Київський національний університет ім. Тараса Шевченка
Кафедра мережевих та інтернет технологій

Лабораторна робота № 1

Дисципліна: Сучасні Інтернет Технології
Тема: Робота з даними в ASP.NET CORE. Реалізація шаблону REPOSITORY.

Підготувала студентка МІТ-41
Пась Олександра

Хід виконання

1. Створити базовий та конкретний інтерфейси репозиторію, що визначає базові методи для роботи з даними (отримання всіх записів, пошук за умовою, додавання, оновлення, видалення).

   В модулі, який відповідає за дані, створюємо папку Interfaces. У цій папці створюємо два інтерфейси конкретний ILabRepository (описуємо конкретні методи) та базовий IRepository (описуємо базові методи).

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

namespace labData.Interfaces
{
    5 references
    public interface ILabRepository: IRepository
    {
        3 references
        Task<User?> GetUserByEmailAsync(string email);
    }
}
```

рис. 2.1 - конкретний інтерфейс

```
using System.Linq.Expressions;
using System.Text;
using System.Threading.Tasks;

namespace labData.Interfaces
{
    1 reference
    public interface IRepository
    {
        5 references
        IQueryable<T?> All<T>() where T : class;
        3 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;

    }
}
```

рис. 2.2 - базовий інтерфейс

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

   Метод ExistsAsync() корисний для асинхронної перевірки наявності файлів (наприклад, статичних ресурсів, зображень, конфігураційних файлів) перед обробкою запитів, що дозволяє уникнути блокування потоків пулу та підвищити продуктивність застосунку.

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

   В тому ж модулі створюємо папку Repositories. В ній створюємо наш базовий клас репозиторію BaseSqlServerRepositor. Тут ми наслідуємо наш базовий інтерфейс та імлементуємо методи.

```
public class BaseSqlServerRepository<TDbContext> : ILabRepository
    where TDbContext : DbContext

{
    12 references
    protected TDbContext Db { get; set; }

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

    1 reference
    public Task<int> AddAsync<T>(T item) where T : class
    {
        Db.Entry(item).State = EntityState.Added;

        return Db.SaveChangesAsync();
    }
```

рис. 2.3 - базовий клас репозиторію

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

   У тій самій папці створюємо наш конкретний клас репозиторію LabSqlServerRepository. Тут ми наслідуємо конкретний інтерфейс ILabRepository та імплементуємо його методи.

```
public class LabSqlServerRepository : BaseSqlServerRepository<labDbContext>, ILabRepository
{
    0 references
    public LabSqlServerRepository(labDbContext db) : base(db)
    {
    }
    2 references
    public new async Task<User?> GetUserByEmailAsync(string email)
    {
        return await FirstOrDefaultAsync<User>(u => u.Email == email);
    }
}
```

рис. 2.4 - конкретний клас репозиторію

   Тут маємо метод пошуку користувача по email.

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

   В файлі programn.cs реєструємо сервіс репозиторію.

```
builder.Services.AddScoped<ILabRepository, LabSqlServerRepository>();
```

рис. 2.5 - реєструємо сервіс

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

   В файлі HomeController.cs додаємо залежність і метод нашого конкретного класу репозиторію.

```
public class HomeController : Controller
{
    private readonly ILogger<HomeController> _logger;
    private readonly ILabRepository _repository;
    0 references
    public HomeController(ILogger<HomeController> logger, ILabRepository repository)
    {
        _logger = logger;
        _repository = repository;
    }
```

рис. 2.6 - додаємо залежність

```
[HttpGet("UserId/{email}")]
0 references
public string UserId(string email)
{
    return _repository.GetUserByEmailAsync(email).Result?.Id ?? "User not found";
}
```

рис. 2.7 - наш метод

Висновок: Під час виконання цієї лабораторної роботи було створено базовий та конкретний інтерфейс репозиторію. Також базовий і конкретний клас репозиторію. Зареєстровано залежності та інтегровано репозиторій у контролер.