(i => { query = query.Include(i); });

            if (filter != null)
                query = query.Where(filter);

            if (orderBy != null)
            {
                query = orderBy(query);
            }

            if (page != null && pageSize != null)
                query = query
                    .Skip(page.Value)
                    .Take(pageSize.Value);

            return query;
        }

        public IQueryable<TM> List(Expression<Func<TM, bool>> filter = null, string orderBy = null, string ascendingDescending = "ASC",
            List<Expression<Func<TM, object>>> includeProperties = null,
       int? page = null, int? pageSize = null)
        {
            IQueryable<TM> query = _entities.Set<TM>();

            if (includeProperties != null)
                includeProperties.ForEach(i => { query = query.Include(i); });

            if (filter != null)
                query = query.Where(filter);

            if (page != null && pageSize != null)
                query = query
                    .OrderBy(orderBy ?? "Id", ascendingDescending == "ASC")
                    .Skip(page.Value)
                    .Take(pageSize.Value);

            return query;
        }

        public Tuple<IQueryable<TM>, int> ListWithPaging(Expression<Func<TM, bool>> filter = null, Func<IQueryable<TM>,
            IOrderedQueryable<TM>> orderBy = null, List<Expression<Func<TM, object>>> includeProperties = null,
        int? page = null, int? pageSize = null)
        {
            IQueryable<TM> query = _entities.Set<TM>();

            if (includeProperties != null)
```

```
                includeProperties.ForEach(i => { query = query.Include(i); });

            if (filter != null)
                query = query.Where(filter);

            if (orderBy != null)
            {
                query = orderBy(query);
            }
            var count = query.Count();
            if (page != null && pageSize != null)
                query = query
                    .Skip(page.Value)
                    .Take(pageSize.Value);

            return new Tuple<IQueryable<TM>, int>(query, count);
        }

        public Tuple<IQueryable<TM>, int> ListWithPaging(Expression<Func<TM, bool>> filter = null, string orderBy = null, string ascendingD
           List<Expression<Func<TM, object>>> includeProperties = null,
      int? page = null, int? pageSize = null)
        {
            IQueryable<TM> query = _entities.Set<TM>();

            if (includeProperties != null)
                includeProperties.ForEach(i => { query = query.Include(i); });

            if (filter != null)
                query = query.Where(filter);

            var count = query.Count();

            if (page != null && pageSize != null)
                query = query
                    .OrderBy(orderBy ?? "Id", ascendingDescending == "ASC")
                    .Skip(page.Value)
                    .Take(pageSize.Value);

            return new Tuple<IQueryable<TM>, int>(query, count);
        }

        public IQueryable<TD> ToDtoListPaging(List<TD> list, string orderBy = null, string ascendingDescending = "ASC", int? page = null, i
        {
            IQueryable<TD> query = list.AsQueryable();

            if (page != null && pageSize != null)
                query = query
                    .OrderBy(orderBy ?? "Id", ascendingDescending == "ASC")
                    .Skip(page.Value)
                    .Take(pageSize.Value);

            return query;
        }

        public virtual IEnumerable<TDto> MapToDtoList<TEntity, TDto>(IEnumerable<TEntity> entity)
            where TEntity : class
            where TDto : class
        {
            Mapper.CreateMap<TEntity, TDto>();
            return Mapper.Map<IEnumerable<TEntity>, IEnumerable<TDto>>(entity);
        }

        public virtual IEnumerable<TEntity> MapToEntityList<TDto, TEntity>(IEnumerable<TDto> dto)
            where TDto : class
            where TEntity : class
        {
            Mapper.CreateMap<TDto, TEntity>();
            return Mapper.Map<IEnumerable<TDto>, IEnumerable<TEntity>>(dto);
        }
    }
}
```

IGenericRepository.cs

```
using System;
using System.Collections.Generic;
using System.Linq.Expressions;

namespace Repository
{
    public interface IGenericRepository<TM, TD>
        where TM : class
        where TD : class
    {
        List<TD> GetAll(bool mapReset = true);
        List<TD> FindBy(Expression<Func<TM, bool>> predicate, bool mapReset = true);
        void Add(TD entity);
        void Add(TM entity);
        int Add(TD entity, bool returnId, string returnName);
        void Delete(Expression<Func<TM, bool>> predicate);
        void Edit(TD entity, bool hasMap = false);
        void Save();
    }
}
```

QueryableExtensions.cs

```
using System.Linq;
using System.Linq.Expressions;
using System.Reflection;

namespace Common.Extensions
{
    public static class QueryableExtensions
    {
        public static IQueryable<TM> OrderBy<TM>(this IQueryable<TM> source, string orderByProperty,
                      bool asc) where TM : class
        {
            var command = asc ? "OrderBy" : "OrderByDescending";
            var type = typeof(TM);
            PropertyInfo property;
            if (!orderByProperty.Contains('.'))
                property = type.GetProperties().FirstOrDefault(w => w.Name == orderByProperty);
            else
            {
                var arrg = orderByProperty.Split('.');
                property = type.GetProperty(arrg[0]).GetType().GetProperty(arrg[1]);
            }
            var parameter = Expression.Parameter(type, "p");
            if (property == null) return null;
            var propertyAccess = Expression.MakeMemberAccess(parameter, property);
            var orderByExpression = Expression.Lambda(propertyAccess, parameter);
            var resultExpression = Expression.Call(typeof (Queryable), command, new[] {type, property.PropertyType},
                source.Expression, Expression.Quote(orderByExpression));
            return source.Provider.CreateQuery<TM>(resultExpression);
        }
    }
}
```

UserRepository.cs

```
using Dto.Dto;
using Dto.GeneralModel;
using Dto.RequestModel;
using System;
using System.Collections.Generic;
using System.Linq;
using Entity;
using Dto.Enums;

namespace Repository
{
    public class UserRepository : GenericRepository<Entity.AeeeEntities, Entity.User, UserDto>
    {
        //Add,Edit,Delete,GetAll,FindBy methods created by GenericRepository
        //base.Add(...);
    }
}
```

GutsFun commented on Dec 10, 2014

Your OrderBy extension will not work correctly for nested properties. I suggest you write it like this :

```
public static IQueryable<TM> OrderBy<TM>(this IQueryable<TM> source, string orderByProperty,
              bool asc) where TM : class
{
    var command = asc ? "OrderBy" : "OrderByDescending";
    var type = typeof(TM);
    var parameter = Expression.Parameter(type, "p");
    string[] PropertyName = orderByProperty.Split('.');
    MemberExpression propertyAccess = (!orderByProperty.Contains('.')) ?
            Expression.Property(parameter, PropertyName[0]) :
            Expression.Property(Expression.Property(parameter, PropertyName[0]), PropertyName[1]);
    var orderByExpression = Expression.Lambda(propertyAccess, parameter);
    var resultExpression = Expression.Call(typeof (Queryable), command, new[] {type, propertyAccess.Type },
        source.Expression, Expression.Quote(orderByExpression));
    return source.Provider.CreateQuery<TM>(resultExpression);
}
```