```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstDemo.Domain.Entities
{
    public interface IEntity<T>
    {
        T Id { get; set; }
    }
}


===========================================================================

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstDemo.Domain.Entities
{
    public class Student : IEntity<Guid>
    {
        public Guid Id { get; set; }
        public string Name { get; set; }
        public double Cgpa { get; set; }
    }
}




===========================================================================


using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstDemo.Domain.Entities
{
    public class Course : IEntity<Guid>
    {
        public Guid Id { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public uint Fees { get; set; }
    }

}



===========================================================================

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstDemo.Domain.Entities
{
    public class CourseEnrollment
    {
        public Guid CourseId { get; set; }
        public Guid StudentId { get; set; }
        public DateTime EnrollmentDate { get; set; }
    }
}




===========================================================================


using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstDemo.Domain.Exceptions
{
    public class DuplicateTitleException : Exception
    {
        public DuplicateTitleException() : base("Title is Duplicate") { }
    }
}



===========================================================================



using FirstDemo.Domain.Entities;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Threading;
using System.Threading.Tasks;

namespace FirstDemo.Domain.Repositories
{
    public interface IRepositoryBase<TEntity, TKey>
        where TEntity : class, IEntity<TKey>
        where TKey : IComparable
    {
        void Add(TEntity entity);
        Task AddAsync(TEntity entity);
        void Edit(TEntity entityToUpdate);
        Task EditAsync(TEntity entityToUpdate);
        IList<TEntity> GetAll();
        Task<IList<TEntity>> GetAllAsync();
        TEntity GetById(TKey id);
        Task<TEntity> GetByIdAsync(TKey id);
        int GetCount(Expression<Func<TEntity, bool>> filter = null);
        Task<int> GetCountAsync(Expression<Func<TEntity, bool>> filter = null);
        void Remove(Expression<Func<TEntity, bool>> filter);
        void Remove(TEntity entityToDelete);
        void Remove(TKey id);
        Task RemoveAsync(Expression<Func<TEntity, bool>> filter);
        Task RemoveAsync(TEntity entityToDelete);
        Task RemoveAsync(TKey id);

    }
}

===========================================================================



using FirstDemo.Domain.Entities;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstDemo.Domain.Repositories
{
    public interface ICourseRepository : IRepositoryBase<Course, Guid>
    {
        Task<bool> IsTitleDuplicateAsync(string title, Guid? id = null);

        Task<(IList<Course> records, int total, int totalDisplay)>
            GetTableDataAsync(string searchTitle, uint searchFeeFrom,
            uint searchFeeTo, string orderBy, int pageIndex, int pageSize);
    }
}


===========================================================================



using System;
using System.Collections.Generic;
using System.Text;

namespace FirstDemo.Domain
{
    public interface IUnitOfWork : IDisposable, IAsyncDisposable
    {
        void Save();
        Task SaveAsync();
    }
}


===========================================================================


using FirstDemo.Application.Features.Training.DTOs;
using FirstDemo.Domain.Entities;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstDemo.Application.Features.Training.Services
{
    public interface ICourseManagementService
    {
        Task CreateCourseAsync(string title, string description, uint fees);
        Task DeleteCourseAsync(Guid id);
        Task<Course> GetCourseAsync(Guid id);
        Task<(IList<Course> records, int total, int totalDisplay)>
            GetPagedCoursesAsync(int pageIndex, int pageSize, string searchTitle, uint searchFeeFrom,
            uint searchFeeTo, string sortBy);
        Task UpdateCourseAsync(Guid id, string title, string description, uint fees);
        Task<(IList<CourseEnrollmentDTO> records, int total, int totalDisplay)>
            GetCourseEnrollmentsAsync(int pageIndex, int pageSize, string orderBy,
            string courseName, string studentName, DateTime enrollmentDateFrom,
            DateTime enrollmentDateTo);
    }
}



===========================================================================


using FirstDemo.Application.Features.Training.DTOs;
using FirstDemo.Domain.Entities;
using FirstDemo.Domain.Exceptions;
using FirstDemo.Application.Features.Training.Services;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstDemo.Application.Features.Training.Services
{
    public class CourseManagementService : ICourseManagementService
    {
        private readonly IApplicationUnitOfWork _unitOfWork;
        public CourseManagementService(IApplicationUnitOfWork unitOfWork)
        {
            _unitOfWork = unitOfWork;
        }

        public async Task CreateCourseAsync(string title, string description, uint fees)
        {
            bool isDuplicateTitle = await _unitOfWork.CourseRepository.IsTitleDuplicateAsync(title);

            if (isDuplicateTitle)
                throw new DuplicateTitleException();

            Course course = new Course
            {
                Title = title,
                Fees = fees,
                Description = description
            };

            _unitOfWork.CourseRepository.Add(course);
            await _unitOfWork.SaveAsync();
        }

        public async Task DeleteCourseAsync(Guid id)
        {
            await _unitOfWork.CourseRepository.RemoveAsync(id);
            await _unitOfWork.SaveAsync();
        }

        public async Task<Course> GetCourseAsync(Guid id)
        {
            return await _unitOfWork.CourseRepository.GetByIdAsync(id);
        }

        public async Task<(IList<Course> records, int total, int totalDisplay)>
            GetPagedCoursesAsync(int pageIndex, int pageSize, string searchTitle, uint searchFeeFrom,
uint searchFeeTo, string sortBy)
        {
            return await _unitOfWork.CourseRepository.GetTableDataAsync(searchTitle, searchFeeFrom,
searchFeeTo, sortBy, pageIndex, pageSize);

        }

        public async Task UpdateCourseAsync(Guid id, string title, string description, uint fees)
        {
            bool isDuplicateTitle = await _unitOfWork.CourseRepository
                .IsTitleDuplicateAsync(title, id);
            if (isDuplicateTitle)
                throw new DuplicateTitleException();
            var course = await GetCourseAsync(id);
            if (course is not null)
            {
                course.Title = title;
                course.Description = description;
                course.Fees = fees;
            }

            await _unitOfWork.SaveAsync();
        }
        public async Task<(IList<CourseEnrollmentDTO> records, int total, int totalDisplay)>
            GetCourseEnrollmentsAsync(int pageIndex, int pageSize, string orderBy,
            string courseName, string studentName, DateTime enrollmentDateFrom,
            DateTime enrollmentDateTo)
        {
            return await _unitOfWork.GetCourseEnrollmentsAsync(
                pageIndex, pageSize, orderBy, courseName,
                studentName, enrollmentDateFrom, enrollmentDateTo);
        }
    }

}



===========================================================================


using FirstDemo.Application.Features.Training.DTOs;
using FirstDemo.Domain;
using FirstDemo.Domain.Repositories;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstDemo.Application
{
    public interface IApplicationUnitOfWork : IUnitOfWork
    {
        ICourseRepository CourseRepository { get; }
        Task<(IList<CourseEnrollmentDTO> records, int total, int totalDisplay)>
            GetCourseEnrollmentsAsync(
            int pageIndex,
            int pageSize,
            string orderBy,
            string courseName,
            string studentName,
            DateTime enrollmentDateFrom,
            DateTime enrollmentDateTo);
    }
}



===========================================================================


using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstDemo.Application.Features.Training.DTOs
{
    public class CourseEnrollmentDTO
    {
        public string StudentName { get; set; }
        public string CourseName { get; set; }
        public DateTime EnrollmentDate { get; set; }
    }
}



===========================================================================

using Autofac;
using FirstDemo.Application.Features.Training.Services;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstDemo.Application
{
    public class ApplicationModule : Module
    {
        protected override void Load(ContainerBuilder builder)
        {
            builder.RegisterType<CourseManagementService>().As<ICourseManagementService>()
                .InstancePerLifetimeScope();
        }
    }
}




===========================================================================

using FirstDemo.Domain.Entities;
using Microsoft.EntityFrameworkCore.Query;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Text;
using System.Threading.Tasks;
using FirstDemo.Domain.Repositories;

namespace FirstDemo.Infrastructure.Repositories
{
    public interface IRepository<TEntity, TKey> : IRepositoryBase<TEntity, TKey>
        where TEntity : class, IEntity<TKey>
        where TKey : IComparable
    {
        IList<TEntity> Get(Expression<Func<TEntity, bool>> filter = null, Func<IQueryable<TEntity>,
IOrderedQueryable<TEntity>> orderBy = null, Func<IQueryable<TEntity>, IIncludableQueryable<TEntity,
object>> include = null, bool isTrackingOff = false);
        (IList<TEntity> data, int total, int totalDisplay) Get(Expression<Func<TEntity, bool>> filter
= null, Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>> orderBy = null,
Func<IQueryable<TEntity>, IIncludableQueryable<TEntity, object>> include = null, int pageIndex = 1,
int pageSize = 10, bool isTrackingOff = false);
        IList<TEntity> Get(Expression<Func<TEntity, bool>> filter, Func<IQueryable<TEntity>,
IIncludableQueryable<TEntity, object>> include = null);
        Task<IList<TEntity>> GetAsync(Expression<Func<TEntity, bool>> filter = null,
Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>> orderBy = null, Func<IQueryable<TEntity>,
IIncludableQueryable<TEntity, object>> include = null, bool isTrackingOff = false);
        Task<(IList<TEntity> data, int total, int totalDisplay)> GetAsync(Expression<Func<TEntity,
bool>> filter = null, Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>> orderBy = null,
Func<IQueryable<TEntity>, IIncludableQueryable<TEntity, object>> include = null, int pageIndex = 1,
int pageSize = 10, bool isTrackingOff = false);
        Task<IList<TEntity>> GetAsync(Expression<Func<TEntity, bool>> filter,
Func<IQueryable<TEntity>, IIncludableQueryable<TEntity, object>> include = null);
        Task<IEnumerable<TResult>> GetAsync<TResult>(Expression<Func<TEntity, TResult>> selector,
Expression<Func<TEntity, bool>> predicate = null, Func<IQueryable<TEntity>,
IOrderedQueryable<TEntity>> orderBy = null, Func<IQueryable<TEntity>, IIncludableQueryable<TEntity,
object>> include = null, bool disableTracking = true, CancellationToken cancellationToken = default)
where TResult : class;
        IList<TEntity> GetDynamic(Expression<Func<TEntity, bool>> filter = null, string orderBy =
null, Func<IQueryable<TEntity>, IIncludableQueryable<TEntity, object>> include = null, bool
isTrackingOff = false);
        (IList<TEntity> data, int total, int totalDisplay) GetDynamic(Expression<Func<TEntity, bool>>
filter = null, string orderBy = null, Func<IQueryable<TEntity>, IIncludableQueryable<TEntity,
object>> include = null, int pageIndex = 1, int pageSize = 10, bool isTrackingOff = false);
        Task<IList<TEntity>> GetDynamicAsync(Expression<Func<TEntity, bool>> filter = null, string
orderBy = null, Func<IQueryable<TEntity>, IIncludableQueryable<TEntity, object>> include = null, bool
isTrackingOff = false);
        Task<(IList<TEntity> data, int total, int totalDisplay)>
GetDynamicAsync(Expression<Func<TEntity, bool>> filter = null, string orderBy = null,
Func<IQueryable<TEntity>, IIncludableQueryable<TEntity, object>> include = null, int pageIndex = 1,
int pageSize = 10, bool isTrackingOff = false);
        Task<TResult> SingleOrDefaultAsync<TResult>(Expression<Func<TEntity, TResult>> selector,
Expression<Func<TEntity, bool>> predicate = null, Func<IQueryable<TEntity>,
IOrderedQueryable<TEntity>> orderBy = null, Func<IQueryable<TEntity>, IIncludableQueryable<TEntity,
object>> include = null, bool disableTracking = true);
    }
}




=============================================================================



using FirstDemo.Domain.Entities;
using FirstDemo.Domain;
using Microsoft.Data.SqlClient;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Query;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Data.SqlTypes;
using System.Globalization;
using System.Linq;
using System.Linq.Dynamic.Core;
using System.Linq.Expressions;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace FirstDemo.Infrastructure.Repositories
{
    public abstract class Repository<TEntity, TKey>
        : IRepository<TEntity, TKey> where TKey : IComparable
        where TEntity : class, IEntity<TKey>
    {
        private DbContext _dbContext;
        private DbSet<TEntity> _dbSet;

        public Repository(DbContext context)
        {
            _dbContext = context;
            _dbSet = _dbContext.Set<TEntity>();
        }

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

        public virtual async Task RemoveAsync(TKey id)
        {
            var entityToDelete = _dbSet.Find(id);
            await RemoveAsync(entityToDelete);
        }

        public virtual async Task RemoveAsync(TEntity entityToDelete)
        {
            await Task.Run(() =>
            {
                if (_dbContext.Entry(entityToDelete).State == EntityState.Detached)
                {
                    _dbSet.Attach(entityToDelete);
                }
                _dbSet.Remove(entityToDelete);
            });
        }

        public virtual async Task RemoveAsync(Expression<Func<TEntity, bool>> filter)
        {
            await Task.Run(() =>
            {
                _dbSet.RemoveRange(_dbSet.Where(filter));
            });
        }

        public virtual async Task EditAsync(TEntity entityToUpdate)
        {
            await Task.Run(() =>
            {
                _dbSet.Attach(entityToUpdate);
                _dbContext.Entry(entityToUpdate).State = EntityState.Modified;
            });
        }
        public virtual async Task<TEntity> GetByIdAsync(TKey id)
        {
            return await _dbSet.FindAsync(id);
        }

        public virtual async Task<int> GetCountAsync(Expression<Func<TEntity, bool>> filter = null)
        {
            IQueryable<TEntity> query = _dbSet;
            int count;

            if (filter != null)
            {
                count = await query.CountAsync(filter);
            }
            else
                count = await query.CountAsync();

            return count;
        }

        public virtual int GetCount(Expression<Func<TEntity, bool>> filter = null)
        {
            IQueryable<TEntity> query = _dbSet;
            int count;

            if (filter != null)
                count = query.Count(filter);
            else
                count = query.Count();
            return count;
        }

        public virtual async Task<IList<TEntity>> GetAsync(Expression<Func<TEntity, bool>> filter,
            Func<IQueryable<TEntity>, IIncludableQueryable<TEntity, object>> include = null)
        {
            IQueryable<TEntity> query = _dbSet;

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

            if (include != null)
                query = include(query);

            return await query.ToListAsync();
        }

        public virtual async Task<IList<TEntity>> GetAllAsync()
        {
            IQueryable<TEntity> query = _dbSet;
            return await query.ToListAsync();
        }

        public virtual async Task<(IList<TEntity> data, int total, int totalDisplay)> GetAsync(
            Expression<Func<TEntity, bool>> filter = null,
            Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>> orderBy = null,
            Func<IQueryable<TEntity>, IIncludableQueryable<TEntity, object>> include = null,
            int pageIndex = 1,
            int pageSize = 10,
            bool isTrackingOff = false)
        {
            IQueryable<TEntity> query = _dbSet;
            var total = query.Count();
            var totalDisplay = query.Count();

            if (filter != null)
            {
                query = query.Where(filter);
                totalDisplay = query.Count();
            }

            if (include != null)
                query = include(query);

            IList<TEntity> data;

            if (orderBy != null)
            {
                var result = orderBy(query).Skip((pageIndex - 1) * pageSize).Take(pageSize);

                if (isTrackingOff)
                    data = await result.AsNoTracking().ToListAsync();
                else
                    data = await result.ToListAsync();
            }
            else
            {
                var result = query.Skip((pageIndex - 1) * pageSize).Take(pageSize);

                if (isTrackingOff)
                    data = await result.AsNoTracking().ToListAsync();
                else
                    data = await result.ToListAsync();
            }

            return (data, total, totalDisplay);
        }

        public virtual async Task<(IList<TEntity> data, int total, int totalDisplay)>
GetDynamicAsync(
            Expression<Func<TEntity, bool>> filter = null,
            string orderBy = null,
            Func<IQueryable<TEntity>, IIncludableQueryable<TEntity, object>> include = null,
            int pageIndex = 1,
            int pageSize = 10,
            bool isTrackingOff = false)
        {
            IQueryable<TEntity> query = _dbSet;
            var total = query.Count();
            var totalDisplay = query.Count();

            if (filter != null)
            {
                query = query.Where(filter);
                totalDisplay = query.Count();
            }

            if (include != null)
                query = include(query);

            IList<TEntity> data;

            if (orderBy != null)
            {
                var result = query.OrderBy(orderBy).Skip((pageIndex - 1) * pageSize).Take(pageSize);

                if (isTrackingOff)
                    data = await result.AsNoTracking().ToListAsync();
                else
                    data = await result.ToListAsync();
            }
            else
            {
                var result = query.Skip((pageIndex - 1) * pageSize).Take(pageSize);

                if (isTrackingOff)
                    data = await result.AsNoTracking().ToListAsync();
                else
                    data = await result.ToListAsync();
            }

            return (data, total, totalDisplay);
        }

        public virtual async Task<IList<TEntity>> GetAsync(
            Expression<Func<TEntity, bool>> filter = null,
            Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>> orderBy = null,
            Func<IQueryable<TEntity>, IIncludableQueryable<TEntity, object>> include = null,
            bool isTrackingOff = false)
        {
            IQueryable<TEntity> query = _dbSet;

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

            if (include != null)
                query = include(query);

            if (orderBy != null)
            {
                var result = orderBy(query);

                if (isTrackingOff)
                    return await result.AsNoTracking().ToListAsync();
                else
                    return await result.ToListAsync();
            }
            else
            {
                if (isTrackingOff)
                    return await query.AsNoTracking().ToListAsync();
                else
                    return await query.ToListAsync();
            }
        }

        public virtual async Task<IList<TEntity>> GetDynamicAsync(
            Expression<Func<TEntity, bool>> filter = null,
            string orderBy = null,
            Func<IQueryable<TEntity>, IIncludableQueryable<TEntity, object>> include = null,
            bool isTrackingOff = false)
        {
            IQueryable<TEntity> query = _dbSet;

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

            if (include != null)
                query = include(query);
            if (orderBy != null)
            {
                var result = query.OrderBy(orderBy);

                if (isTrackingOff)
                    return await result.AsNoTracking().ToListAsync();
                else
                    return await result.ToListAsync();
            }
            else
            {
                if (isTrackingOff)
                    return await query.AsNoTracking().ToListAsync();
                else
                    return await query.ToListAsync();
            }
        }

        public virtual void Add(TEntity entity)
        {
            _dbSet.Add(entity);
        }

        public virtual void Remove(TKey id)
        {
            var entityToDelete = _dbSet.Find(id);
            Remove(entityToDelete);
        }

        public virtual void Remove(TEntity entityToDelete)
        {
            if (_dbContext.Entry(entityToDelete).State == EntityState.Detached)
            {
                _dbSet.Attach(entityToDelete);
            }
            _dbSet.Remove(entityToDelete);
        }

        public virtual void Remove(Expression<Func<TEntity, bool>> filter)
        {
            _dbSet.RemoveRange(_dbSet.Where(filter));
        }

        public virtual void Edit(TEntity entityToUpdate)
        {
            _dbSet.Attach(entityToUpdate);
            _dbContext.Entry(entityToUpdate).State = EntityState.Modified;
        }

        public virtual IList<TEntity> Get(Expression<Func<TEntity, bool>> filter,
            Func<IQueryable<TEntity>, IIncludableQueryable<TEntity, object>> include = null)
        {
            IQueryable<TEntity> query = _dbSet;

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

            if (include != null)
                query = include(query);

            return query.ToList();
        }
        public virtual IList<TEntity> GetAll()
        {
            IQueryable<TEntity> query = _dbSet;
            return query.ToList();
        }

        public virtual TEntity GetById(TKey id)
        {
            return _dbSet.Find(id);
        }

        public virtual (IList<TEntity> data, int total, int totalDisplay) Get(
            Expression<Func<TEntity, bool>> filter = null,
            Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>> orderBy = null,
            Func<IQueryable<TEntity>, IIncludableQueryable<TEntity, object>> include = null,
            int pageIndex = 1, int pageSize = 10, bool isTrackingOff = false)
        {
            IQueryable<TEntity> query = _dbSet;
            var total = query.Count();
            var totalDisplay = query.Count();

            if (filter != null)
            {
                query = query.Where(filter);
                totalDisplay = query.Count();
            }

            if (include != null)
                query = include(query);

            if (orderBy != null)
            {
                var result = orderBy(query).Skip((pageIndex - 1) * pageSize).Take(pageSize);
                if (isTrackingOff)
                    return (result.AsNoTracking().ToList(), total, totalDisplay);
                else
                    return (result.ToList(), total, totalDisplay);
            }
            else
            {
                var result = query.Skip((pageIndex - 1) * pageSize).Take(pageSize);
                if (isTrackingOff)
                    return (result.AsNoTracking().ToList(), total, totalDisplay);
                else
                    return (result.ToList(), total, totalDisplay);
            }
        }

        public virtual (IList<TEntity> data, int total, int totalDisplay) GetDynamic(
            Expression<Func<TEntity, bool>> filter = null,
            string orderBy = null,
            Func<IQueryable<TEntity>, IIncludableQueryable<TEntity, object>> include = null,
            int pageIndex = 1, int pageSize = 10, bool isTrackingOff = false)
        {
            IQueryable<TEntity> query = _dbSet;
            var total = query.Count();
            var totalDisplay = query.Count();

            if (filter != null)
            {
                query = query.Where(filter);
                totalDisplay = query.Count();
            }
            if (include != null)
                query = include(query);

            if (orderBy != null)
            {
                var result = query.OrderBy(orderBy).Skip((pageIndex - 1) * pageSize).Take(pageSize);
                if (isTrackingOff)
                    return (result.AsNoTracking().ToList(), total, totalDisplay);
                else
                    return (result.ToList(), total, totalDisplay);
            }
            else
            {
                var result = query.Skip((pageIndex - 1) * pageSize).Take(pageSize);
                if (isTrackingOff)
                    return (result.AsNoTracking().ToList(), total, totalDisplay);
                else
                    return (result.ToList(), total, totalDisplay);
            }
        }

        public virtual IList<TEntity> Get(Expression<Func<TEntity, bool>> filter = null,
            Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>> orderBy = null,
            Func<IQueryable<TEntity>, IIncludableQueryable<TEntity, object>> include = null,
            bool isTrackingOff = false)
        {
            IQueryable<TEntity> query = _dbSet;

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

            if (include != null)
                query = include(query);

            if (orderBy != null)
            {
                var result = orderBy(query);

                if (isTrackingOff)
                    return result.AsNoTracking().ToList();
                else
                    return result.ToList();
            }
            else
            {
                if (isTrackingOff)
                    return query.AsNoTracking().ToList();
                else
                    return query.ToList();
            }
        }

        public virtual IList<TEntity> GetDynamic(Expression<Func<TEntity, bool>> filter = null,
            string orderBy = null,
            Func<IQueryable<TEntity>, IIncludableQueryable<TEntity, object>> include = null,
            bool isTrackingOff = false)
        {
            IQueryable<TEntity> query = _dbSet;

            if (filter != null)
            {
                query = query.Where(filter);
            }
            if (include != null)
                query = include(query);

            if (orderBy != null)
            {
                var result = query.OrderBy(orderBy);

                if (isTrackingOff)
                    return result.AsNoTracking().ToList();
                else
                    return result.ToList();
            }
            else
            {
                if (isTrackingOff)
                    return query.AsNoTracking().ToList();
                else
                    return query.ToList();
            }
        }

        public async Task<IEnumerable<TResult>> GetAsync<TResult>(Expression<Func<TEntity, TResult>>?
selector,
            Expression<Func<TEntity, bool>>? predicate = null,
            Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>>? orderBy = null,
            Func<IQueryable<TEntity>, IIncludableQueryable<TEntity, object>>? include = null,
            bool disableTracking = true,
            CancellationToken cancellationToken = default) where TResult : class
        {
            var query = _dbSet.AsQueryable();
            if (disableTracking) query.AsNoTracking();
            if (include is not null) query = include(query);
            if (predicate is not null) query = query.Where(predicate);
            return orderBy is not null
                ? await orderBy(query).Select(selector!).ToListAsync(cancellationToken)
                : await query.Select(selector!).ToListAsync(cancellationToken);
        }

        public async Task<TResult> SingleOrDefaultAsync<TResult>(Expression<Func<TEntity, TResult>>?
selector,
            Expression<Func<TEntity, bool>>? predicate = null,
            Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>>? orderBy = null,
            Func<IQueryable<TEntity>, IIncludableQueryable<TEntity, object>>? include = null,
            bool disableTracking = true)
        {
            var query = _dbSet.AsQueryable();
            if (disableTracking) query.AsNoTracking();
            if (include is not null) query = include(query);
            if (predicate is not null) query = query.Where(predicate);
            return (orderBy is not null
                ? await orderBy(query).Select(selector!).FirstOrDefaultAsync()
                : await query.Select(selector!).FirstOrDefaultAsync())!;
        }
    }
}



==========================================================================


using FirstDemo.Domain.Entities;
using FirstDemo.Domain.Repositories;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Text;
using System.Threading.Tasks;

namespace FirstDemo.Infrastructure.Repositories
{
    public class CourseRepository : Repository<Course, Guid>, ICourseRepository
    {
        public CourseRepository(IApplicationDbContext context) : base((DbContext)context)
        {
        }

        public async Task<(IList<Course> records, int total, int totalDisplay)>
            GetTableDataAsync(string searchTitle, uint searchFeeFrom, uint searchFeeTo, string
orderBy, int pageIndex, int pageSize)
        {
            Expression<Func<Course, bool>> expression = null;
            if(!string.IsNullOrWhiteSpace(searchTitle))
            {
                expression = x => x.Title.Contains(searchTitle) &&
                (x.Fees >= searchFeeFrom && x.Fees <= searchFeeTo);
            }
            return await GetDynamicAsync(expression,
                orderBy, null, pageIndex, pageSize, true);
        }

        public async Task<bool> IsTitleDuplicateAsync(string title, Guid? id = null)
        {
            if(id.HasValue)
            {
                return (await GetCountAsync(x => x.Id != id.Value && x.Title == title)) > 0;
            }
            else
            {
                return (await GetCountAsync(x => x.Title == title)) > 0;
            }
        }
    }
}



===========================================================================



using FirstDemo.Domain;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstDemo.Infrastructure
{
    public abstract class UnitOfWork : IUnitOfWork
    {
        private readonly DbContext _dbContext;
        protected IAdoNetUtility AdoNetUtility { get; private set; }

        public UnitOfWork(DbContext dbContext)
        {
            _dbContext = dbContext;
            AdoNetUtility = new AdoNetUtility(_dbContext.Database.GetDbConnection());
        }
        public virtual void Dispose() => _dbContext?.Dispose();

        public virtual async ValueTask DisposeAsync() => await _dbContext.DisposeAsync();

        public virtual void Save() => _dbContext?.SaveChanges();

        public virtual async Task SaveAsync() => await _dbContext.SaveChangesAsync();
    }
}
using FirstDemo.Domain.Entities;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace FirstDemo.Infrastructure
{
    public class ApplicationDbContext : IdentityDbContext, IApplicationDbContext
    {
        private readonly string _connectionString;
        private readonly string _migrationAssembly;

        public ApplicationDbContext(string connectionString, string migrationAssembly)
        {
            _connectionString = connectionString;
            _migrationAssembly = migrationAssembly;
        }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            if (!optionsBuilder.IsConfigured)
            {
                optionsBuilder.UseSqlServer(_connectionString,
                    x => x.MigrationsAssembly(_migrationAssembly));
            }

            base.OnConfiguring(optionsBuilder);
        }

        protected override void OnModelCreating(ModelBuilder builder)
        {
            builder.Entity<CourseEnrollment>().ToTable("CourseEnrollments");
            builder.Entity<CourseEnrollment>().HasKey(x => new { x.CourseId, x.StudentId });

            builder.Entity<CourseEnrollment>()
                .HasOne<Course>()
                .WithMany()
                .HasForeignKey(x => x.CourseId);

            builder.Entity<CourseEnrollment>()
                            .HasOne<Student>()
                            .WithMany()
                            .HasForeignKey(x => x.StudentId);


            builder.Entity<Course>().HasData(new Course[]
            {
                new Course{ Id=new Guid("672b26ca-6a94-46a0-8296-5583a37c84d9"), Title = "Test Course
1", Description= " Test Description 1", Fees = 2000 },
                new Course{ Id=new Guid("7c47af1a-8fe1-4424-bce9-b002d606a86f"), Title = "Test Course
2", Description= " Test Description 2", Fees = 3000 }
            });
            base.OnModelCreating(builder);
        }
        public DbSet<Course> Courses { get; set; }
        public DbSet<Student> Students { get; set; }
    }
}


============================================================================



using FirstDemo.Domain.Entities;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace FirstDemo.Infrastructure
{
    public class ApplicationDbContext : IdentityDbContext, IApplicationDbContext
    {
        private readonly string _connectionString;
        private readonly string _migrationAssembly;

        public ApplicationDbContext(string connectionString, string migrationAssembly)
        {
            _connectionString = connectionString;
            _migrationAssembly = migrationAssembly;
        }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            if (!optionsBuilder.IsConfigured)
            {
                optionsBuilder.UseSqlServer(_connectionString,
                    x => x.MigrationsAssembly(_migrationAssembly));
            }

            base.OnConfiguring(optionsBuilder);
        }

        protected override void OnModelCreating(ModelBuilder builder)
        {
            builder.Entity<CourseEnrollment>().ToTable("CourseEnrollments");
            builder.Entity<CourseEnrollment>().HasKey(x => new { x.CourseId, x.StudentId });

            builder.Entity<CourseEnrollment>()
                .HasOne<Course>()
                .WithMany()
                .HasForeignKey(x => x.CourseId);

            builder.Entity<CourseEnrollment>()
                            .HasOne<Student>()
                            .WithMany()
                            .HasForeignKey(x => x.StudentId);


            builder.Entity<Course>().HasData(new Course[]
            {
                new Course{ Id=new Guid("672b26ca-6a94-46a0-8296-5583a37c84d9"), Title = "Test Course
1", Description= " Test Description 1", Fees = 2000 },
                new Course{ Id=new Guid("7c47af1a-8fe1-4424-bce9-b002d606a86f"), Title = "Test Course
2", Description= " Test Description 2", Fees = 3000 }
            });
            base.OnModelCreating(builder);
        }

        public DbSet<Course> Courses { get; set; }
        public DbSet<Student> Students { get; set; }
    }
}


===========================================================================


using FirstDemo.Application;
using FirstDemo.Application.Features.Training.DTOs;
using FirstDemo.Domain.Repositories;
using FirstDemo.Infrastructure.Repositories;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstDemo.Infrastructure
{
    public class ApplicationUnitOfWork : UnitOfWork, IApplicationUnitOfWork
    {
        public ICourseRepository CourseRepository { get; private set; }

        public ApplicationUnitOfWork(ICourseRepository courseRepository, IApplicationDbContext
dbContext) : base((DbContext)dbContext)
        {
            CourseRepository = courseRepository;
        }

        public async Task<(IList<CourseEnrollmentDTO> records,
            int total, int totalDisplay)> GetCourseEnrollmentsAsync(
            int pageIndex,
            int pageSize,
            string orderBy,
            string courseName,
            string studentName,
            DateTime enrollmentDateFrom,
            DateTime enrollmentDateTo)
        {
            var data = await AdoNetUtility.QueryWithStoredProcedureAsync<CourseEnrollmentDTO>(
                "GetCourseEnrollments",
                new Dictionary<string, object>
                {
                    { "PageIndex",  pageIndex},
                    { "PageSize",  pageSize },
                    { "OrderBy",  orderBy },
                    { "CourseName",  courseName},
                    { "StudentName",  studentName },
                    { "EnrollmentDateFrom",  enrollmentDateFrom},
                    { "EnrollmentDateTo",  enrollmentDateTo }
                },
                new Dictionary<string, Type>
                {
                    {"Total", typeof(int)},
                    {"TotalDisplay", typeof(int) }
                }
                );

            return (data.result, (int)data.outValues["Total"], (int)data.outValues["TotalDisplay"]);

        }
    }
}
```

```
===========================================================================



using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Primitives;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstDemo.Infrastructure
{
    public class DataTablesAjaxRequestUtility
    {
        private HttpRequest _request;

        public int Start
        {
            get
            {
                return int.Parse(RequestData.Where(x => x.Key == "start")
                    .FirstOrDefault().Value);
            }
        }
        public int Length
        {
            get
            {
                return int.Parse(RequestData.Where(x => x.Key == "length")
                    .FirstOrDefault().Value);
            }
        }

        public string SearchText
        {
            get
            {
                return RequestData.Where(x => x.Key == "search[value]")
                    .FirstOrDefault().Value;
            }
        }

        public DataTablesAjaxRequestUtility(HttpRequest request)
        {
            _request = request;
        }

        public int PageIndex
        {
            get
            {
                if (Length > 0)
                    return (Start / Length) + 1;
                else
                    return 1;
            }
        }

        public int PageSize
        {
            get
            {
                if (Length == 0)
                    return 10;
                else
                    return Length;
            }
        }

        private IEnumerable<KeyValuePair<string, StringValues>> RequestData
        {
            get
            {
                var method = _request.Method.ToLower();
                if (method == "get")
                    return _request.Query;
                else if (method == "post")
                    return _request.Form;
                else
                    throw new InvalidOperationException("Http method not supported, use get or
post");
            }
        }

        public static object EmptyResult
        {
            get
            {
                return new
                {
                    recordsTotal = 0,
                    recordsFiltered = 0,
                    data = (new string[] { }).ToArray()
                };
            }
        }

        public string GetSortText(string[] columnNames)
        {
            var sortText = new StringBuilder();
            for (var i = 0; i < columnNames.Length; i++)
            {
                if (RequestData.Any(x => x.Key == $"order[{i}][column]"))
                {
                    if (sortText.Length > 0)
                        sortText.Append(",");

                    var columnValue = RequestData.Where(x => x.Key == $"order[{i}]
[column]").FirstOrDefault();
                    var directionValue = RequestData.Where(x => x.Key == $"order[{i}]
[dir]").FirstOrDefault();

                    var column = int.Parse(columnValue.Value.ToArray()[0]);
                    var direction = directionValue.Value.ToArray()[0];
                    var sortDirection = $"{columnNames[column]} {(direction == "asc" ? "asc" :
"desc")}";
                    sortText.Append(sortDirection);
                }
            }
            return sortText.ToString();
        }
    }
}
```

```
============================================================================


using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstDemo.Infrastructure
{
    public interface IAdoNetUtility
    {
        Task<TReturn> ExecuteScalarAsync<TReturn>(string storedProcedureName, IDictionary<string,
object> parameters = null);
        IDictionary<string, object> ExecuteStoredProcedure(string storedProcedureName,
IDictionary<string, object> parameters = null, IDictionary<string, Type> outParameters = null);
        Task<IDictionary<string, object>> ExecuteStoredProcedureAsync(string storedProcedureName,
IDictionary<string, object> parameters = null, IDictionary<string, Type> outParameters = null);
        Task<(IList<TReturn> result, IDictionary<string, object> outValues)>
QueryWithStoredProcedureAsync<TReturn>(string storedProcedureName, IDictionary<string, object>
parameters = null, IDictionary<string, Type> outParameters = null) where TReturn : class, new();
    }
}



============================================================================


using FirstDemo.Domain.Entities;
using Microsoft.EntityFrameworkCore;

namespace FirstDemo.Infrastructure
{
    public interface IApplicationDbContext
    {
        DbSet<Course> Courses { get; set; }
    }
}


============================================================================



using Autofac;
using FirstDemo.Application;
using FirstDemo.Domain.Repositories;
using FirstDemo.Infrastructure.Repositories;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstDemo.Infrastructure
{
    public class InfrastructureModule : Module
    {
        private readonly string _connectionString;
        private readonly string _migrationAssembly;
        public InfrastructureModule(string connectionString, string migrationAssembly)
        {
            _connectionString = connectionString;
            _migrationAssembly = migrationAssembly;
        }

        protected override void Load(ContainerBuilder builder)
        {
            builder.RegisterType<ApplicationDbContext>().AsSelf()
                .WithParameter("connectionString", _connectionString)
                .WithParameter("migrationAssembly", _migrationAssembly)
                .InstancePerLifetimeScope();

            builder.RegisterType<ApplicationDbContext>().As<IApplicationDbContext>()
                .WithParameter("connectionString", _connectionString)
                .WithParameter("migrationAssembly", _migrationAssembly)
                .InstancePerLifetimeScope();

            builder.RegisterType<ApplicationUnitOfWork>().As<IApplicationUnitOfWork>()
                .InstancePerLifetimeScope();

            builder.RegisterType<CourseRepository>().As<ICourseRepository>()
                .InstancePerLifetimeScope();
        }
    }
}


===========================================================================


using Microsoft.AspNetCore.Mvc.ViewFeatures;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.Json;
using System.Threading.Tasks;

namespace FirstDemo.Infrastructure
{
    public static class TempDataExtensions
    {
        public static void Put<T>(this ITempDataDictionary tempData, string key, T Value) where T :
class
        {
            tempData[key] = JsonSerializer.Serialize(Value);

        }
        public static T Get<T>(this ITempDataDictionary tempData, string key) where T : class
        {
            object o;
            tempData.TryGetValue(key, out o);
            return o == null ? null : JsonSerializer.Deserialize<T>((string)o);

        }
        public static T Peek<T>(this ITempDataDictionary tempData, string key) where T : class
        {
            object o = tempData.Peek(key);
            return o == null ? null : JsonSerializer.Deserialize<T>((string)o);

        }
    }
}



===========================================================================
using Microsoft.Data.SqlClient;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Data.Common;
using System.Data.SqlTypes;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstDemo.Infrastructure
{
    public class AdoNetUtility : IAdoNetUtility
    {
        private readonly DbConnection _connection;
        private readonly int _timeout;

        public AdoNetUtility(DbConnection connection, int timeout = 0)
        {
            _connection = connection;
            _timeout = timeout;
        }

        public virtual IDictionary<string, object> ExecuteStoredProcedure(string storedProcedureName,
            IDictionary<string, object> parameters = null, IDictionary<string, Type> outParameters =
null)
        {
            var command = CreateCommand(storedProcedureName, parameters, outParameters);
            command = ConvertNullToDbNull(command);

            var connectionOpened = false;
            if (command.Connection.State == ConnectionState.Closed)
            {
                command.Connection.Open();
                connectionOpened = true;
            }

            DbTransaction transaction = command.Connection.BeginTransaction();

            try
            {
                command.ExecuteNonQuery();
                transaction.Commit();
            }
            catch(DbException dex)
            {
                transaction.Rollback();
            }

            return CopyOutParams(command, outParameters);
        }

        public virtual async Task<IDictionary<string, object>> ExecuteStoredProcedureAsync(string
storedProcedureName,
            IDictionary<string, object> parameters = null, IDictionary<string, Type> outParameters =
null)
        {
            var command = CreateCommand(storedProcedureName, parameters, outParameters);
            command = ConvertNullToDbNull(command);

            var connectionOpened = false;
            if (command.Connection.State == ConnectionState.Closed)
            {
                await command.Connection.OpenAsync();
                connectionOpened = true;
            }

            DbTransaction transaction = command.Connection.BeginTransaction();

            try
            {
                await command.ExecuteNonQueryAsync();
                await transaction.CommitAsync();
            }
            catch(DbException dex)
            {
               transaction.RollbackAsync();
            }

            return CopyOutParams(command, outParameters);
        }

        public virtual async Task<(IList<TReturn> result, IDictionary<string, object> outValues)>
            QueryWithStoredProcedureAsync<TReturn>(string storedProcedureName,
            IDictionary<string, object> parameters = null, IDictionary<string, Type> outParameters =
null)
            where TReturn : class, new()
        {
            var command = CreateCommand(storedProcedureName, parameters, outParameters);

            var connectionOpened = false;
            if (command.Connection.State == ConnectionState.Closed)
            {
                await command.Connection.OpenAsync();
                connectionOpened = true;
            }

            IList<TReturn> result = null;
            try
            {
                result = await ExecuteQueryAsync<TReturn>(command);
            }
            finally
            {
                if (connectionOpened)
                    await command.Connection.CloseAsync();
            }

            var outValues = CopyOutParams(command, outParameters);

            return (result, outValues);
        }

        public virtual async Task<TReturn> ExecuteScalarAsync<TReturn>(string storedProcedureName,
            IDictionary<string, object> parameters = null)
        {
            var command = CreateCommand(storedProcedureName, parameters);

            var connectionOpened = false;
            if (command.Connection.State == ConnectionState.Closed)
            {
                await command.Connection.OpenAsync();
                connectionOpened = true;
            }

            TReturn result;

            try
            {
                result = await ExecuteScalarAsync<TReturn>(command);
            }
            finally
            {
                if (connectionOpened)
                    await command.Connection.CloseAsync();
            }

            return result;
        }

        private DbCommand CreateCommand(string storedProcedureName,
            IDictionary<string, object> parameters = null,
            IDictionary<string, Type> outParameters = null)
        {
            var command = _connection.CreateCommand();
            command.CommandText = storedProcedureName;
            command.CommandType = CommandType.StoredProcedure;
            command.CommandTimeout = _timeout;

            if (parameters != null)
            {
                foreach (var item in parameters)
                {
                    command.Parameters.Add(CreateParameter(item.Key, item.Value));
                }
            }

            if (outParameters != null)
            {
                foreach (var item in outParameters)
                {
                    command.Parameters.Add(CreateOutputParameter(item.Key,
                        item.Value));
                }
            }

            return command;
        }

        private DbParameter CreateParameter(string name, object value)
        {
            return new SqlParameter(name, CorrectSqlDateTime(value));
        }

        private DbParameter CreateOutputParameter(string name, DbType dbType)
        {
            var outParam = new SqlParameter(name, CorrectSqlDateTime(dbType));
            outParam.Direction = ParameterDirection.Output;
            return outParam;
        }

        private DbParameter CreateOutputParameter(string name, Type type)
        {
            var outParam = new SqlParameter(name, GetDbTypeFromType(type));
            outParam.Direction = ParameterDirection.Output;
            return outParam;
        }

        private SqlDbType GetDbTypeFromType(Type type)
        {
            if (type == typeof(int) ||
                type == typeof(uint) ||
                type == typeof(short) ||
                type == typeof(ushort))
                return SqlDbType.Int;
            else if (type == typeof(long) || type == typeof(ulong))
                return SqlDbType.BigInt;
            else if (type == typeof(double) || type == typeof(decimal))
                return SqlDbType.Decimal;
            else if (type == typeof(string))
                return SqlDbType.NVarChar;
            else if (type == typeof(DateTime))
                return SqlDbType.DateTime;
            else if (type == typeof(bool))
                return SqlDbType.Bit;
            else if (type == typeof(Guid))
                return SqlDbType.UniqueIdentifier;
            else if (type == typeof(char))
                return SqlDbType.NVarChar;
            else
                return SqlDbType.NVarChar;
        }

        private object ChangeType(Type propertyType, object itemValue)
        {
            if (itemValue is DBNull)
                return null;

            return itemValue is decimal && propertyType == typeof(double) ?
                Convert.ToDouble(itemValue) : itemValue;
        }

        private object CorrectSqlDateTime(object parameterValue)
        {
            if (parameterValue != null && parameterValue.GetType().Name == "DateTime")
            {
                if (Convert.ToDateTime(parameterValue) < SqlDateTime.MinValue.Value)
                    return SqlDateTime.MinValue.Value;
                else
                    return parameterValue;
            }
            else
                return parameterValue;
        }

        private async Task<IList<TReturn>> ExecuteQueryAsync<TReturn>(DbCommand command)
        {
            var reader = await command.ExecuteReaderAsync();
            var result = new List<TReturn>();

            while (await reader.ReadAsync())
            {
                var type = typeof(TReturn);
                var constructor = type.GetConstructor(new Type[] { });
                if (constructor == null)
                    throw new InvalidOperationException("An empty contructor is required for the
return type");

                var instance = constructor.Invoke(new object[] { });

                for (var i = 0; i < reader.FieldCount; i++)
                {
                    var property = type.GetProperty(reader.GetName(i));
                    property?.SetValue(instance, ChangeType(property.PropertyType,
reader.GetValue(i)));
                }

                result.Add((TReturn)instance);
            }

            return result;
        }

        private async Task<TReturn> ExecuteScalarAsync<TReturn>(DbCommand command)
        {
            command = ConvertNullToDbNull(command);

            if (command.Connection.State != ConnectionState.Open)
                command.Connection.Open();

            var result = await command.ExecuteScalarAsync();

            if (result == DBNull.Value)
                return default;
            else
                return (TReturn)result;
        }

        private DbCommand ConvertNullToDbNull(DbCommand command)
        {
            for (int i = 0; i < command.Parameters.Count; i++)
            {
                if (command.Parameters[i].Value == null)
                    command.Parameters[i].Value = DBNull.Value;
            }

            return command;
        }

        private IDictionary<string, object> CopyOutParams(DbCommand command,
            IDictionary<string, Type> outParameters)
        {
            Dictionary<string, object> result = null;
            if (outParameters != null)
            {
                result = new Dictionary<string, object>();
                foreach (var item in outParameters)
                {
                    result.Add(item.Key, command.Parameters[item.Key].Value);
                }
            }

            return result;
        }
    }
}



===========================================================================


using Autofac;
using FirstDemo.Application.Features.Training;
using FirstDemo.Application.Features.Training.Services;

namespace FirstDemo.Web.Areas.Admin.Models
{
    public class CourseCreateModel
    {
        private ILifetimeScope _scope;
        private ICourseManagementService _courseManagementService;
        public string Title { get; set; }
        public string Description { get; set; }
        public uint Fees { get; set; }

        public CourseCreateModel() { }

        public CourseCreateModel(ICourseManagementService courseManagementService)
        {
            _courseManagementService = courseManagementService;
        }

        internal void Resolve(ILifetimeScope scope)
        {
            _scope = scope;
            _courseManagementService = _scope.Resolve<ICourseManagementService>();
        }

        internal async Task CreateCourseAsync()
        {
            await _courseManagementService.CreateCourseAsync(Title, Description, Fees);
        }
    }
}



===========================================================================



namespace FirstDemo.Web.Areas.Admin.Models
{
    public class CourseEnrollmentSearch
    {
        public string CourseName { get; set; }
        public string StudentName { get; set; }
        public DateTime EnrollmentDateFrom { get; set; }
        public DateTime EnrollmentDateTo { get; set; }
    }
}


===========================================================================




using Autofac;
using FirstDemo.Application.Features.Training.Services;
using System.Web;

namespace FirstDemo.Web.Areas.Admin.Models
{
    public class CourseEnrollmentListModel
    {
        private ILifetimeScope _scope;
        private ICourseManagementService _courseManagementService;

        public CourseEnrollmentSearch SearchItem { get; set; }

        public CourseEnrollmentListModel()
        {
        }

        public CourseEnrollmentListModel(ICourseManagementService courseManagementService)
        {
            _courseManagementService = courseManagementService;
        }

        public void Resolve(ILifetimeScope scope)
        {
            _scope = scope;
            _courseManagementService = _scope.Resolve<ICourseManagementService>();
        }

        public async Task<object> GetPagedCourseEnrollmentsAsync(int pageIndex, int pageSize, string
orderBy)
        {
            var data = await _courseManagementService.GetCourseEnrollmentsAsync(
                pageIndex,
                pageSize,
                orderBy,
                SearchItem.CourseName,
                SearchItem.StudentName,
                SearchItem.EnrollmentDateFrom,
                SearchItem.EnrollmentDateTo
                );

            return new
            {
                recordsTotal = data.total,
                recordsFiltered = data.totalDisplay,
                data = (from record in data.records
                        select new string[]
                        {
                                HttpUtility.HtmlEncode(record.StudentName),
                                HttpUtility.HtmlEncode(record.CourseName),
                                record.EnrollmentDate.ToString()
                        }
                    ).ToArray()
            };
        }
    }
}

===========================================================================


using Autofac;
using FirstDemo.Application.Features.Training;
using static System.Formats.Asn1.AsnWriter;
using FirstDemo.Infrastructure;
using System.Web;
using FirstDemo.Application.Features.Training.Services;

namespace FirstDemo.Web.Areas.Admin.Models
{
    public class CourseListModel
    {
        private ILifetimeScope _scope;

        private  ICourseManagementService _courseManagementService;
        public CourseSearch SearchItem { get; set; }

        public CourseListModel()
        {
        }

        public CourseListModel(ICourseManagementService courseService)
        {
            _courseManagementService = courseService;
        }

                public void Resolve(ILifetimeScope scope)
                {
                        _scope = scope;
            _courseManagementService = _scope.Resolve<ICourseManagementService>();
                }

                public async Task<object> GetPagedCoursesAsync(DataTablesAjaxRequestUtility
dataTablesUtility)
        {
            var data = await _courseManagementService.GetPagedCoursesAsync(
                dataTablesUtility.PageIndex,
                dataTablesUtility.PageSize,
                SearchItem.Title,
                SearchItem.CourseFeeFrom,
                SearchItem.CourseFeeTo,
                dataTablesUtility.GetSortText(new string[] { "Title", "Description", "Fees" }));

            return new
            {
                recordsTotal = data.total,
                recordsFiltered = data.totalDisplay,
                data = (from record in data.records
                        select new string[]
                        {
                                HttpUtility.HtmlEncode(record.Title),
                                HttpUtility.HtmlEncode(record.Description),
                                record.Fees.ToString(),
                                record.Id.ToString()
                        }
                    ).ToArray()
            };
        }

        internal async Task DeleteCourseAsync(Guid id)
        {
            await _courseManagementService.DeleteCourseAsync(id);
        }
    }
}



===========================================================================


using Autofac;
using FirstDemo.Web.Areas.Admin.Models;
using FirstDemo.Web.Models;
using System.Reflection;
using Module = Autofac.Module;

namespace FirstDemo.Web
{
    public class WebModule : Module
    {
        protected override void Load(ContainerBuilder builder)
        {
            builder.RegisterType<UnicodeSmsSender>().As<ISmsSender>();
            builder.RegisterType<CourseCreateModel>().AsSelf();
            builder.RegisterType<CourseUpdateModel>().AsSelf();
            builder.RegisterType<CourseListModel>().AsSelf();
        }
    }
```

```
}



===========================================================================



using AutoMapper;
using FirstDemo.Domain.Entities;
using FirstDemo.Web.Areas.Admin.Models;

namespace FirstDemo.Web
{
    public class WebProfile : Profile
    {
        public WebProfile()
        {
            CreateMap<CourseUpdateModel, Course>()
                .ReverseMap();
        }
    }
}


===========================================================================


using Autofac.Extensions.DependencyInjection;
using Autofac;
using FirstDemo.Web.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using FirstDemo.Web;
using Serilog;
using Serilog.Events;
using FirstDemo.Application;
using FirstDemo.Infrastructure;
using System.Reflection;

var builder = WebApplication.CreateBuilder(args);
builder.Host.UseSerilog((ctx, lc) => lc
    .MinimumLevel.Debug()
    .MinimumLevel.Override("Microsoft", LogEventLevel.Warning)
    .Enrich.FromLogContext()
    .ReadFrom.Configuration(builder.Configuration));

try
{
    var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw
new InvalidOperationException("Connection string 'DefaultConnection' not found.");
    var migrationAssembly = Assembly.GetExecutingAssembly().FullName;

    builder.Host.UseServiceProviderFactory(new AutofacServiceProviderFactory());
    builder.Host.ConfigureContainer<ContainerBuilder>(containerBuilder =>
    {
        containerBuilder.RegisterModule(new ApplicationModule());
        containerBuilder.RegisterModule(new InfrastructureModule(connectionString,
            migrationAssembly));
        containerBuilder.RegisterModule(new WebModule());
    });


    // Add services to the container.
    builder.Services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(connectionString,
        (m) => m.MigrationsAssembly(migrationAssembly)));

    builder.Services.AddDatabaseDeveloperPageExceptionFilter();
    builder.Services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies());

    builder.Services.AddDefaultIdentity<IdentityUser>(options =>
options.SignIn.RequireConfirmedAccount = true)
        .AddEntityFrameworkStores<ApplicationDbContext>();
    builder.Services.AddControllersWithViews();
    builder.Services.AddScoped<IEmailSender, HtmlEmailSender>();

    var app = builder.Build();

    // Configure the HTTP request pipeline.
    if (app.Environment.IsDevelopment())
    {
        app.UseMigrationsEndPoint();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        // The default HSTS value is 30 days. You may want to change this for production scenarios,
see https://aka.ms/aspnetcore-hsts.
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();



    app.MapControllerRoute(
        name: "areas",
        pattern: "{area:exists}/{controller=Home}/{action=Index}/{id?}");

    app.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");

    app.MapRazorPages();

    app.Run();

    Log.Information("Application Starting...");
}
catch (Exception ex)
{
    Log.Fatal(ex, "Failed to start application.");
}
finally
{
    Log.CloseAndFlush();
}



===========================================================================
using Autofac;
using FirstDemo.Domain.Exceptions;
using FirstDemo.Infrastructure;
using FirstDemo.Web.Areas.Admin.Models;
using Microsoft.AspNetCore.Mvc;

namespace FirstDemo.Web.Areas.Admin.Controllers
{
    [Area("Admin")]
    public class CourseController : Controller
    {
        private readonly ILifetimeScope _scope;
        private readonly ILogger<CourseController> _logger;

        public CourseController(ILifetimeScope scope,
            ILogger<CourseController> logger)
        {
            _scope = scope;
            _logger = logger;
        }

        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Create()
        {
            var model = _scope.Resolve<CourseCreateModel>();
            return View(model);
        }

        [HttpPost, ValidateAntiForgeryToken]
        public async Task<IActionResult> Create(CourseCreateModel model)
        {
            if (ModelState.IsValid)
            {
                try
                {
                    model.Resolve(_scope);
                    await model.CreateCourseAsync();
                    TempData.Put("ResponseMessage", new ResponseModel
                    {
                        Message = "Course Created successfully",
                        Type = ResponseTypes.Success
                    });

                    return RedirectToAction("Index");
                }
                catch(DuplicateTitleException de)
                {
                    TempData.Put("ResponseMessage", new ResponseModel
                    {
                        Message = de.Message,
                        Type = ResponseTypes.Danger
                    });
                }
                catch (Exception ex)
                {
                    _logger.LogError(ex, "Failed to create Course.");
                    _logger.LogError(ex, "Server Error.");
                    TempData.Put("ResponseMessage", new ResponseModel
                    {
                        Message = "There was a problem in creating course",
                        Type = ResponseTypes.Danger
                    });
                }
            }


            return View(model);
        }

        [HttpPost]
        public async Task<JsonResult> GetCourses(CourseListModel model)
        {
            var dataTablesModel = new DataTablesAjaxRequestUtility(Request);
            model.Resolve(_scope);

            var data = await model.GetPagedCoursesAsync(dataTablesModel);
            return Json(data);
        }

        public async Task<JsonResult> GetCourseEnrollments()
        {
            CourseEnrollmentListModel model = new();
            model.Resolve(_scope);
            model.SearchItem = new CourseEnrollmentSearch
            {
                CourseName = "C#",
                StudentName = "JalalUddin",
                EnrollmentDateFrom = new DateTime(2020, 1, 1),
                EnrollmentDateTo = new DateTime(2030, 2, 2)
            };

            var data = await model.GetPagedCourseEnrollmentsAsync(1, 10, "CourseName");
            return Json(data);
        }

        public async Task<IActionResult> Update(Guid id)
        {
            var model = _scope.Resolve<CourseUpdateModel>();
            await model.LoadAsync(id);
            return View(model);
        }

        [HttpPost, ValidateAntiForgeryToken]
        public async Task<IActionResult> Update(CourseUpdateModel model)
        {
            model.Resolve(_scope);
            if (ModelState.IsValid)
            {
                try
                {
                    await model.UpdateCourseAsync();
                                        TempData.Put("ResponseMessage", new ResponseModel
                                        {
                                                Message = "Updating Course Succesfully",
                                                Type = ResponseTypes.Success
                                        });
                                        return RedirectToAction("Index");
                }
                catch (DuplicateTitleException de)
                {
                    TempData.Put("ResponseMessage", new ResponseModel
                    {
                        Message = de.Message,
                        Type = ResponseTypes.Danger
                    });
                }
```

```
                catch (Exception e)
                {
                    _logger.LogError(e, "Server Error");
                    _logger.LogError($"{e.Message}: Server Error", e);
                    TempData.Put("ResponseMessage", new ResponseModel
                    {
                        Message = "There is a problem in updating Course",
                        Type = ResponseTypes.Danger
                    });
                }
            }

            return View(model);
        }

        [HttpPost, ValidateAntiForgeryToken]
        public async Task<IActionResult> Delete(Guid id)
        {
            var model = _scope.Resolve<CourseListModel>();
            if (ModelState.IsValid)
            {
                try
                {
                    await model.DeleteCourseAsync(id);
                    TempData.Put("ResponseMessage", new ResponseModel
                    {
                        Message = "Course deleted successfully",
                        Type = ResponseTypes.Success
                    });

                    return RedirectToAction("Index");

                }
                catch (Exception e)
                {
                    _logger.LogError(e, "Server Error");
                    _logger.LogError($"{e.Message}: Server Error", e);

                    TempData.Put("ResponseMessage", new ResponseModel
                    {
                        Message = "There is a problem in creating course",
                        Type = ResponseTypes.Danger
                    });

                }

            }

            return RedirectToAction("Index");

        }

    }
}


============================================================================
```