

# Generic Repository Pattern with Generic Services

**Generic Services will make your life easy as you did not need to implement same functionality again and again. Just write them once and use.**

| | | | |
|---|---|---|---|
| Ratings | (2) | Updated | 10/11/2016 |
| Favorites | Add to favorites | License | Apache License, Version 2.0 |
| | | Share | |

Technologies C#, Generics, Repository Pattern, ASP.NET MVC 4, Entity Framework 6

Topics Generic C# resuable code, Generic Repository

Report abuse to Microsoft

Description Q and A

## Introduction

*Generic Repository Pattern is currently wide used for MVC .NET with **EF.** In this simple code i had created a simple project with need only one table User and perform crud Functionality using Generic Service.*

### Creating Generic Service

*When i Started a repository pattern i soon realize that there must be a way to build a base service that perfom all the basic process for every service just Generic reporsitory. On this weekend i have tried to build a Generic Service with repository pattern . Its really awesome it reduces almost 60% of code from services.DownLoad Code*

## Description

To Create Generic Serivce you Create a generic interface that have three tamplate Variable

1. M(Template for POCO model)
2. T(Tamplate for  Table for table of EF)
3. E(Tampalte for Entity class of generic repository)

C#

```
public interface IService<M, T, E>
   {
      M GetSingle(Expression<Func<T, bool>> whereCondition);

      void Add(M entity);

      void Delete(M entity);

      void Update(M entity);

      List<M> GetAll(Expression<Func<T, bool>> whereCondition);

      List<M> GetAll();

      IQueryable<T> Query(Expression<Func<T, bool>> whereCondition);

      long Count(Expression<Func<T, bool>> whereCondition);

      long Count();
   }
```

Create a Base Service template that implement ISevice class and implement all the generic methods.

User automapper to convert to POCO from Entity and vice versa.

C#

```
   public abstract class ServiceBase<M, T, E> : IService<M, T, E>
```

```
    where E : RepositoryBase<T>
    where T : class
{
    E Entity;
    public ServiceBase(RepositoryBase<T> obj)
    {
        AutoMapper.Mapper.CreateMap<T, M>();
        AutoMapper.Mapper.CreateMap<M, T>();
        Entity = (E)obj;
    }

    public M GetSingle(System.Linq.Expressions.Expression<Func<T, bool>> whereCondition)
    {
        var model = Entity.GetQueryable().Where(whereCondition).FirstOrDefault();
        return Mapper.Map<T, M>(model);

    }

    public void Add(M entity)
    {
        var model = Mapper.Map<M, T>(entity);
        Entity.Add(model);
    }

    public void Delete(M entity)
    {
        T model = Mapper.Map<M, T>(entity);
        Entity.Delete(model);
    }

    public void Update(M entity)
    {
        try
        {
            T model = Mapper.Map<M, T>(entity);
            T tEntity = Entity.GetSingle(Entity.SearchFilters(model));
            Entity.Update(tEntity, model);
        }
        catch (Exception)
        {

            throw;
        }
    }

    public List<M> GetAll(System.Linq.Expressions.Expression<Func<T, bool>> whereCondition)
    {
        return Entity.GetQueryable().Where(whereCondition).Select(Mapper.Map<T, M>).ToList();
    }
```

```
        public List<M> GetAll()
        {
            return Entity.GetQueryable().Select(Mapper.Map<T, M>).ToList();
        }

        public IQueryable<T> Query(System.Linq.Expressions.Expression<Func<T, bool>> whereCondition)
        {
            return Entity.GetQueryable().Where(whereCondition).AsQueryable();
        }

        public long Count(System.Linq.Expressions.Expression<Func<T, bool>> whereCondition)
        {
            return Entity.GetQueryable().Where(whereCondition).Count();
        }

        public long Count()
        {
            return Entity.GetQueryable().Count();
        }
    }
```

### Service that implement service base Class

Create an Interface for your custom Service that implement IService class.Create Service that implement Service base.

You will automatically have all the functionality for your service

1. USerModel(Plan C# Class)

2. Identity_User (Entity Framwork Class)

3. USers,IUsers(Class that implement Generice Repository)

C#

```
#region Interfaces
  public interface IUsersService : IService<UserModel, Identity_User, IUsers>
  {
  }
#endregion

  public class UsersService : ServiceBase<UserModel, Identity_User, Users>, IUsersService
  {
```

```
        public UsersService()
            : this(new Users())
        {

        }
        public UsersService(Users usr)
            : base(usr)
        { }
    }
```

Here is Magic in Controller.Controller you will have access to all Mathods that you have implement in Generic class

C#

```
public class HomeController : Controller
    {
        IUsersService serUSer;
        public HomeController()
            : this(new UsersService())
        { }
        public HomeController(IUsersService usr)
        {
            serUSer = usr ?? new UsersService();
        }
        public ActionResult Index()
        {
            UserModel result = new UserModel();
            try
            {
                result.UserList = serUSer.GetAll();
            }
            catch (Exception ex)
            {

                throw ex;
            }
            return View(result);
        }
}
```