Сучасні інтернет технології. Лабораторне заняття №2
РОБОТА З ДАНИМИ В ASP.NET CORE.
РЕАЛІЗАЦІЯ ШАБЛОНУ REPOSITORY

Виконала: студентка групи МІТ-41, Панченко Владислава

Завдання для виконання:

1. Ознайомитися з теоретичними основами шаблону Repository (визначення, призначення, переваги та недоліки), принципу інверсії залежностей, механізму впровадження залежностей. Проаналізувати можливу проблему подвійного відстеження при оновленні даних.

   Базовий інтерфейс IRepository визначає узагальнені методи для CRUD-операцій та запитів (All, AddAsync, UpdateAsync тощо), забезпечуючи абстракцію. Конкретний інтерфейс IWebRepository успадковує IRepository і додає специфічний метод для домену: GetUserByEmailAsync, що відповідає потребам веб-застосунку.

```
namespace WebApplicationData.Interfaces
{
    2 references
    public interface IRepository
    {

        5 references
        IQueryable<T> All<T>() where T : class;


        4 references
        IQueryable<T> ReadAll<T>() where T : class;


        1 reference
        IQueryable<T> ReadWhere<T>(Expression<Func<T, bool>> expression) where T : class;


        2 references
        Task<T?> FirstOrDefaultAsync<T>(Expression<Func<T, bool>> expression) where T : class;

        1 reference
        Task<T?> SingleAsync<T>(Expression<Func<T, bool>> expression) where T : class;

        1 reference
        Task<T?> ReadSingleAsync<T>(Expression<Func<T, bool>> expression) where T : class;

        1 reference
        Task<int> AddAsync<T>(T item) where T : class;

        1 reference
        Task<int> UpdateAsync<T>(T item) where T : class;

        1 reference
        Task<int> RemoveAsync<T>(T item) where T : class;


        1 reference
        Task<bool> ExistsAsync<T>(Expression<Func<T, bool>> expression) where T : class;
    }
```

```
using WebApplicationData.Data;
using WebApplicationData.Interfaces;

namespace WebApplicationData.Repositories
{
    4 references
    public interface IWebRepository : IRepository
    {
        1 reference
        Task<WebApplicationUser?> GetUserByEmailAsync(string email);
    }
}
```

2. Створити базовий та конкретний інтерфейси репозиторію, що визначає базові методи для роботи з даними (отримання всіх записів, пошук за

умовою, додавання, оновлення, видалення). Додати базовий метод для перевірки існування сутності за умовою (наприклад, ExistsAsync()). Пояснити, у яких випадках він може бути корисним.

Метод ExistsAsync<T>(...) реалізовано через All<T>().AnyAsync(expression). Це є оптимізованим підходом, оскільки він генерує запит EXISTS у SQL, повертаючи лише true/false без завантаження об'єктів. Корисний для валідації унікальності.

```
using Microsoft.EntityFrameworkCore;
using System.Linq.Expressions;
using WebApplicationData.Interfaces;

namespace WebApplicationData.Repositories
{
    3 references
    public class BaseSqlServerRepository<TDbContext> : IRepository where TDbContext : DbContext
    {
        10 references
        protected TDbContext Db { get; set; }

        1 reference
        public BaseSqlServerRepository(TDbContext db)
        {
            Db = db;
        }

        1 reference
        public async Task<int> AddAsync<T>(T item) where T : class
        {
            await Db.AddAsync(item);
            return await Db.SaveChangesAsync();
        }

        5 references
        public IQueryable<T> All<T>() where T : class
        {
            return Db.Set<T>().AsQueryable();
        }

        1 reference
        public async Task<bool> ExistsAsync<T>(Expression<Func<T, bool>> expression) where T : class
        {
            return await All<T>().AnyAsync(expression);
        }

        2 references
        public async Task<T?> FirstOrDefaultAsync<T>(Expression<Func<T, bool>> expression) where T : class
        {
            return await All<T>().FirstOrDefaultAsync(expression);
        }
```

3. Реалізувати базовий клас репозиторію, який інкапсулює роботу з контекстом бази даних та забезпечує виконання CRUD-операцій.

Базовий клас BaseSqlServerRepository є узагальненим і містить поле protected TDBContext Db. Він інкапсулює контекст бази даних та реалізує всі методи IRepository. Методи читання, як-от ReadAll<T>(), використовують

.AsNoTracking() для оптимізації продуктивності, запобігаючи відстеженню об'єктів контекстом.

```
4 references
public IQueryable<T> ReadAll<T>() where T : class
{
    return All<T>().AsNoTracking();
}

1 reference
public async Task<T?> ReadSingleAsync<T>(Expression<Func<T, bool>> expression) where T : class
{
    return await ReadAll<T>().SingleOrDefaultAsync(expression);
}

1 reference
public IQueryable<T> ReadWhere<T>(Expression<Func<T, bool>> expression) where T : class
{
    return ReadAll<T>().Where(expression);
}

1 reference
public async Task<int> RemoveAsync<T>(T item) where T : class
{
    Db.Remove(item);
    return await Db.SaveChangesAsync();
}

1 reference
public async Task<T?> SingleAsync<T>(Expression<Func<T, bool>> expression) where T : class
{
    return await All<T>().SingleOrDefaultAsync(expression);
}

1 reference
public async Task<int> UpdateAsync<T>(T item) where T : class
{

    var local = Db.Set<T>()
        .Local
        .FirstOrDefault(entry => entry.Equals(item));

    if (local != null)
    {
      Db.Entry(local).State = EntityState.Detached;
    }

    Db.Entry(item).State = EntityState.Modified;
    return await Db.SaveChangesAsync();
}
```

4. Створити конкретний клас репозиторію для веб-застосунку та розширити його функціональність методом, що виконує пошук користувача за унікальною властивістю (наприклад, email).

Клас WebRepository успадковує BaseSqlServerRepository та реалізує специфічний для домену метод GetUserByEmailAsync(string email), використовуючи при цьому базовий метод FirstOrDefaultAsync.

```
using WebApplicationData.Data;
using WebApplicationData.Interfaces;

namespace WebApplicationData.Repositories
{
    2 references
    public class WebRepository : BaseSqlServerRepository<WebApplicationDbContext>, IWebRepository
    {
        0 references
        public WebRepository(WebApplicationDbContext db) : base(db)
        {
        }

        1 reference
        public async Task<WebApplicationUser?> GetUserByEmailAsync(string email)
        {
            return await FirstOrDefaultAsync<WebApplicationUser>(u => u.Email == email);
        }
    }
}
```

5. Зареєструвати залежності (інтерфейсу репозиторію та його реалізації) у контейнері впровадження залежностей.

Залежність зареєстрована як builder.Services.AddScoped<IWebRepository, WebRepository>(). Це забезпечує Впровадження Залежностей (DI). Вибір Scoped є стандартним для DbContext і репозиторіїв, гарантуючи один екземпляр на запит.

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using WebApplicationData.Data;
using WebApplicationData.Interfaces;
using WebApplicationData.Repositories;

var builder = WebApplication.CreateBuilder(args);


var connectionString = builder.Configuration.GetConnectionString("DefaultConnection")
    ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");


builder.Services.AddDbContext<WebApplicationDbContext>(options =>
    options.UseSqlServer(connectionString));


builder.Services.AddDatabaseDeveloperPageExceptionFilter();


builder.Services.AddDefaultIdentity<WebApplicationUser>(options =>
{
    options.SignIn.RequireConfirmedAccount = false;
})
.AddEntityFrameworkStores<WebApplicationDbContext>();


builder.Services.AddScoped<IWebRepository, WebRepository>();


builder.Services.AddControllersWithViews();
builder.Services.AddRazorPages();

var app = builder.Build();
```

6. Інтегрувати репозиторій у контролер: реалізувати метод, який отримує дані, використовуючи репозиторій.

Репозиторій впроваджується у конструктор HomeController через його абстракцію (IWebRepository). Метод дії IndexAsync() використовує впроваджений репозиторій (_repository.ReadAll<WebApplicationUser>()) для отримання даних користувачів.

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Diagnostics;
using WebApplication1.Models;
using WebApplicationData.Data;
using WebApplicationData.Interfaces;
using WebApplicationData.Repositories;


namespace WebApplication1.Controllers
{
    3 references
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;
        private readonly IWebRepository _repository;

        0 references
        public HomeController(ILogger<HomeController> logger, IWebRepository repository)
        {
            _logger = logger;
            _repository = repository;
        }

        0 references
        public async Task<IActionResult> IndexAsync()
        {
            var users = await _repository.ReadAll<WebApplicationUser>().ToListAsync();
            return View();
        }

        0 references
        public IActionResult Privacy()
        {
            return View();
        }


        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
        0 references
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
        }
    }
}
```