

# **МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ**

**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. ТАРАСА ШЕВЧЕНКА**

**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

**КАФЕДРА МЕРЕЖЕВИХ ТА ІНТЕРНЕТ ТЕХНОЛОГІЙ**

**Лабораторне заняття №2**

**Тема: РОБОТА З ДАНИМИ В ASP.NET CORE.  
РЕАЛІЗАЦІЯ ШАБЛОНУ REPOSITORY**

**Виконав студент групи: МІТ-41  
Кухарчук Богдан Петрович**

## Хід виконання роботи:

У процесі зростання складності системи пряме використання DbContext призводить до порушення принципу інверсії залежностей (DIP), внаслідок чого бізнес-логіка безпосередньо залежить від конкретної реалізації Entity Framework Core. Такий підхід ускладнює тестування, супровід і масштабування застосунку через дублювання запитів та тісний зв'язок між шарами системи.

Для усунення цих недоліків доцільно впровадити патерн Repository, який виступає проміжним шаром між доменною моделлю та інфраструктурою бази даних. Репозиторій інкапсулює логіку доступу до даних, забезпечує уніфікований інтерфейс і дозволяє використовувати різні реалізації через механізм залежностей. Таким чином, бізнес-логіка працює з абстракціями, а не з конкретними реалізаціями, що підвищує гнучкість та розширюваність системи.

### 1. Створення Інтерфейсів

Першим етапом реалізації є створення інтерфейсів, які виступають своєрідним контрактом між бізнес-логікою та шаром доступу до даних. Інтерфейс визначає, що саме має виконувати репозиторій, але не містить реалізації цих дій.

У межах проєкту Web Application Data наступним кроком є розробка базового інтерфейсу **IRepository.cs** Рис.1, який містить набір універсальних методів, спільних для роботи з будь-якими сутностями системи.

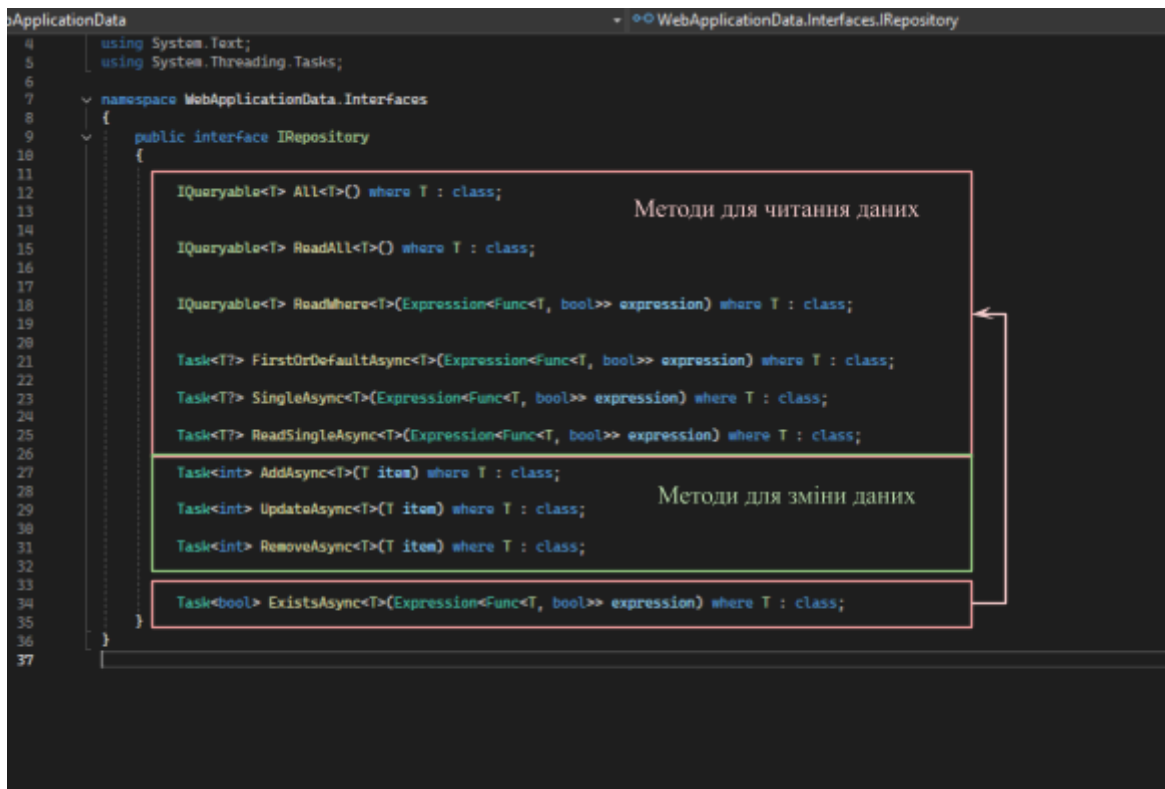


Рис.1 - Структура інтерфейсу IRepository.cs

Наступним етапом є створення конкретного інтерфейсу **IWebRepository.cs** Рис.2, який успадковує всі методи базового інтерфейсу **IRepository**. Даний інтерфейс може додатково містити специфічні методи, необхідні для реалізації вимог конкретного проєкту. У межах п'ятого завдання в інтерфейсі **IWebRepository.cs** було реалізовано специфічний метод пошуку користувачів за електронною адресою (e-mail), що розширює базову функціональність репозиторію відповідно до потреб проєкту.

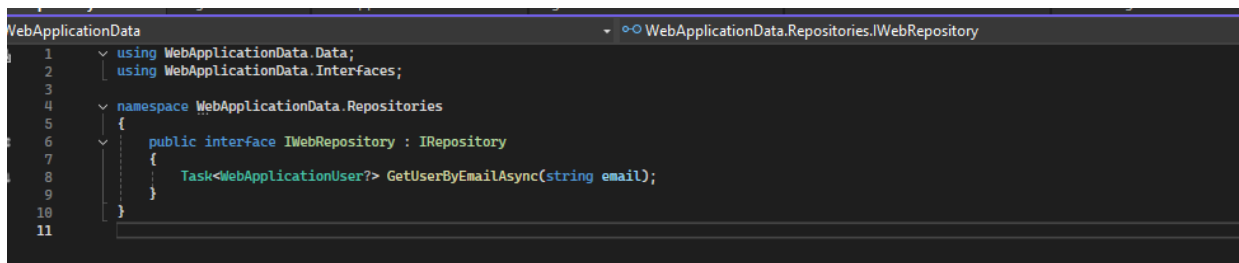
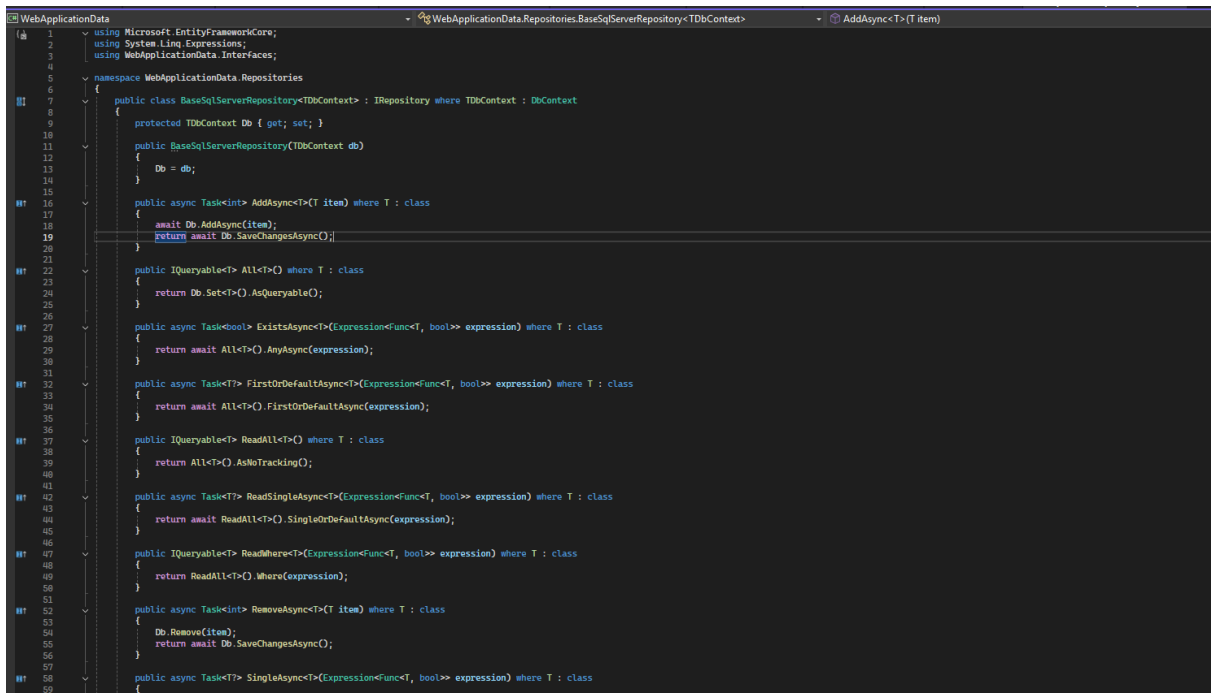


Рис.2 -Структура інтерфейсу IWebRepository.cs

## 2. Створення Реалізацій Репозиторію

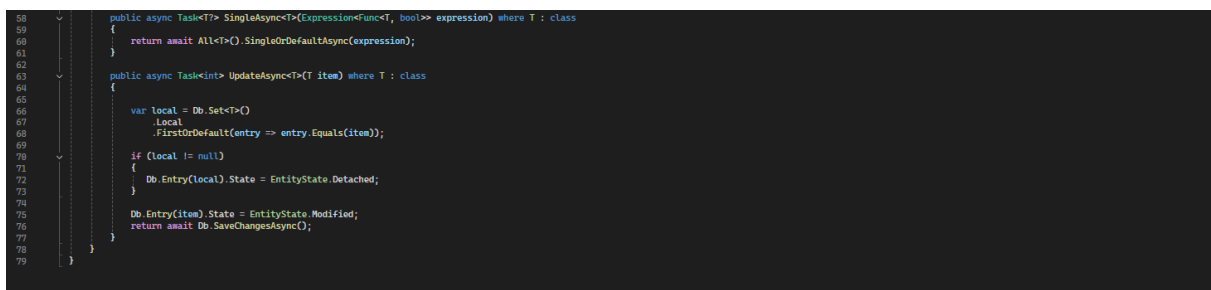
Наступним етапом є створення реалізації репозиторію, у межах якої безпосередньо описується логіка, визначена в інтерфейсах.

На цьому етапі було створено базовий клас **BaseSqlServerRepository.cs** — універсальну реалізацію інтерфейсу **IRepository**, яка забезпечує взаємодію з базою даних через об'єкт **DbContext** Рис.3 - Рис.4.



```
1 using Microsoft.EntityFrameworkCore;
2 using System.Linq.Expressions;
3 using WebApplicationData.Interfaces;
4
5 namespace WebApplicationData.Repositories
6 {
7     public class BaseSqlServerRepository<DbContext> : IRepository where DbContext : DbContext
8     {
9         protected DbContext Db { get; set; }
10
11         public BaseSqlServerRepository(DbContext db)
12         {
13             Db = db;
14         }
15
16         public async Task<int> AddAsync<T>(T item) where T : class
17         {
18             await Db.AddAsync(item);
19             return await Db.SaveChangesAsync();
20         }
21
22         public IQueryable<T> All<T>() where T : class
23         {
24             return Db.Set<T>().AsQueryable();
25         }
26
27         public async Task<bool> ExistsAsync<T>(Expression<Func<T, bool>> expression) where T : class
28         {
29             return await All<T>().AnyAsync(expression);
30         }
31
32         public async Task<T?> FirstOrDefaultAsync<T>(Expression<Func<T, bool>> expression) where T : class
33         {
34             return await All<T>().FirstOrDefaultAsync(expression);
35         }
36
37         public IQueryable<T> ReadAll<T>() where T : class
38         {
39             return All<T>().AsNoTracking();
40         }
41
42         public async Task<T?> ReadSingleAsync<T>(Expression<Func<T, bool>> expression) where T : class
43         {
44             return await ReadAll<T>().SingleOrDefaultAsync(expression);
45         }
46
47         public IQueryable<T> ReadWhere<T>(Expression<Func<T, bool>> expression) where T : class
48         {
49             return ReadAll<T>().Where(expression);
50         }
51
52         public async Task<int> RemoveAsync<T>(T item) where T : class
53         {
54             Db.Remove(item);
55             return await Db.SaveChangesAsync();
56         }
57
58         public async Task<T?> SingleAsync<T>(Expression<Func<T, bool>> expression) where T : class
59         {
60             return await All<T>().SingleOrDefaultAsync(expression);
61         }
62
63         public async Task<int> UpdateAsync<T>(T item) where T : class
64         {
65             var local = Db.Set<T>().
66                 .Local
67                 .FirstOrDefault(entry => entry.Equals(item));
68             if (local != null)
69             {
70                 Db.Entry(local).State = EntityState.Detached;
71             }
72             Db.Entry(item).State = EntityState.Modified;
73             return await Db.SaveChangesAsync();
74         }
75     }
76 }
```

Рис.3 - Загальна структура **BaseSqlServerRepository.cs**



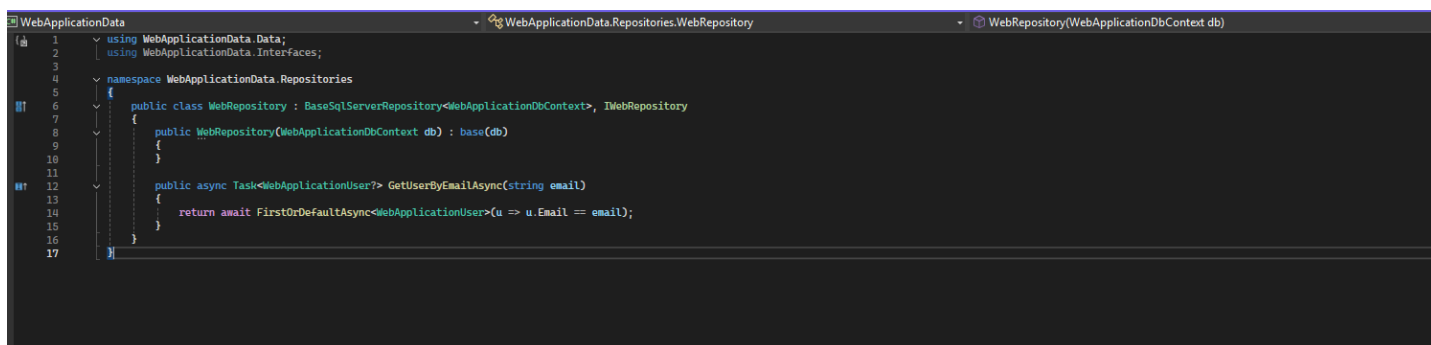
```
58 public async Task<T?> SingleAsync<T>(Expression<Func<T, bool>> expression) where T : class
59 {
60     return await All<T>().SingleOrDefaultAsync(expression);
61 }
62
63 public async Task<int> UpdateAsync<T>(T item) where T : class
64 {
65     var local = Db.Set<T>().
66         .Local
67         .FirstOrDefault(entry => entry.Equals(item));
68     if (local != null)
69     {
70         Db.Entry(local).State = EntityState.Detached;
71     }
72     Db.Entry(item).State = EntityState.Modified;
73     return await Db.SaveChangesAsync();
74 }
75 }
```

Рис.4 - Реалізація **UpdateAsync**, де перевіряється чи є об'єкт в локальному кеші та від'єднують старий, щоб уникнути конфлікту трекінгу

Клас **WebRepository** Рис.5 використовується у додатку для роботи з даними користувачів. Він успадковує базовий клас **BaseSqlServerRepository**, що забезпечує універсальні методи взаємодії з базою даних через DbContext, та реалізує конкретний інтерфейс **IWebRepository**.

Конструктор класу приймає об'єкт **WebApplicationDbContext**, який передається у базовий клас для подальшої роботи з БД.

Клас також містить метод **GetUserByEmailAsync**, який реалізує специфічну для проекту функціональність пошуку користувача за електронною поштою. Метод асинхронно звертається до бази даних та повертає об'єкт користувача null, якщо такий не знайдено.

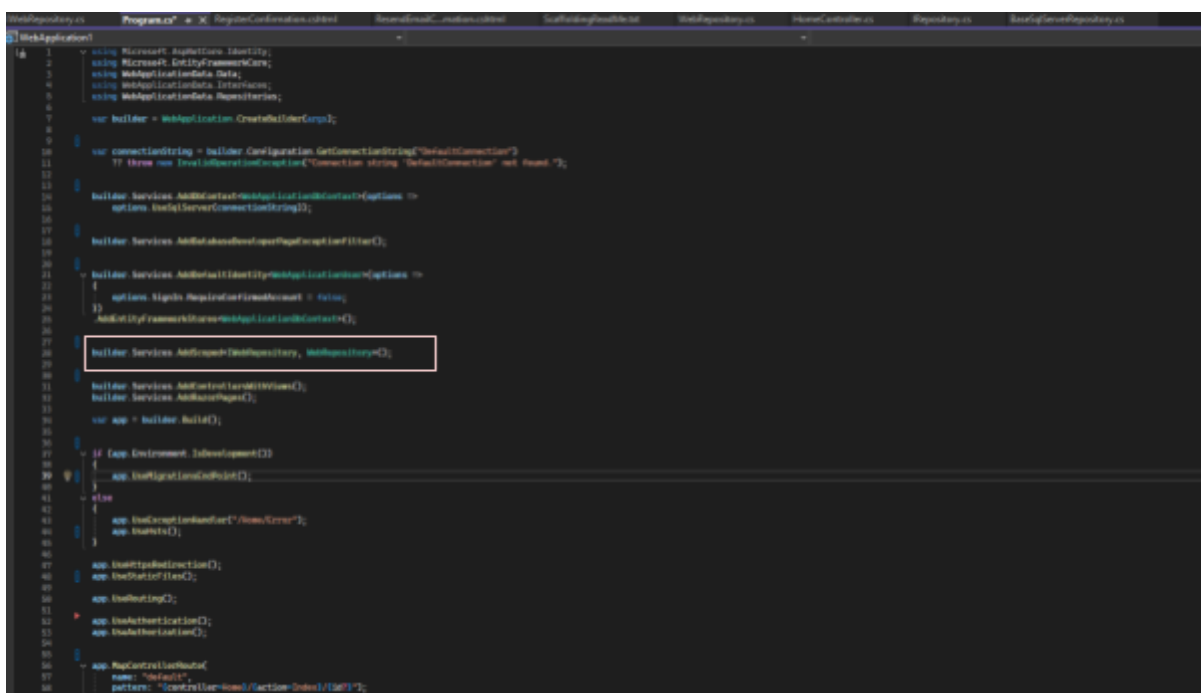


```
1 using WebApplicationData.Data;
2 using WebApplicationData.Interfaces;
3
4 namespace WebApplicationData.Repositories
5 {
6     public class WebRepository : BaseSqlServerRepository<WebApplicationDbContext>, IWebRepository
7     {
8         public WebRepository(WebApplicationDbContext db) : base(db)
9         {
10         }
11
12         public async Task<WebApplicationUser?> GetUserByEmailAsync(string email)
13         {
14             return await FirstOrDefaultAsync<WebApplicationUser>(u => u.Email == email);
15         }
16     }
17 }
```

Рис. 5 - Структура класу **WebRepository**

### 3. Реєстрація Репозиторію

Наступним етапом, один із ключових, є реєстрація репозиторіїв. На цьому кроці потрібно повідомити основному проєкту, що при зверненні до інтерфейсу **IWebRepository** слід використовувати клас **WebRepository**. Для цього у файлі **Program.cs** додається відповідний рядок коду, який налаштовує впровадження залежностей Рис.6.



```
1 using Microsoft.AspNetCore.Identity;
2 using Microsoft.EntityFrameworkCore;
3 using WebApplicationData.Data;
4 using WebApplicationData.Interfaces;
5 using WebApplicationData.Repositories;
6
7 var builder = WebApplication.CreateBuilder(args);
8
9
10 var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
11 ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");
12
13
14 builder.Services.AddDbContext<WebApplicationDbContext>(options =>
15     options.UseSqlServer(connectionString));
16
17 builder.Services.AddAntiforgery(options =>
18     options.EnableRequestFormProtection = true);
19
20 builder.Services.AddAuthentication<WebApplicationDbContext>(options =>
21     {
22         options.SignIn.RequireConfirmedAccount = true;
23     })
24     .AddJwtBearer(options =>
25         {
26             options.TokenValidationParameters.ValidateIssuer = false;
27         });
28
29 builder.Services.AddScoped<IWebRepository, WebRepository>();
30
31 builder.Services.AddControllersWithViews();
32 builder.Services.AddRazorPages();
33
34 var app = builder.Build();
35
36
37 if (app.Environment.IsDevelopment())
38 {
39     app.UseDeveloperExceptionPage();
40 }
41 else
42 {
43     app.UseExceptionHandler("/Home/Error");
44     app.UseHsts();
45 }
46
47 app.UseHttpsRedirection();
48 app.UseStaticFiles();
49 app.UseRouting();
50
51 app.UseAuthorization();
52 app.UseEndpoints(endpoints =>
53 {
54     endpoints.MapControllerRoute(
55         name: "default",
56         pattern: "{controller}/{action}/{id?}");
57 });
```

Рис. 6 - Коли хтось просить **IWebRepository**, DI-контейнер надасть йому **WebRepository**

#### 4. Використання Репозиторію в Контролері **Рис.7**

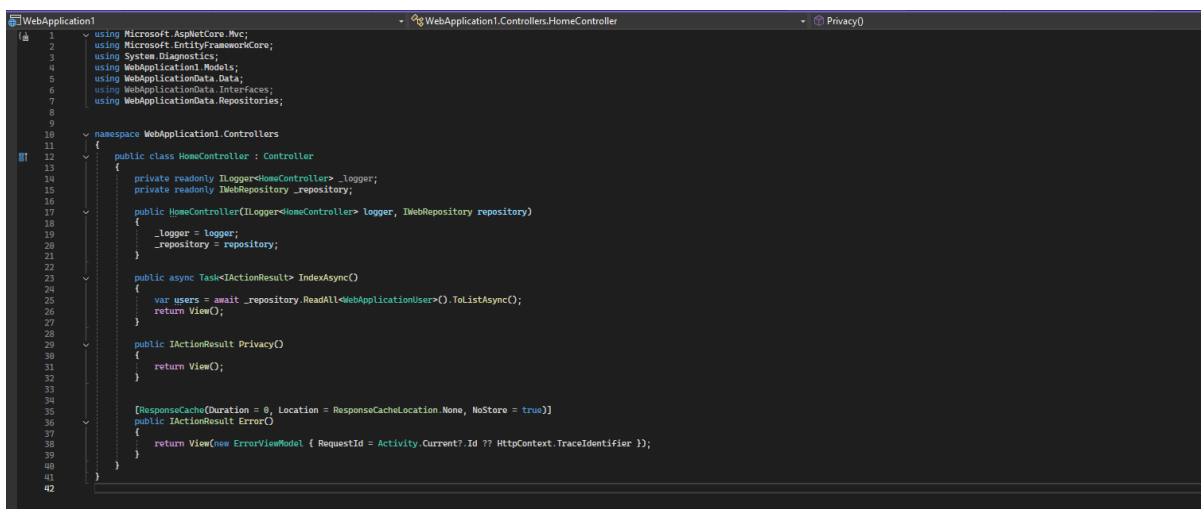


Рис.7 - Структура контролера **HomeController.cs**

Заключним, але не менш важливим етапом є використання репозиторію в контролері, що дозволяє отримувати дані з бази для відображення у веб-додатку.

У класі **HomeController** реалізовано впровадження залежностей через Dependency Injection: у конструктор передаються логгер **ILogger<HomeController>** та інтерфейс **IWebRepository**, що дозволяє контролеру взаємодіяти з даними через абстракцію, а не напряму через **DbContext**.

Метод **IndexAsync** асинхронно отримує список всіх користувачів з бази даних за допомогою методу **ReadAll<WebApplicationUser>()** репозиторію та перетворює результат у список для подальшого використання у вигляді.

Інші методи контролера (**Privacy**, **Error**) забезпечують стандартну роботу сторінок, включно з обробкою помилок.

#### 5. Завантаження результатів та звіту на GitHub репозиторій

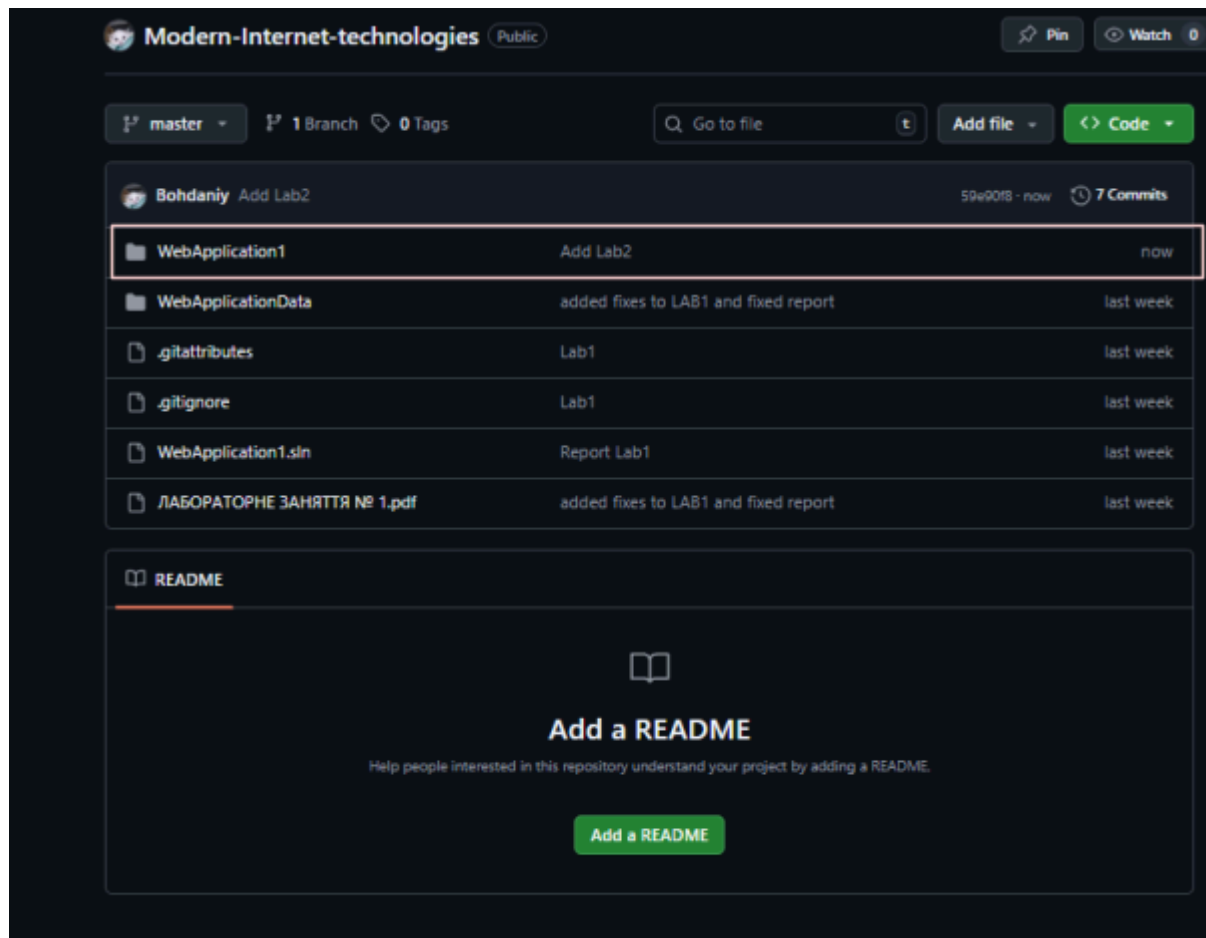


Рис. 9 - Завантаження Лабораторної на репозиторій

## Висновок

У ході виконання роботи було реалізовано шар доступу до даних із застосуванням патерну Repository в проєкті **WebApplicationData**. Було створено базові та специфічні інтерфейси (**IRepository**, **IWebRepository**) та їх реалізації (**BaseSQLServerRepository**, **WebRepository**), налаштовано Dependency Injection для підключення репозиторію до контролерів, а також продемонстровано використання репозиторію в контролері HomeController для асинхронного отримання даних користувачів.

Запровадження репозиторію забезпечило ізоляцію бізнес-логіки від конкретної реалізації Entity Framework Core, підвищило гнучкість, тестованість та масштабованість системи.

Результати роботи та весь вихідний код проєкту було завантажено у GitHub репозиторій для подальшого використання та контролю версій.