CODE REVIEW

Generic repository and generic service

2

I use the Repository/Service design pattern in my projects. I have doubts whether it is worth using common services

`BaseEntity` :

```
public class BaseEntity
   {
       private DateTime _addedDate;
       private DateTime _modifiedDate;
       protected BaseEntity()
       {
           Id = Guid.NewGuid();
           AddedDate = DateTime.UtcNow;
       }
```

```
        public Guid Id { get; set; }
        public DateTime AddedDate
        {
            get => DateTime.SpecifyKind(_addedDate, DateTimeKind.Utc);
            private set => _addedDate = value;
        }
        public DateTime ModifiedDate
        {
            get => DateTime.SpecifyKind(_modifiedDate, DateTimeKind.Utc);
            set => _modifiedDate = value;
        }
    }
```

IGenericRepository :

```
    public interface IGenericRepository<T> where T : class
    {
        Task<T> FirstAsync(Expression<Func<T, bool>> predicate);
        Task<T> FirstOrDefaultAsync(Expression<Func<T, bool>> predicate);

        /// <summary>
        /// Get all queries
        /// </summary>
        /// <returns>IQueryable queries</returns>
        IQueryable<T> GetAll();

        /// <summary>
        /// Find queries by predicate
        /// </summary>
        /// <param name="predicate">search predicate (LINQ)</param>
        /// <returns>IQueryable queries</returns>
        IQueryable<T> FindBy(Expression<Func<T, bool>> predicate);

        /// <summary>
        /// Find entity by keys
        /// </summary>
        /// <param name="keys">search key</param>
        /// <returns>T entity</returns>
        Task<T> FindAsync(params object[] keys);

        /// <summary>
        /// Add new entity
        /// </summary>
        /// <param name="entity"></param>
```

```
        /// <summary>
        /// Remove entity from database
        /// </summary>
        /// <param name="entity"></param>
        void Delete(T entity);

        /// <summary>
        /// Remove entity from database
        /// </summary>
        /// <param name="keys">entity keys</param>
        void Delete(params object[] keys);

        /// <summary>
        /// Edit entity
        /// </summary>
        /// <param name="entity"></param>
        Task UpdateAsync(T entity);

        /// <summary>
        /// Persists all updates to the data source.
        /// </summary>
        void SaveChanges();
        Task SaveChangesAsync();
    }
```

GenericRepository :

```
 public class GenericRepository<T> : IGenericRepository<T> where T : BaseEntity
    {
        private readonly DbContext _context;
        private readonly DbSet<T> _dbSet;

        public GenericRepository(DbContext context)
        {
            _context = context;
            _dbSet = context.Set<T>();
        }
        public virtual async Task<T> FirstAsync(Expression<Func<T, bool>> predicate)
        {
            return await _dbSet.FirstAsync(predicate);
        }

        public virtual async Task <T> FirstOrDefaultAsync(Expression<Func<T, bool>>
predicate)
```

```
public virtual IQueryable<T> GetAll()
{
    return _dbSet.AsNoTracking();
}

public virtual IQueryable<T> FindBy(Expression<Func<T, bool>> predicate)
{
    return _dbSet.Where(predicate);
}

public async Task<T> FindAsync(params object[] keys)
{
    return await _dbSet.FindAsync(keys);
}

public virtual async Task AddAsync(T entity)
{
    await _dbSet.AddAsync(entity);
}

public virtual void Delete(T entity)
{
    _dbSet.Remove(entity);
}

public virtual void Delete(params object[] keys)
{
    var entity = _dbSet.Find(keys);
    _dbSet.Remove(entity);
}

public virtual async Task UpdateAsync(T entity)
{
    var existing = await _dbSet.FindAsync(entity.Id);
    if (existing != null)
    {
        existing.ModifiedDate = DateTime.UtcNow;
        _context.Entry(existing).CurrentValues.SetValues(entity);
        _context.Entry(existing).Property("AddedDate").IsModified = false;
    }
}

public virtual void SaveChanges()
{
     context.SaveChanges();
```

```
        {
            await _context.SaveChangesAsync();
        }
    }
```

CommonService :

```
    public class CommonService<T> : ICommonService<T> where T : BaseEntity
    {
        private readonly IGenericRepository<T> _repository;

        public CommonService(IGenericRepository<T> repository)
        {
            _repository = repository;
        }

        public virtual async Task<T> FirstAsync(Expression<Func<T, bool>> predicate)
        {
            return await _repository.FirstAsync(predicate);
        }

        public virtual async Task<T> FirstOrDefaultAsync(Expression<Func<T, bool>>
predicate)
        {
            return await _repository.FirstOrDefaultAsync(predicate);
        }

        public virtual IQueryable<T> GetAll()
        {
            return _repository.GetAll();
        }

        public virtual IQueryable<T> FindBy(Expression<Func<T, bool>> predicate)
        {
            return _repository.FindBy(predicate);
        }

        public async Task<T> FindAsync(params object[] keys)
        {
            return await _repository.FindAsync(keys);
        }

        public virtual async Task AddAsync(T entity)
        {
```

```
        public virtual async Task DeleteAsync(T entity)
        {
            _repository.Delete(entity);
            await _repository.SaveChangesAsync();
        }

        public virtual async Task DeleteAsync(params object[] keys)
        {
            var entity = await _repository.FindAsync(keys);
            _repository.Delete(entity);
            await _repository.SaveChangesAsync();
        }

        public virtual async Task UpdateAsync(T entity)
        {
            await _repository.UpdateAsync(entity);
            await _repository.SaveChangesAsync();
        }
    }
```

`ConcreteService`:

```
public class DepartmentService : CommonService<Department>, IDepartmentService
    {
        private readonly IGenericRepository<Department> _repository;

        public DepartmentService(IGenericRepository<Department> repository) :
base(repository)
        {
            _repository = repository;
        }
    }
```

Questions:

1. Is it normal that I use the shared service as a base class to avoid duplicate code?
2. Do I need to create empty services if all the necessary operations have in the base class (`CommonService`), or in this case, always use common services (inject `ICommonService<EntityName>` and usage and create specific service classes only when there is a specific 'non-genericable' logic in that?

edited May 29 '18 at 18:01

Jamal ♦
31.4k 12 123 230

asked Jul 25 '17 at 21:50

Artem Polishchuk
11 4

1 Even a generic repository is disputable. What are the driving forces behind your architecture? It's hard to give any advice as to what is "normal" without knowing why you need all these layers. – Gert Arnold Jul 25 '17 at 22:32

So the common-service just adds the *save* operation to the repository? This doesn't look right. Why did you implement it this way? – t3chb0t Jul 26 '17 at 4:20

@t3chb0t I need Repository for access to context (i think use dbcontext in 'common' or another service not good way) i use common service to avoid duplication same code, but my service can contains custom code, so i dont know i need create empty service and usage common service as base class, or need usage always commonserivce and create 'custom' service only when i have additional logic? – Artem Polishchuk Jul 30 '17 at 7:49

What happens when you need to create a stored procedure (or w/e db equivalent you're using) because EF doesn't perform well enough (or insert some other reason). Are you adding that function to the repository? If so, is it then still an `IGenericRepository` or just an `IRepository` ? Just pointing out that there's a decent chance there will be an exception in this architecture, best to know how you're going to handle it ahead of time. I use nearly this exact architecture and I've been running into exceptions ever since. – Shelby115 May 29 '18 at 18:12