ДОДАТОК А

Вихідний код програми

```
using System.Globalization;
using System.Linq;
using System.Threading.Tasks;
using System.Web.Mvc;

using Smartify.DAL.Infrastructure;
using Smartify.DAL.UnitOfWork;
using Smartify.Entities.Models;
using Smartify.Services.Interfaces;
using Smartify.UI.Models;

namespace Smartify.UI.Controllers
{
    [Authorize]
    public class GoalController : Controller
    {
        private readonly IUnitOfWorkAsync _unitOfWork;
        private readonly IGoalService _goalService;
        private readonly ICategoryService _categoryService;

        public GoalController(
            IUnitOfWorkAsync unitOfWork,
            IGoalService goalService,
            ICategoryService categoryService)
        {
            _unitOfWork = unitOfWork;
            _goalService = goalService;
            _categoryService = categoryService;
        }

        // GET: Goal
        [AllowAnonymous]
        public ActionResult Index()
        {
            var goals = _goalService.Queryable().ToList();
```

```
            return View(goals);
        }

        public ActionResult Details(int? id)
        {
            if (!id.HasValue)
            {
                return this.View();
            }

            var goal = _goalService.Query(x => x.Id == id)
                .Select()
                .FirstOrDefault();

            return this.View(goal);
        }

        [HttpGet]
        public ActionResult Add()
        {
            var                  categories                  =
_categoryService.Queryable().ToList();
            var categoryListItems =
                categories.Select(
                    x => new SelectListItem { Text = x.Title,
Value = x.Id.ToString(CultureInfo.InvariantCulture) })
                    .ToList();

            var createVm = new CreateGameViewModel { Categories
= categoryListItems };
            return View(createVm);
        }

        [HttpPost]
        public async Task<ActionResult> Add(CreateGameViewModel
goalVm)
        {
            if (goalVm == null)
```

```
                {
                    return this.View();
                }

                var goal = новий Goal
                            {
                                Title = goalVm.Title,
                                Description                    =
goalVm.Description,
                                CategoryId = goalVm.CategoryId,
                                ObjectState = ObjectState.Added
                            };

                _goalService.Insert(goal);

                await _unitOfWork.SaveChangesAsync();

                return RedirectToAction(«Index»);
            }

            [HttpGet]
            public ActionResult Edit(int? id)
            {
                if (!id.HasValue)
                {
                    return RedirectToAction(«Index»);
                }

                var                  categories                  =
_categoryService.Queryable().ToList();
                var categoryListItems =
                    categories.Select(
                        x => new SelectListItem { Text = x.Title,
Value = x.Id.ToString(CultureInfo.InvariantCulture) })
                        .ToList();

                var goal = _goalService.Query(x => x.Id == id)
                   .Select()
                   .FirstOrDefault();
```

```
                if (goal == null)
                {
                    return this.View();
                }

                var createVm = New CreateGameViewModel
                                {
                                    Categories                  =
categoryListItems,
                                    Title = goal.Title,
                                    Description = goal.
                                    CategoryId = goal.CategoryId
                                };

                return this.View(createVm);
            }

            [HttpPost]
            public async Task<ActionResult> Edit(Goal goal)
            {
                if (goal == null)
                {
                    return RedirectToAction(«Index»);
                }

                goal.ObjectState = ObjectState.Modified;
                _goalService.Update(goal);

                await _unitOfWork.SaveChangesAsync();

                return RedirectToAction(«Index»);
            }

            [HttpGet]
            public ActionResult Delete(int? id)
            {
                if (!id.HasValue)
                {
```

```
                return RedirectToAction(«Index»);
            }

            var goal = _goalService.Query(x => x.Id == id)
                .Select()
                .FirstOrDefault();

            return View(goal);
        }

        [HttpPost, ActionName(«Delete»)]
        public async Task<ActionResult> DeleteConfirmed(int? id)
        {
            if (!id.HasValue)
            {
                return RedirectToAction(«Index»);
            }

            var goal = _goalService.Query(x => x.Id == id)
              .Select()
              .FirstOrDefault();

            if (goal == null)
            {
                return RedirectToAction(«Index»);
            }

            goal.ObjectState = ObjectState.Deleted;

            _goalService.Delete(goal);
            await _unitOfWork.SaveChangesAsync();

return RedirectToAction(«Index»);
        }
    }
}


using System.Data.Entity;
```

```
using Smartify.DAL;
using Smartify.Entities.Models;
using Smartify.Entities.Models.Mapping;

namespace Smartify.Entities
{
    public class SmartifyContext : DataContext
    {
        public SmartifyContext()
            : base(«SmartifyContext»)
        {
        }

        public DbSet<Goal> Goals { get; set; }
        public DbSet<Category> Categories { get; set; }
        public DbSet<User> Users { get; set; }
        public DbSet<Role> Roles { get; set; }

        protected override void OnModelCreating(DbModelBuilder
modelBuilder)
        {
            modelBuilder.Configurations.Add(new GoalMap());
            modelBuilder.Configurations.Add(new CategoryMap());
            modelBuilder.Configurations.Add(new UserMap());
            modelBuilder.Configurations.Add(new RoleMap());
}
    }
}

using System;
використовуючи System.Data;

using Smartify.DAL.Infrastructure;
using Smartify.DAL.Repositories;

namespace Smartify.DAL.UnitOfWork
{
    public interface IUnitOfWork : IDisposable
```

```
        {
            int SaveChanges();
            void Dispose(bool disposing);
            IRepository<TEntity> Repository<TEntity>() where TEntity
: class, IObjectState;
            void BeginTransaction(IsolationLevel isolationLevel =
IsolationLevel.Unspecified);
    bool Commit();
            void Rollback ();
        }
    }

    використовуючи System.Threading;
    використовуючи System.Threading.Tasks;

    using Smartify.DAL.Infrastructure;
    using Smartify.DAL.Repositories;

    namespace Smartify.DAL.UnitOfWork
    {
        public interface IUnitOfWorkAsync : IUnitOfWork
        {
            Task<int> SaveChangesAsync();
            Task<int>            SaveChangesAsync(CancellationToken
cancellationToken);
            IRepositoryAsync<TEntity>    RepositoryAsync<TEntity>()
where TEntity : class, IObjectState;
    }
    }

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

    using Smartify.DAL.Infrastructure;

    namespace Smartify.DAL.Repositories
    {
```

```
        public interface IRepository<TEntity> where TEntity : class,
IObjectState
        {
            TEntity Find(params object[] keyValues);
            IQueryable<TEntity> SelectQuery(string query, params
object[] parameters);
            void Insert(TEntity entity);
            void InsertRange(IEnumerable<TEntity> entities);
            void InsertOrUpdateGraph(TEntity entity);
            void InsertGraphRange(IEnumerable<TEntity> entities);
            void Update (TEntity entity);
            void Delete (object id);
            void Delete (TEntity entity);
            IQueryFluent<TEntity>       Query(IQueryObject<TEntity>
queryObject);
            IQueryFluent<TEntity>    Query(Expression<Func<TEntity,
bool>> query);
            IQueryFluent<TEntity> Query();
            IQueryable<TEntity> Queryable();
            IRepository<T> GetRepository<T>() where T : class,
IObjectState;
    }
    }

    використовуючи System.Threading;
    використовуючи System.Threading.Tasks;

    using Smartify.DAL.Infrastructure;

    namespace Smartify.DAL.Repositories
    {
        public      interface      IRepositoryAsync<TEntity>      :
IRepository<TEntity> where TEntity : class, IObjectState
        {
            Task<TEntity> FindAsync(params object[] keyValues);
            Task<TEntity>               FindAsync(CancellationToken
cancellationToken, params object[] keyValues);
            Task<bool> DeleteAsync(params object[] keyValues);
```

```
            Task<bool>                DeleteAsync(CancellationToken
cancellationToken, params object[] keyValues);
      }
      }

      using System;
      використовуючи System.Collections.Generic;
      використовуючи System.Linq;
      використовуючи System.Linq.Expressions;
      використовуючи System.Threading.Tasks;

      using Smartify.DAL.Infrastructure;

      namespace Smartify.DAL.Repositories
      {
          public  interface  IQueryFluent<TEntity>  where  TEntity  :
IObjectState
          {
              IQueryFluent<TEntity> OrderBy(Func<IQueryable<TEntity>,
IOrderedQueryable<TEntity>> orderBy);
              IQueryFluent<TEntity>  Include(Expression<Func<TEntity,
object>> expression);
              IEnumerable<TEntity> SelectPage(int page, int pageSize,
out int totalCount);
              IEnumerable<TResult>
Select<TResult>(Expression<Func<TEntity, TResult>> selector = null);
              IEnumerable<TEntity> Select();
              Task<IEnumerable<TEntity>> SelectAsync();
              IQueryable<TEntity>   SqlQuery(string   query,   params
object[] parameters);
      }
      }

      using System;
      використовуючи System.Linq.Expressions;

      namespace Smartify.DAL.Repositories
      {
          public interface IQueryObject<TEntity>
```

```
        {
            Expression<Func<TEntity, bool>> Query();
            Expression<Func<TEntity,                          bool>>
And(Expression<Func<TEntity, bool>> query);
            Expression<Func<TEntity,                          bool>>
Or(Expression<Func<TEntity, bool>> query);
            Expression<Func<TEntity,                          bool>>
And(IQueryObject<TEntity> queryObject);
            Expression<Func<TEntity,                          bool>>
Or(IQueryObject<TEntity> queryObject);
     }
     }

     using System;

     using Smartify.DAL.Infrastructure;

     namespace Smartify.DAL.DataContexts
     {
         public interface IDataContext: IDisposable
         {
             int SaveChanges();
             void  SyncObjectState<TEntity>(TEntity  entity)  where
TEntity : class, IObjectState;
     void SyncObjectsStatePostCommit();
         }
     }

     використовуючи System.Threading;
     використовуючи System.Threading.Tasks;

     namespace Smartify.DAL.DataContexts
     {
         public interface IDataContextAsync : IDataContext
         {
             Task<int>            SaveChangesAsync(CancellationToken
cancellationToken);
     Task<int> SaveChangesAsync();
         }
```

```
}

using System;
using System.Data.Entity;
використовуючи System.Threading;
використовуючи System.Threading.Tasks;

using Smartify.DAL.DataContexts;
using Smartify.DAL.Infrastructure;

namespace Smartify.DAL
{
    public class DataContext : DbContext, IDataContextAsync
    {
         #region Private Fields
        private readonly Guid _instanceId;
        bool _disposed;
        #endregion Private Fields

        public DataContext(string nameOrConnectionString)
            : base(nameOrConnectionString)
        {
            _instanceId = Guid.NewGuid();
            Configuration.LazyLoadingEnabled = false;
            Configuration.ProxyCreationEnabled = false;

            Configuration.AutoDetectChangesEnabled = true;
        }

        public Guid InstanceId { get { return _instanceId; } }

        public override int SaveChanges()
        {
            SyncObjectsStatePreCommit();
            var changes = base.SaveChanges();
            SyncObjectsStatePostCommit();
            return changes;
        }
```

```
            public override async Task<int> SaveChangesAsync()
            {
                return                                          await
this.SaveChangesAsync(CancellationToken.None);
            }

            public         override         async         Task<int>
SaveChangesAsync(CancellationToken cancellationToken)
            {
                SyncObjectsStatePreCommit();
                var          changesAsync          =          await
base.SaveChangesAsync(cancellationToken);
                SyncObjectsStatePostCommit();
                return changesAsync;
            }

            public  void  SyncObjectState<TEntity>(TEntity  entity)
where TEntity : class, IObjectState
            {
                Entry(entity).State                                =
StateHelper.ConvertState(entity.ObjectState);
            }

            private void SyncObjectsStatePreCommit()
            {
                foreach         (var         dbEntityEntry         in
ChangeTracker.Entries())
                {
                    dbEntityEntry.State                            =
StateHelper.ConvertState(((IObjectState)dbEntityEntry.Entity).Object
State);
                }
            }

            public void SyncObjectsStatePostCommit()
            {
                foreach         (var         dbEntityEntry         in
ChangeTracker.Entries())
                {
```

```
((IObjectState)dbEntityEntry.Entity).ObjectState                    =
StateHelper.ConvertState(dbEntityEntry.State);
                }
            }

            protected override void Dispose(bool disposing)
            {
                if (!_disposed)
                {
                    if (disposing)
                    {

                    }

                    _disposed = true;
    }

                base.Dispose(disposing);
            }
        }
    }

    використовуючи System.ComponentModel.DataAnnotations.Schema;

    using Smartify.DAL.Infrastructure;

    namespace Smartify.DAL
    {
        Public class Entity : IObjectState
        {
            [NotMapped]
            public ObjectState ObjectState { get; set; }
    }
    }

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

```
використовуючи System.Linq.Expressions;
використовуючи System.Threading.Tasks;

using Smartify.DAL.Infrastructure;
using Smartify.DAL.Repositories;

namespace Smartify.DAL
{
    public sealed class QueryFluent<TEntity> : IQueryFluent<TEntity> where TEntity : class, IObjectState
    {
        #region Private Fields
        private readonly Expression<Func<TEntity, bool>> _expression;
        private readonly List<Expression<Func<TEntity, object>>> _includes;
        private readonly Repository<TEntity> _repository;
        private Func<IQueryable<TEntity>, IOrderedQueryable<TEntity>> _orderBy;
        #endregion Private Fields

        #region Constructors
        public QueryFluent(Repository<TEntity> repository)
        {
            _repository = repository;
            _includes = new List<Expression<Func<TEntity, object>>>();
        }

        public QueryFluent(Repository<TEntity> repository, IQueryObject<TEntity> queryObject) : this(repository) { _expression = queryObject.Query(); }

        public QueryFluent(Repository<TEntity> repository, Expression<Func<TEntity, bool>> expression) : this(repository) { _expression = expression }
        #endregion Constructors
```

```
            public                              IQueryFluent<TEntity>
OrderBy(Func<IQueryable<TEntity>,         IOrderedQueryable<TEntity>>
orderBy)
            {
                _orderBy = orderBy;
                return this;
            }

            public                              IQueryFluent<TEntity>
Include(Expression<Func<TEntity, object>> expression)
            {
                _includes.Add(expression);
                return this;
            }

            Public  IEnumerable<TEntity>  SelectPage(int  page,  int
pageSize, out int totalCount)
            {
                totalCount                                          =
_repository.Select(_expression).Count();
                return  _repository.Select(_expression,  _orderBy,
_includes, page, pageSize);
            }

            public   IEnumerable<TEntity>   Select()   {   return
_repository.Select(_expression, _orderBy, _includes);

            public                                IEnumerable<TResult>
Select<TResult>(Expression<Func<TEntity, TResult>> selector) { return
_repository.Select(_expression,                                 _orderBy,
_includes).Select(selector); }

            public async Task<IEnumerable<TEntity>> SelectAsync() {
return    await    _repository.SelectAsync(_expression,    _orderBy,
_includes); }

            public IQueryable<TEntity> SqlQuery(string query, params
object[]   parameters)   {   return   _repository.SelectQuery(query,
parameters).AsQueryable(); }
```

```
    }
    }

    using System;
    використовуючи System.Linq.Expressions;

    використовуючи LinqKit;

    using Smartify.DAL.Repositories;

    namespace Smartify.DAL
    {
        public abstract class QueryObject<TEntity> :
IQueryObject<TEntity>
        {
            private Expression<Func<TEntity, bool>> _query;

            public virtual Expression<Func<TEntity, bool>> Query()
            {
                return _query;
            }

            public Expression<Func<TEntity, bool>>
And(Expression<Func<TEntity, bool>> query)
            {
                return _query = _query == null? query :
_query.And(query.Expand());
            }

            public Expression<Func<TEntity, bool>>
Or(Expression<Func<TEntity, bool>> query)
            {
                return _query = _query == null? query :
_query.Or(query.Expand());
            }

            public Expression<Func<TEntity, bool>>
And(IQueryObject<TEntity> queryObject)
            {
```

```
                return And(queryObject.Query());
            }

            public          Expression<Func<TEntity,          bool>>
Or(IQueryObject<TEntity> queryObject)
    {
                return Or(queryObject.Query());
            }
        }
    }

    using System;
    використовуючи System.Collections.Generic;
    using System.Data.Entity;
    використовуючи System.Diagnostics;
    використовуючи System.Linq;
    використовуючи System.Linq.Expressions;
    використовуючи System.Threading;
    використовуючи System.Threading.Tasks;

    використовуючи LinqKit;

    using Smartify.DAL.DataContexts;
    using Smartify.DAL.Infrastructure;
    using Smartify.DAL.Repositories;
    using Smartify.DAL.UnitOfWork;

    namespace Smartify.DAL
    {
        public class Repository<TEntity> : IRepositoryAsync<TEntity>
where TEntity : class, IObjectState
        {
            #region Private Fields

            private readonly IDataContextAsync _context;
            private readonly DbSet<TEntity> _dbSet;
            private readonly IUnitOfWorkAsync _unitOfWork;

            #endregion Private Fields
```

```
            public      Repository(IDataContextAsync      context,
IUnitOfWorkAsync unitOfWork)
            {
                _context = context;
                _unitOfWork = unitOfWork;

                var dbContext = context as DbContext;

                if (dbContext! = null)
                {
                    _dbSet = dbContext.Set<TEntity>();
                }
            }

            Public virtual TEntity Find(params object[] keyValues)
            {
                return _dbSet.Find(keyValues);
            }

            public  virtual  IQueryable<TEntity>  SelectQuery(string
query, params object[] parameters)
            {
                return                          _dbSet.SqlQuery(query,
parameters).AsQueryable();
            }

            Public virtual void Insert(TEntity entity)
            {
                entity.ObjectState = ObjectState.Added;
                _dbSet.Attach(entity);
                _context.SyncObjectState(entity);
            }

            public  virtual  void  InsertRange(IEnumerable<TEntity>
entities)
            {
                foreach (var entity in entities)
                {
```

```
                Insert(entity);
            }
        }

        public virtual void
InsertGraphRange(IEnumerable<TEntity> entities)
        {
            _dbSet.AddRange(entities);
        }

        Public virtual void Update(TEntity entity)
        {
            entity.ObjectState = ObjectState.Modified;
            _dbSet.Attach(entity);
            _context.SyncObjectState(entity);
        }

        public virtual void Delete(object id)
        {
            var entity = _dbSet.Find(id);
            Delete (entity);
        }

        public virtual void Delete(TEntity entity)
        {
            entity.ObjectState = ObjectState.Deleted;
            _dbSet.Attach(entity);
            _context.SyncObjectState(entity);
        }

        public IQueryFluent<TEntity> Query()
        {
            return new QueryFluent<TEntity>(this);
        }

        public virtual IQueryFluent<TEntity>
Query(IQueryObject<TEntity> queryObject)
        {
            return new QueryFluent<TEntity>(this, queryObject);
```

```
        }

        public virtual IQueryFluent<TEntity>
Query(Expression<Func<TEntity, bool>> query)
        {
            return new QueryFluent<TEntity>(this, query);
        }

        public IQueryable<TEntity> Queryable()
        {
            return _dbSet;
        }

        public IRepository<T> GetRepository<T>() where T :
class, IObjectState
        {
            return _unitOfWork.Repository<T>();
        }

        public virtual async Task<TEntity> FindAsync(params
object[] keyValues)
        {
            return await _dbSet.FindAsync(keyValues);
        }

        public virtual async Task<TEntity>
FindAsync(CancellationToken cancellationToken, params object[]
keyValues)
        {
            return await _dbSet.FindAsync(cancellationToken,
keyValues);
        }

        public virtual async Task<bool> DeleteAsync(params
object[] keyValues)
        {
            return await DeleteAsync(CancellationToken.None,
keyValues);
        }
```

```
            public virtual async Task<bool>
DeleteAsync(CancellationToken cancellationToken, params object[]
keyValues)
            {
                var entity = await FindAsync(cancellationToken,
keyValues);

                if (entity == null)
                {
                    return false;
                }

                entity.ObjectState = ObjectState.Deleted;
                _dbSet.Attach(entity);

                return true;
            }

            internal IQueryable<TEntity> Select(
                Expression<Func<TEntity, bool>> filter = null,
                Func<IQueryable<TEntity>,
IOrderedQueryable<TEntity>> orderBy = null,
                List<Expression<Func<TEntity, object>>> includes =
null,
                int?
                int? pageSize = null)
            {
                IQueryable<TEntity> query=_dbSet;

                if (includes != null)
                {
                    query = includes.Aggregate(query, (current,
include) => current.Include(include));
                }
                if (orderBy! = null)
                {
                    query = orderBy(query);
                }
```

```
                if (filter! = null)
                {
                    query = query.AsExpandable().Where(filter);
                }
                if (page != null && pageSize != null)
                {
                    query = query.Skip((page.Value - 1) * 
pageSize.Value).Take(pageSize.Value);
                }
                return query;
            }

            internal async Task<IEnumerable<TEntity>> SelectAsync(
                Expression<Func<TEntity, bool>> filter = null,
                Func<IQueryable<TEntity>, 
IOrderedQueryable<TEntity>> orderBy = null,
                List<Expression<Func<TEntity, object>>> includes = 
null,
                int?
                int? pageSize = null)
            {
                return await Select(filter, orderBy, includes, page, 
pageSize).ToListAsync();
            }

            public virtual void InsertOrUpdateGraph(TEntity entity)
            {
                SyncObjectGraph(entity);
                _entitesChecked = null;
                _dbSet.Attach(entity);
            }

            HashSet<object> _entitesChecked; // tracking of all 
process entities в графіці об'єкта при запуску SyncObjectGraph

            private void SyncObjectGraph(object entity) // scan 
object graph for all
            {
                if (_entitesChecked == null)
```

```
                    _entitesChecked = новий HashSet<object>();

                if (_entitesChecked.Contains(entity))
                    return;

                _entitesChecked.Add(entity);

                var objectState = entity as IObjectState;

                if (objectState != null && objectState.ObjectState
== ObjectState.Added)
                    _context.SyncObjectState((IObjectState)entity);

                // Set tracking state for child collections
                foreach           (var           prop           in
entity.GetType().GetProperties())
                {
                    // Apply changes to 1-1 and M-1 properties
                    var trackableRef = prop.GetValue(entity, null)
as IObjectState;
                    if (trackableRef! = null)
                    {
                        if       (trackableRef.ObjectState       ==
ObjectState.Added)

_context.SyncObjectState((IObjectState)entity);

                        SyncObjectGraph(prop.GetValue(entity,
null));
                    }

                    // Apply changes to 1-M properties
                    var  items  =  prop.GetValue(entity,  null)  as
IEnumerable<IObjectState>;
                    if (items == null) continue;

                    Debug.WriteLine(«Checking    collection:»    +
prop.Name);
```

```
                    foreach (var item in items)
    SyncObjectGraph(item);
                }
            }
        }
    }

    using System;
    using System.Data.Entity;

    using Smartify.DAL.Infrastructure;

    namespace Smartify.DAL
    {
        public class StateHelper
        {
            public static EntityState ConvertState(ObjectState
state)
            {
                switch (state)
                {
                    case ObjectState.Added:
                        return EntityState.Added;

                    case ObjectState.Modified:
                        return EntityState.Modified;

                    case ObjectState.Deleted:
                        return EntityState.Deleted;

                    default:
                        return EntityState.Unchanged;
                }
            }

            public static ObjectState ConvertState(EntityState
state)
            {
                switch (state)
```

```
                {
                    case EntityState.Detached:
                        return ObjectState.Unchanged;

                    case EntityState.Unchanged:
                        return ObjectState.Unchanged;

                    case EntityState.Added:
                        return ObjectState.Added;

                    case EntityState.Deleted:
                        return ObjectState.Deleted;

                    case EntityState.Modified:
                        return ObjectState.Modified;

                    default:
                        throw                                   new
ArgumentOutOfRangeException(«state»);
      }
              }
          }
      }

      using System;
      використовуючи System.Collections.Generic;
      використовуючи System.Data;
      використовуючи System.Data.Common;
      using System.Data.Entity.Core.Objects;
      using System.Data.Entity.Infrastructure;
      використовуючи System.Threading;
      використовуючи System.Threading.Tasks;

      using Microsoft.Practices.ServiceLocation;

      using Smartify.DAL.DataContexts;
      using Smartify.DAL.Infrastructure;
      using Smartify.DAL.Repositories;
      using Smartify.DAL.UnitOfWork;
```

```
namespace Smartify.DAL
{
    public class UoF : IUnitOfWorkAsync
    {
        #region Private Fields

        private IDataContextAsync _dataContext;
        private bool _disposed;
        private ObjectContext _objectContext;
        private DbTransaction _transaction;
        private Dictionary<string, dynamic> _repositories;

        #endregion Private Fields

        #region Constuctor/Dispose

        public UoF(IDataContextAsync dataContext)
        {
            _dataContext = dataContext;
            _repositories = new Dictionary<string, dynamic>();
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        public virtual void Dispose(bool disposing)
        {
            if (_disposed)
                return;

            if (disposing)
            {
                // free other managed objects that implement
                // IDisposable only
```

```
                    try
                    {
                        if (_objectContext != null &&
_objectContext.Connection.State == ConnectionState.Open)
                        {
                            _objectContext.Connection.Close();
                        }
                    }
                    catch (ObjectDisposedException)
                    {
                        // do nothing, theobjectContext has already
been disposed
                    }

                    if (_dataContext! = null)
                    {
                        _dataContext.Dispose();
                        _dataContext = null;
                    }
                }

                // release any unmanaged objects
                // set the object references to null

                _disposed = true;
            }

            #endregion Constuctor/Dispose

            public int SaveChanges()
            {
                return _dataContext.SaveChanges();
            }

            public IRepository<TEntity> Repository<TEntity>() where
TEntity : class, IObjectState
            {
                if (ServiceLocator.IsLocationProviderSet)
                {
```

```
                    return
ServiceLocator.Current.GetInstance<IRepository<TEntity>>();
                }
                return RepositoryAsync<TEntity>();
            }
            public Task<int> SaveChangesAsync()
            {
                return _dataContext.SaveChangesAsync();
            }
            public   Task<int>   SaveChangesAsync(CancellationToken
cancellationToken)
            {
                return
_dataContext.SaveChangesAsync(cancellationToken);
            }

            public                          IRepositoryAsync<TEntity>
RepositoryAsync<TEntity>() where TEntity : class, IObjectState
            {
                if (ServiceLocator.IsLocationProviderSet)
                {
                    return
ServiceLocator.Current.GetInstance<IRepositoryAsync<TEntity>>();
                }

                if (_repositories == null)
                {
                    _repositories    =    new    Dictionary<string,
dynamic>();
                }

                var type = typeof(TEntity).

                if (_repositories.ContainsKey(type))
                {
                    return
(IRepositoryAsync<TEntity>)_repositories[type];
                }
                var repositoryType = typeof(Repository<>);
```

```
                _repositories.Add(type,
Activator.CreateInstance(repositoryType.MakeGenericType(typeof(TEnti
ty)), _dataContext, this)));

                return _repositories[type];
            }

            #region Unit of Work Transactions

            public       void       BeginTransaction(IsolationLevel
isolationLevel = IsolationLevel.Unspecified)
            {
                _objectContext                                    =
((IObjectContextAdapter)_dataContext).ObjectContext;
                if        (_objectContext.Connection.State!       =
ConnectionState.Open)
                {
                    _objectContext.Connection.Open();
                }
                _transaction                                      =
_objectContext.Connection.BeginTransaction(isolationLevel);
            }

            public bool Commit()
            {
                _transaction.Commit();
                return true;
            }

            public void Rollback()
            {
                _transaction.Rollback();
                _dataContext.SyncObjectsStatePostCommit();
            }

    #endregion
        }
    }
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	83 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Костюченко В.М.			Розробка рекомендувального сервісу для відстеження прогресу досягнення мети	Шифр групи	Код напр./спец.
Перевірив		Бродецький Ф.А.				ІТШІп-23-1	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Золотухін О.В.					