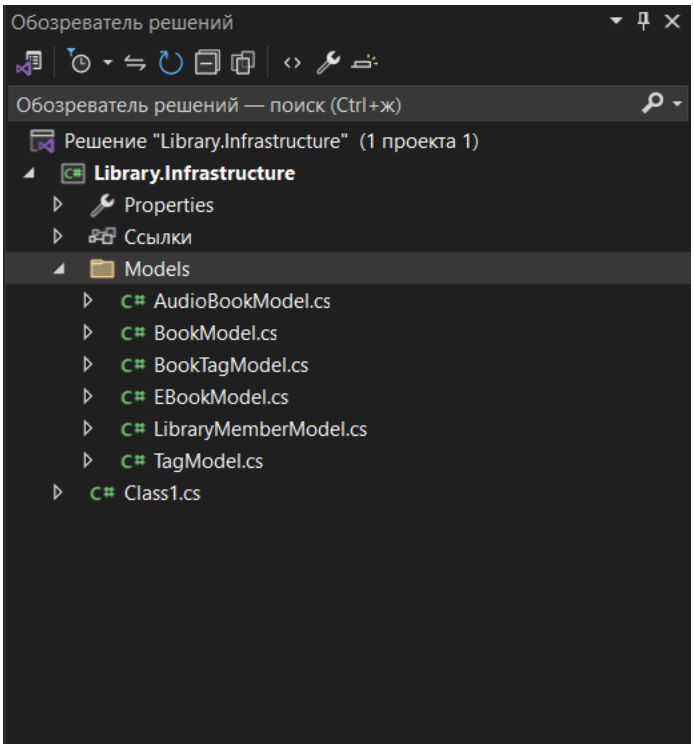
Лабораторна робота №3

Робота із базами даних в .NET. Використання бібліотеки Entity Framework.

Виконав студент групи 302-ТК
Писаренко Дмитрій

1. Створити проєкт {Назва тематики}.Infrastructure. У новоствореному проєкті створити папку Models. У створеній папці додати модель(набір класів), яка відповідає створеній у першій лабораторній роботі моделі, та містить всередині себе властивості. Наприклад, якщо існував у першій лабораторній клас Bus, у папці Models необхідно створити клас BusModel. Отримана модель має обов'язково мати зв'язки один-до-одного та один-до-багатьох. За додавання зв'язків багато-до-багатьох можна отримати додаткові бали. Для наслідування більш пріоритетно застосовувати підхід Таблиця на тип (Table-per-Type).

Структура проєкту:



Клас BookModel:

```
Library.Infrastructure                                  Library.Infrastructure.Models.Boo
 1   using System;
 2   using System.Collections.Generic;
 3   using System.Linq;
 4   using System.Text;
 5   using System.Threading.Tasks;
 6
 7   namespace Library.Infrastructure.Models
 8   {
         Ссылок: 4
 9       public class BookModel
10       {
             Ссылок: 0
11           public int Id { get; set; }
             Ссылок: 0
12           public string Title { get; set; }
             Ссылок: 0
13           public string Author { get; set; }
14
15           // Один-до-багатьох: Книга належить одному учаснику
             Ссылок: 0
16           public int LibraryMemberId { get; set; }
             Ссылок: 0
17           public LibraryMemberModel LibraryMember { get; set; }
18
19           // Багато-до-багатьох: Книга має багато тегів
             Ссылок: 0
20           public List<BookTagModel> BookTags { get; set; }
21       }
22   }
```

Клас EBookModel:

```
Library.Infrastructure                                  Library.Infrastruct
 1   using System;
 2   using System.Collections.Generic;
 3   using System.Linq;
 4   using System.Text;
 5   using System.Threading.Tasks;
 6
 7   namespace Library.Infrastructure.Models
 8   {
         Ссылок: 0
 9       public class EBookModel : BookModel
10       {
             Ссылок: 0
11           public string FileFormat { get; set; }
             Ссылок: 0
12           public double FileSizeMB { get; set; }
13       }
14   }
```

Клас AudioBookModel:

```
Library.Infrastructure                                  Library.Infrastructure.Models.AudioBo
 1   using System;
 2   using System.Collections.Generic;
 3   using System.Linq;
 4   using System.Text;
 5   using System.Threading.Tasks;
 6
 7   namespace Library.Infrastructure.Models
 8   {
         Ссылок: 0
 9       public class AudioBookModel : BookModel
10       {
             Ссылок: 0
11           public string AudioFormat { get; set; }
             Ссылок: 0
12           public double DurationInMinutes { get; set; }
13       }
14   }
```

Клас LibraryMemberModel:

```
Library.Infrastructure                          Library.Infrastructure.Models.LibraryMemb
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace Library.Infrastructure.Models
8   {
        Ссылок: 1
9       public class LibraryMemberModel
10      {
            Ссылок: 0
11          public int Id { get; set; }
            Ссылок: 0
12          public string FullName { get; set; }
            Ссылок: 0
13          public DateTime JoinDate { get; set; }
14
15          // Один-до-багатьох: Один учасник може позичати багато книжок
            Ссылок: 0
16          public List<BookModel> BorrowedBooks { get; set; }
17      }
18  }
```

Клас TagModel:

```
Library.Infrastructure                          Library.Infrastructure.Models.TagMod
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace Library.Infrastructure.Models
8   {
        Ссылок: 1
9       public class TagModel
10      {
            Ссылок: 0
11          public int Id { get; set; }
            Ссылок: 0
12          public string Name { get; set; }
13
14          // Багато-до-багатьох: Один тег – багато книжок
            Ссылок: 0
15          public List<BookTagModel> BookTags { get; set; }
16      }
17  }
```

Клас BookTegModel: (таблиця зв'язку багато-до-багатьох)

```
Library.Infrastructure                          Library.Infrastruc
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace Library.Infrastructure.Models
8   {
        Ссылок: 2
9       public class BookTagModel
10      {
            Ссылок: 0
11          public int BookId { get; set; }
            Ссылок: 0
12          public BookModel Book { get; set; }
13
            Ссылок: 0
14          public int TagId { get; set; }
            Ссылок: 0
15          public TagModel Tag { get; set; }
16      }
17  }
```

2. Створити клас {Назва тематики}Context, який наслідуватиметься від класу DbContext та опише схему бази даних для обраної тематики застосовуючи Entity Framework та анотації або Fluent API на вибір, Fluent API більш пріоритетно.

LibraryContext.cs

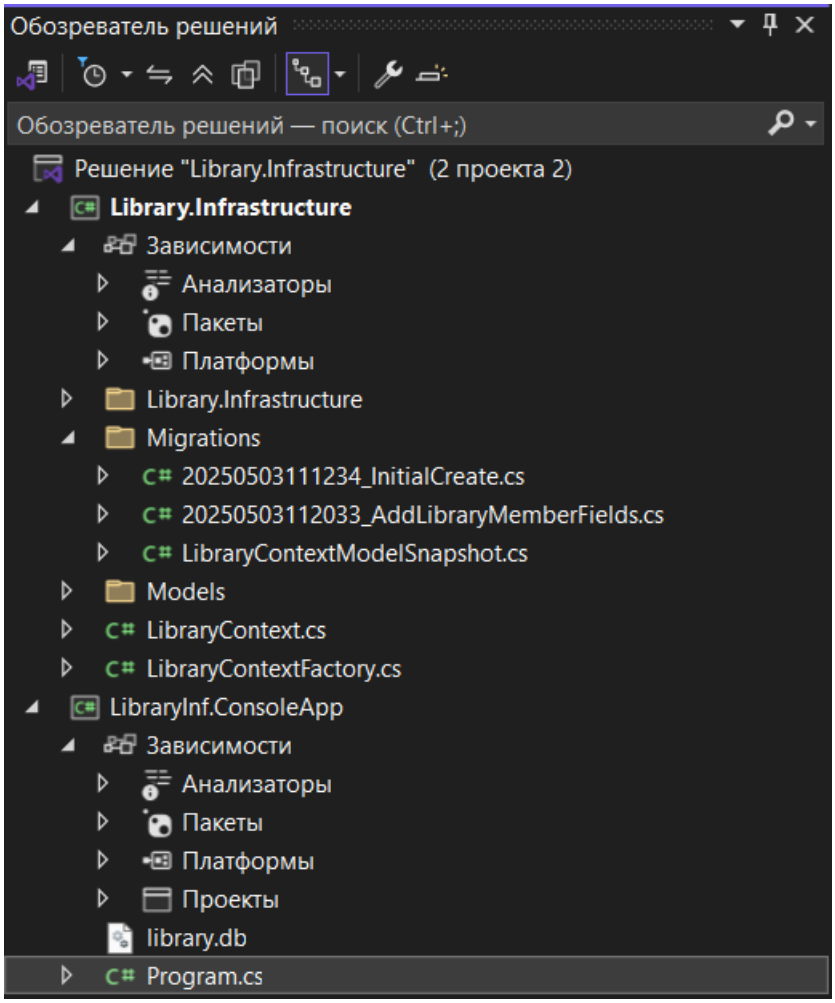
Library.Infrastructure — Library.Infrastructure.LibraryContext

```
using Library.Infrastructure.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;


namespace Library.Infrastructure
{
    Ссылок: 2
    public class LibraryContext : DbContext
    {
        Ссылок: 0
        public LibraryContext(DbContextOptions<LibraryContext> options)
            : base(options)
        {
        }

        Ссылок: 0
        public DbSet<BookModel> Books { get; set; }
        Ссылок: 0
        public DbSet<AudioBookModel> AudioBooks { get; set; }
        Ссылок: 0
        public DbSet<EBookModel> EBooks { get; set; }
        Ссылок: 0
        public DbSet<LibraryMemberModel> LibraryMembers { get; set; }
        Ссылок: 0
        public DbSet<TagModel> Tags { get; set; }
        Ссылок: 0
        public DbSet<BookTagModel> BookTags { get; set; }

        Ссылок: 0
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            // ===== Table-per-Type успадкування =====
            modelBuilder.Entity<BookModel>().ToTable("Books");
            modelBuilder.Entity<AudioBookModel>().ToTable("AudioBooks");
            modelBuilder.Entity<EBookModel>().ToTable("EBooks");

            // ===== Один-до-багатьох: Book → LibraryMember =====
            modelBuilder.Entity<BookModel>()
                .HasOne(b => b.LibraryMember)
                .WithMany(l => l.BorrowedBooks)
                .HasForeignKey(b => b.LibraryMemberId)
                .OnDelete(DeleteBehavior.Cascade);
```

```
            // ===== Багато-до-багатьох: Book ↔ Tag через BookTag =====
            modelBuilder.Entity<BookTagModel>()
                .HasKey(bt => new { bt.BookId, bt.TagId });

            modelBuilder.Entity<BookTagModel>()
                .HasOne(bt => bt.Book)
                .WithMany(b => b.BookTags)
                .HasForeignKey(bt => bt.BookId);

            modelBuilder.Entity<BookTagModel>()
                .HasOne(bt => bt.Tag)
                .WithMany(t => t.BookTags)
                .HasForeignKey(bt => bt.TagId);

            base.OnModelCreating(modelBuilder);
        }
    }
```

3. Створити міграцію використовуючи Entity Framework. Створену міграцію застосувати до реляційної бази даних. Можна використовувати будь-яку реляційну СУБД, як MS SQL, PostgreSQL, MySQL, тощо. Також слід використати SQLite, коли немає можливості розробляти та здавати Лабораторну роботу на одній системі. При використанні SQLite файл бази даних необхідно додати до гіт репозиторія.

Структура:



Program.cs:

LibraryMemberModel.cs | BookModel.cs | Program.cs | Libran

LibraryInf.ConsoleApp

```
using Library.Infrastructure;
using Library.Infrastructure.Models;
using Microsoft.EntityFrameworkCore;

var options = new DbContextOptionsBuilder<LibraryContext>()
    .UseSqlite("Data Source=library.db")
    .Options;

using var context = new LibraryContext(options);

// Застосовуємо міграції
context.Database.Migrate();

// Додаємо тестового учасника, якщо відсутній
var member = context.LibraryMembers.FirstOrDefault();
if (member == null)
{
    member = new LibraryMemberModel
    {
        Name = "Тестовий користувач",
        BorrowedBooks = new List<BookModel>()
    };
    context.LibraryMembers.Add(member);
    context.SaveChanges();
    Console.WriteLine("Додано тестового учасника.");
}

// Додаємо тестову книгу, якщо відсутня
if (!context.Books.Any(b => b.Title == "Test Book"))
{
    var testBook = new BookModel
    {
        Title = "Test Book",
        Author = "Test Author",
        LibraryMemberId = member.Id,
        BookTags = new List<BookTagModel>()
    };

    context.Books.Add(testBook);
    context.SaveChanges();
    Console.WriteLine("Додано тестову книгу.");
}
else
{
    Console.WriteLine("Тестова книга вже існує.");
}

Console.WriteLine("Готово.");
```

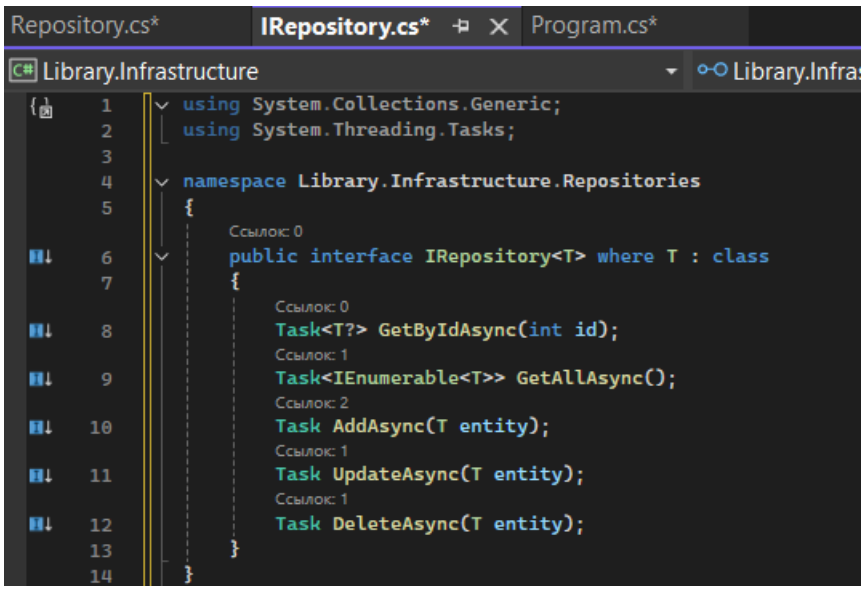
Результат:

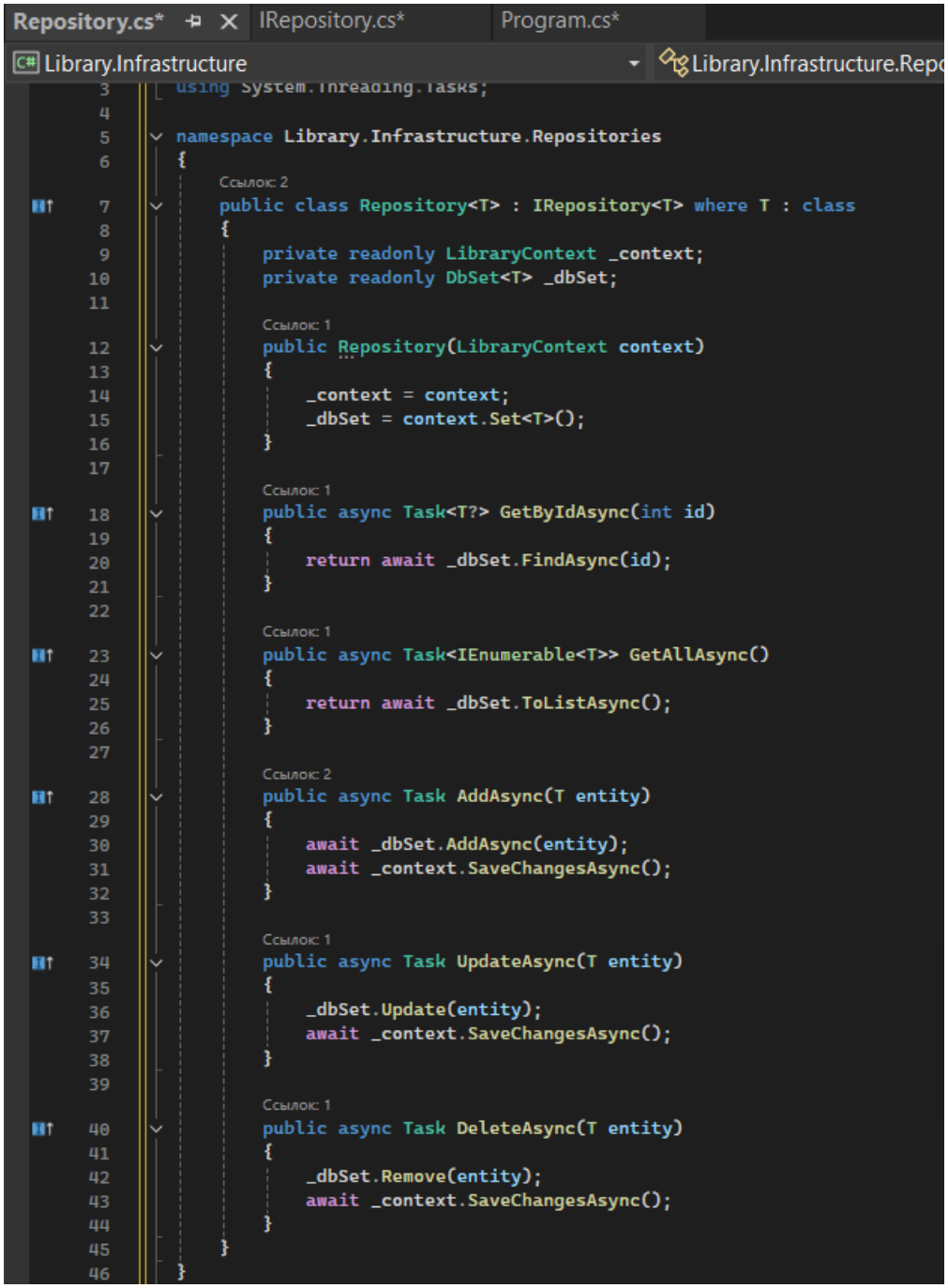
```
>>            dotnet run --project ./LibraryInf.ConsoleApp
>>
Додано тестового учасника.
Додано тестову книгу.
Готово.
```

4. Реалізувати шаблон проєктування Репозиторій, {Назва тематики}Context який буде використовувати клас {Назва тематики}Context для досьупу до даних із БД:

```
public interface IRepository<T> where T : class
{
    Task<T> GetByIdAsync(int id);
    Task<IEnumerable<T>> GetAllAsync();
```

```
    Task AddAsync(T entity);
    Task Update(T entity);
    Task Delete(T entity);
}
```





5. Оновити асинхрону версію дженерік CRUD сервісу, щоб він використовував репозиторій для доступу до даних:

```
public interface ICrudServiceAsync<T>
{
        public Task<bool> CreateAsync(T element);
        public Task<T> ReadAsync(Guid id);
public Task<IEnumerable<T>> ReadAllAsync();
        public Task<IEnumerable<T>> ReadAllAsync(int page, int amount);
        public Task<bool> UpdateAsync(T element);
        public Task<bool> RemoveAsync(T element);
        public Task<bool> SaveAsync();
```

}

CrudServiceAsync.cs* | **ICrudServiceAsync.cs*** | Program.cs*

Library.Infrastructure | Library.Infrastructure.S

```
1  using System;
2  using System.Collections.Generic;
3  using System.Threading.Tasks;
4
5  namespace Library.Infrastructure.Services
6  {
       Ссылок: 0
7      public interface ICrudServiceAsync<T>
8      {
           Ссылок: 0
9          Task<bool> CreateAsync(T element);
           Ссылок: 0
10         Task<T?> ReadAsync(Guid id);
           Ссылок: 0
11         Task<IEnumerable<T>> ReadAllAsync();
           Ссылок: 0
12         Task<IEnumerable<T>> ReadAllAsync(int page, int amount);
           Ссылок: 0
13         Task<bool> UpdateAsync(T element);
           Ссылок: 1
14         Task<bool> RemoveAsync(T element);
           Ссылок: 1
15         Task<bool> SaveAsync();
16     }
17 }
```

CrudServiceAsync.cs* | ICrudServiceAsync.cs* | Program.cs*

Library.Infrastructure | Library.Infrastructure.Services.CrudServiceAsync

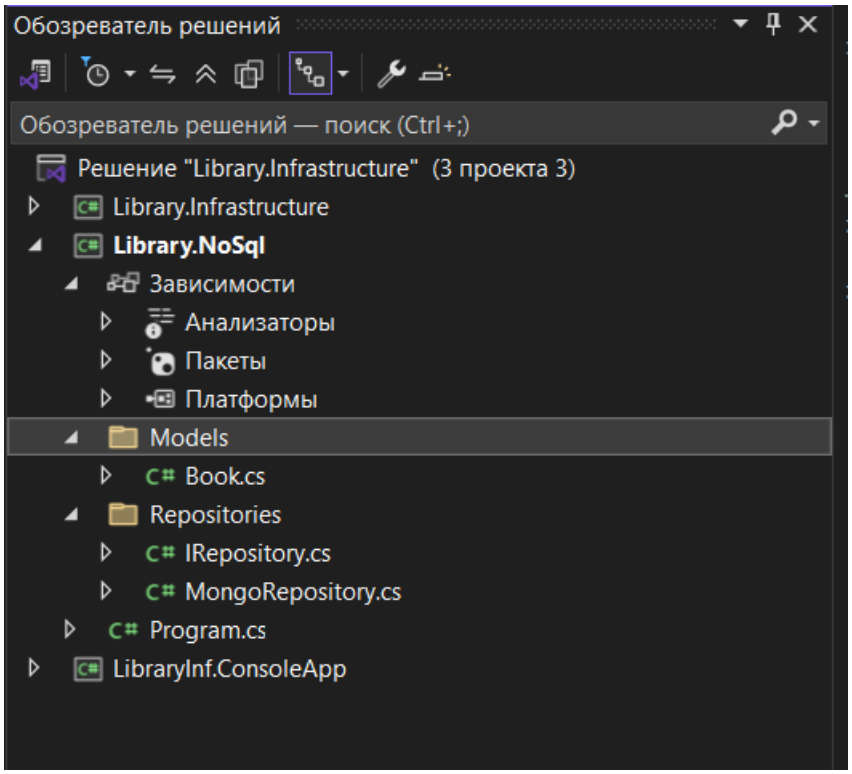
```
1  using Library.Infrastructure.Repositories;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Threading.Tasks;
6  using Microsoft.EntityFrameworkCore;
7
8  namespace Library.Infrastructure.Services
9  {
       Ссылок: 1
10     public class CrudServiceAsync<T> : ICrudServiceAsync<T> where T : class
11     {
12         private readonly IRepository<T> _repository;
13
           Ссылок: 0
14         public CrudServiceAsync(IRepository<T> repository)
15         {
16             _repository = repository;
17         }
18
           Ссылок: 1
19         public async Task<bool> CreateAsync(T element)
20         {
21             await _repository.AddAsync(element);
22             return true;
23         }
24
           Ссылок: 1
25         public async Task<T?> ReadAsync(Guid id)
26         {
27             var entity = await _repository.GetAllAsync();
28             return entity.FirstOrDefault(e => (e as dynamic).Id == id);
29         }
30
           Ссылок: 1
31         public async Task<IEnumerable<T>> ReadAllAsync()
32         {
33             return await _repository.GetAllAsync();
34         }
35
           Ссылок: 1
36         public async Task<IEnumerable<T>> ReadAllAsync(int page, int amount)
37         {
38             var all = await _repository.GetAllAsync();
39             return all.Skip((page - 1) * amount).Take(amount);
40         }
41
           Ссылок: 1
42         public async Task<bool> UpdateAsync(T element)
43         {
44             await _repository.UpdateAsync(element);
45             return true;
46         }
47
           Ссылок: 1
48         public async Task<bool> RemoveAsync(T element)
49         {
50             await _repository.DeleteAsync(element);
51             return true;
52         }
53
           Ссылок: 1
54         public Task<bool> SaveAsync()
55         {
56             return Task.FromResult(true);
57         }
58     }
```

6. Модифікуйте консольний застосунок, щоб він використовував оновлену версію CRUD сервісу та дані із бази даних.

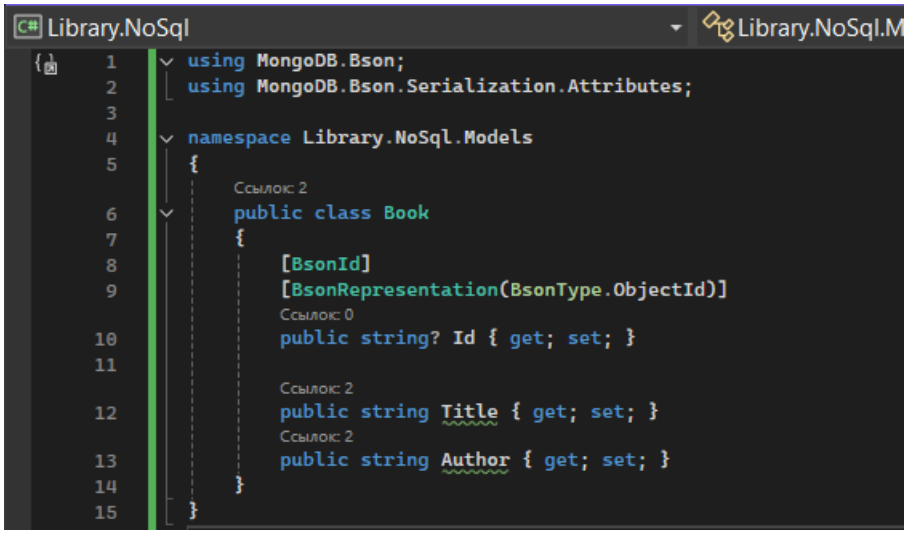
Додаткове завдання

Створіть проєкт {Назва тематики}.Nosql. У цьому проєкті реалізуйте репозиторій із 4 завдання, який буде використовувати нереляційну базу даних, як наприклад MongoDB. У звіт додайте скріншоти об’єктів із нереляційної бази даних. Використовувати Entity Framework для нереляційної БД необов’язково.

Структура:



Клас Book:



Клас IRepository:

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

namespace Library.NoSql.Repositories
{
    Ссылок: 0
    public interface IRepository<T> where T : class
    {
        Ссылок: 0
        Task<T?> GetByIdAsync(string id);
        Ссылок: 0
        Task<IEnumerable<T>> GetAllAsync();
        Ссылок: 0
        Task AddAsync(T entity);
        Ссылок: 0
        Task UpdateAsync(T entity);
        Ссылок: 0
        Task DeleteAsync(string id);
    }
}
```

Клас MongoRepository:

```
using MongoDB.Driver;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Library.NoSql
{
    Ссылок: 2
    public class MongoRepository<T> where T : class
    {
        private readonly IMongoCollection<T> _collection;

        Ссылок: 1
        public MongoRepository(string connectionString, string dbName, string collectionName)
        {
            var client = new MongoClient(connectionString);
            var database = client.GetDatabase(dbName);
            _collection = database.GetCollection<T>(collectionName);
        }

        Ссылок: 1
        public async Task AddAsync(T item) => await _collection.InsertOneAsync(item);

        Ссылок: 1
        public async Task<List<T>> GetAllAsync() =>
            await _collection.Find(_ => true).ToListAsync();
    }
}
```

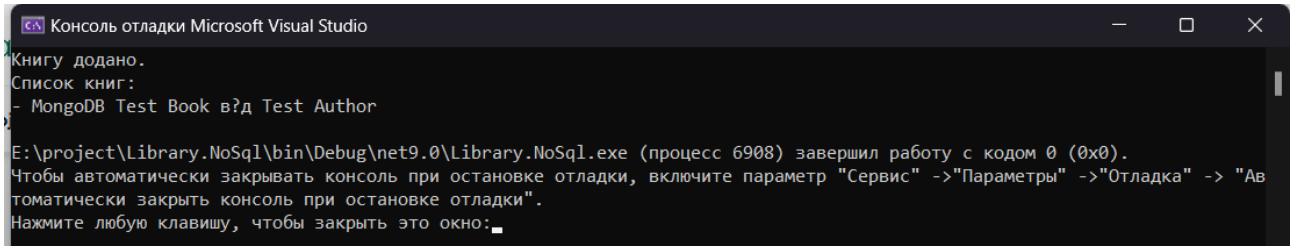
Клас Program:

Program.cs | Book.cs | MongoRepository.cs | IRepository.cs | CrudServiceAsync.cs | ICrudServiceAsync.cs

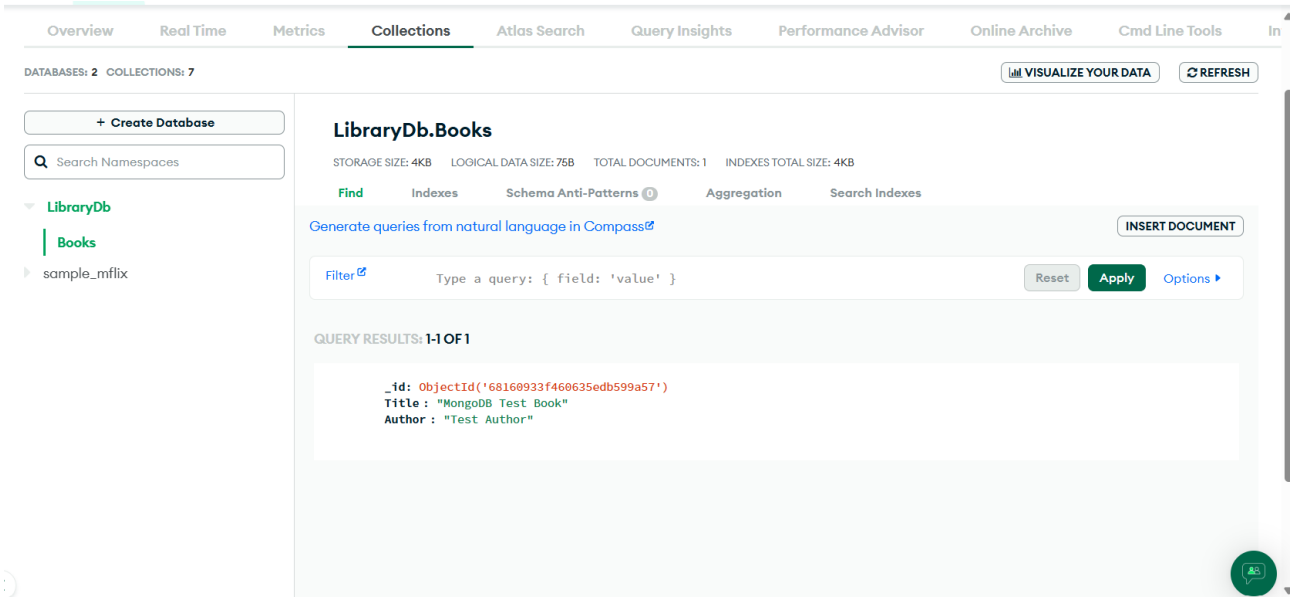
Library.NoSql | Program | Main()

```
using Library.NoSql;
using Library.NoSql.Models;
using Library.NoSql.Repositories;
using System;
using System.Threading.Tasks;

Ссылок: 0
class Program
{
    Ссылок: 0
    static async Task Main()
    {
        var connectionString = "mongodb+srv://ledimoana:Test1234@cluster0.jgfyrtu.mongodb.net/?retryWrites=true&w=majority";
        var repository = new MongoRepository<Book>(connectionString, "LibraryDb", "Books");

        var newBook = new Book
        {
            Title = "MongoDB Test Book",
            Author = "Test Author"
        };

        await repository.AddAsync(newBook);
        Console.WriteLine("Книгу додано.");

        var books = await repository.GetAllAsync();
        Console.WriteLine("Список книг:");
        foreach (var book in books)
        {
            Console.WriteLine($"- {book.Title} від {book.Author}");
        }
    }
}
```

Результат в консолі:



Результат в БД:



БД створена через сайт
https://cloud.mongodb.com/v2#/org/68160468d36ccf1679644c89/projects/

Посилання на GitHub:
https://github.com/D1mon4ik1/NUPP_NET_2025_302_TK_Pysarenko_Lab/tree/lab3/