

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ПОЛТАВСЬКА ПОЛІТЕХНІКА  
ІМЕНІ ЮРІЯ КОНДРАТЮКА»**

**Навчально науковий інститут інформаційних технологій і робототехніки  
Кафедра комп'ютерних та інформаційних технологій і систем**

**Лабораторна робота № 2**

**з навчальної дисципліни  
«Технології на платформі NET»**

**Виконав:**

*Студент групи 403-ТН*

*Солонецький Роман Миколайович*

**Перевірив:**

*Тютюнник Петро Богданович*

## Тема: Багатопоковість. Асинхроність. IEnumerable. LINQ.

У межах роботи було модифіковано проєкт «Smart Home». До класів пристроїв додано статичні методи CreateNew() для автоматичної генерації випадкових об'єктів.

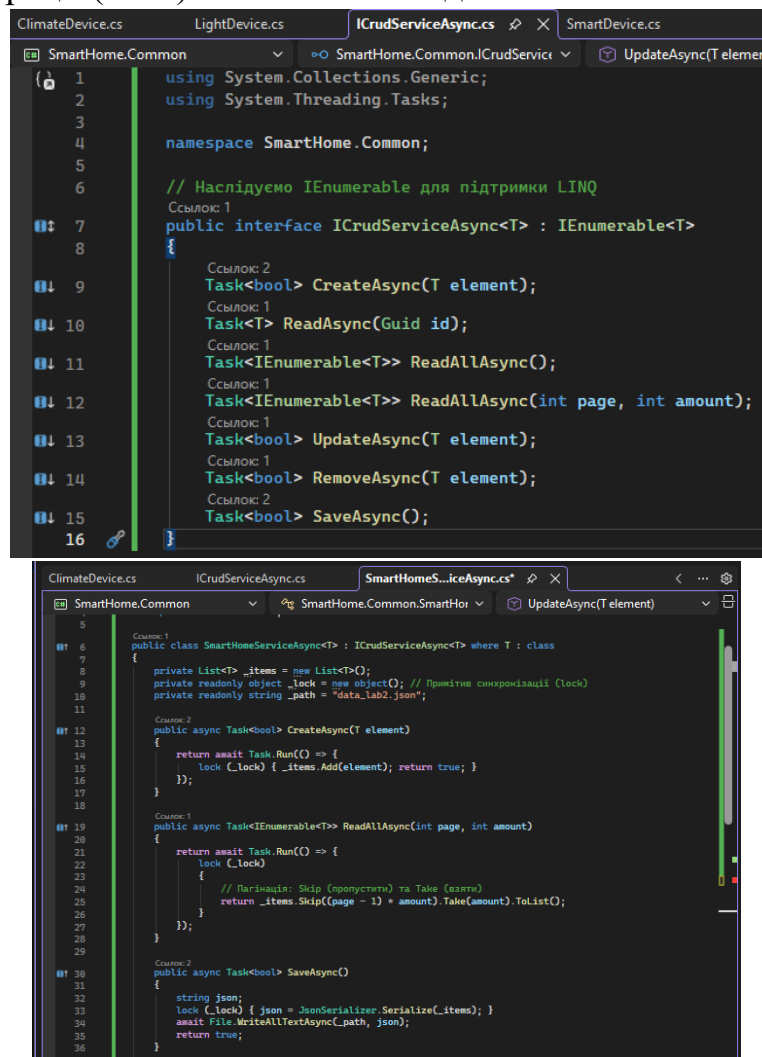
Код методів CreateNew() у класах LightDevice та ClimateDevice:

```
// Статичний метод для генерації (Завдання лаб 2)
Ссылка 1
public static LightDevice CreateNew()
{
    var rnd = new Random();
    return new LightDevice($"Лампа #{rnd.Next(1, 1000)}", rnd.Next(0, 101))
    {
        Color = "White",
        BulbType = "LED"
    };
}

// Статичний метод для генерації (Завдання лаб 2)
Ссылка 2
public static ClimateDevice CreateNew()
{
    var rnd = new Random();
    return new ClimateDevice($"Термостат #{rnd.Next(1, 1000)}", rnd.Next(16, 30))
    {
        Mode = "Auto"
    };
}
```

## Асинхронний CRUD сервіс

Реалізовано інтерфейс ICrudServiceAsync<T>, який підтримує асинхронні операції (Task) та є безпечним для багатьох потоків (thread-safe)



The screenshot displays the implementation of the `ICrudServiceAsync<T>` interface in the `SmartHomeServiceAsync.cs` file. The interface is defined in `ICrudServiceAsync.cs` and includes methods for creating, reading, updating, and deleting elements asynchronously. The implementation in `SmartHomeServiceAsync.cs` uses a `List<T>` to store data and a `lock` to ensure thread safety. The `CreateAsync` method adds a new element to the list. The `ReadAllAsync` method returns a list of elements, with an optional `page` and `amount` for pagination. The `UpdateAsync` method updates an existing element. The `RemoveAsync` method removes an element from the list. The `SaveAsync` method serializes the list to JSON and saves it to a file.

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

namespace SmartHome.Common;

// Наслідуюмо IEnumerable для підтримки LINQ
Ссылка 1
public interface ICrudServiceAsync<T> : IEnumerable<T>
{
    Ссылка 2
    Task<bool> CreateAsync(T element);
    Ссылка 1
    Task<T> ReadAsync(Guid id);
    Ссылка 1
    Task<IEnumerable<T>> ReadAllAsync();
    Ссылка 1
    Task<IEnumerable<T>> ReadAllAsync(int page, int amount);
    Ссылка 1
    Task<bool> UpdateAsync(T element);
    Ссылка 1
    Task<bool> RemoveAsync(T element);
    Ссылка 2
    Task<bool> SaveAsync();
}

Ссылка 1
public class SmartHomeServiceAsync<T> : ICrudServiceAsync<T> where T : class
{
    private List<T> _items = new List<T>();
    private readonly object _lock = new object(); // Примітка синхронізації (lock)
    private readonly string _path = "data_lab2.json";

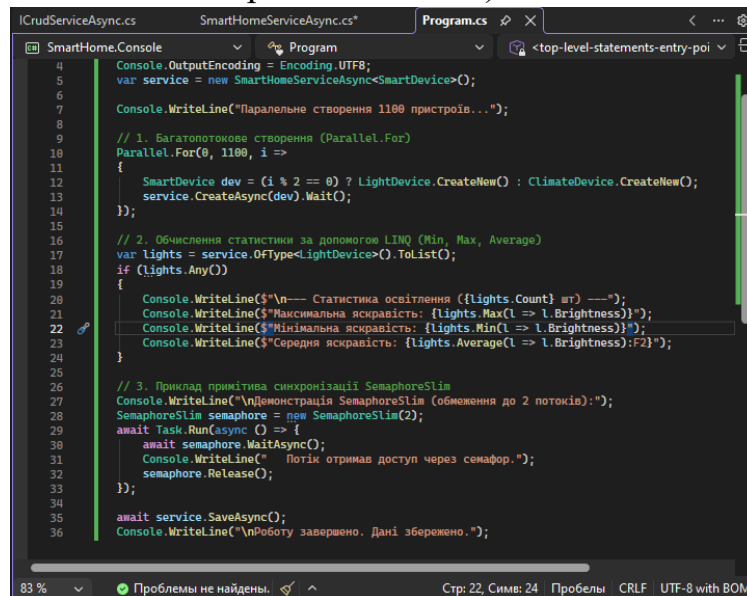
    Ссылка 2
    public async Task<bool> CreateAsync(T element)
    {
        return await Task.Run(() => {
            lock (_lock) { _items.Add(element); return true; }
        });
    }

    Ссылка 1
    public async Task<IEnumerable<T>> ReadAllAsync(int page, int amount)
    {
        return await Task.Run(() => {
            lock (_lock)
            {
                // Паринг: Skip (пропущено) та Take (взято)
                return _items.Skip((page - 1) * amount).Take(amount).ToList();
            }
        });
    }

    Ссылка 2
    public async Task<bool> SaveAsync()
    {
        string json;
        lock (_lock) { json = JsonSerializer.Serialize(_items); }
        await File.WriteAllTextAsync(_path, json);
        return true;
    }
}
```

## Робота з багатопотоковістю та LINQ

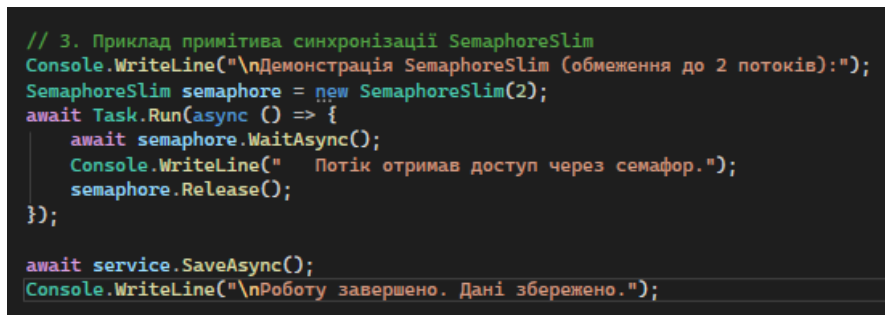
У консольному застосунку реалізовано паралельне створення 1100 об'єктів за допомогою `Parallel.For`. Після генерації проведено аналіз даних (мінімальні, максимальні та середні значення) за допомогою методів LINQ.



```
1 4 | ICrudServiceAsync.cs | SmartHomeServiceAsync.cs* | Program.cs | <top-level-statements-entry-poi>
2 |
3 | SmartHome.Console |
4 | Console.OutputEncoding = Encoding.UTF8;
5 | var service = new SmartHomeServiceAsync<SmartDevice>();
6 |
7 | Console.WriteLine("Паралельне створення 1100 пристроїв...");
8 |
9 | // 1. Багатопотокове створення (Parallel.For)
10 | Parallel.For(0, 1100, i =>
11 | {
12 |     SmartDevice dev = (i % 2 == 0) ? LightDevice.CreateNew() : ClimateDevice.CreateNew();
13 |     service.CreateAsync(dev).Wait();
14 | });
15 |
16 | // 2. Обчислення статистики за допомогою LINQ (Min, Max, Average)
17 | var lights = service.OfType<LightDevice>().ToList();
18 | if (lights.Any())
19 | {
20 |     Console.WriteLine($"--- Статистика освітлення ({lights.Count} шт) ---");
21 |     Console.WriteLine($"Максимальна яскравість: {lights.Max(l => l.Brightness)}");
22 |     Console.WriteLine($"Мінімальна яскравість: {lights.Min(l => l.Brightness)}");
23 |     Console.WriteLine($"Середня яскравість: {lights.Average(l => l.Brightness):F2}");
24 | }
25 |
26 | // 3. Приклад примітива синхронізації SemaphoreSlim
27 | Console.WriteLine("\nДемонстрація SemaphoreSlim (обмеження до 2 потоків):");
28 | SemaphoreSlim semaphore = new SemaphoreSlim(2);
29 | await Task.Run(async () => {
30 |     await semaphore.WaitAsync();
31 |     Console.WriteLine(" Потік отримав доступ через семафор.");
32 |     semaphore.Release();
33 | });
34 |
35 | await service.SaveAsync();
36 | Console.WriteLine("\nРоботу завершено. Дані збережено.");
```

## Примітиви синхронізації

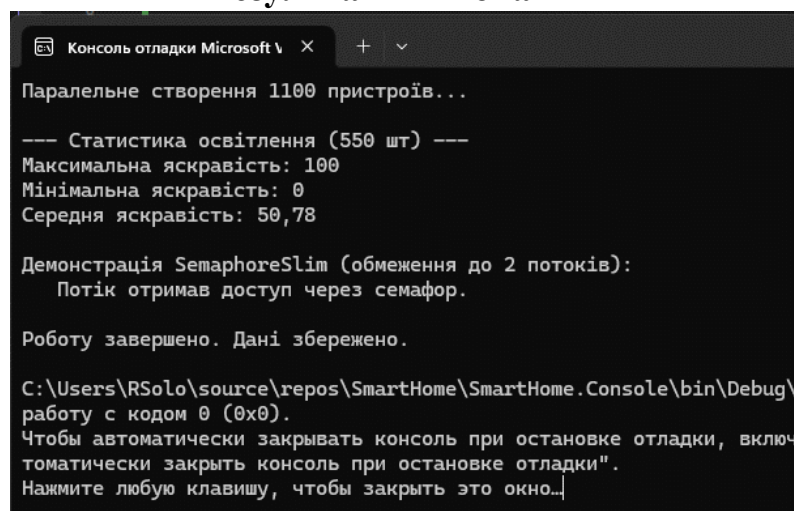
Продемонстровано використання примітивів синхронізації, таких як `lock` та `SemaphoreSlim`, для забезпечення безпечного доступу до спільних ресурсів



```
// 3. Приклад примітива синхронізації SemaphoreSlim
Console.WriteLine("\nДемонстрація SemaphoreSlim (обмеження до 2 потоків):");
SemaphoreSlim semaphore = new SemaphoreSlim(2);
await Task.Run(async () => {
    await semaphore.WaitAsync();
    Console.WriteLine(" Потік отримав доступ через семафор.");
    semaphore.Release();
});

await service.SaveAsync();
Console.WriteLine("\nРоботу завершено. Дані збережено.");
```

## Результати виконання



```
Консоль отладки Microsoft V X + v
Паралельне створення 1100 пристроїв...

--- Статистика освітлення (550 шт) ---
Максимальна яскравість: 100
Мінімальна яскравість: 0
Середня яскравість: 50,78

Демонстрація SemaphoreSlim (обмеження до 2 потоків):
Потік отримав доступ через семафор.

Роботу завершено. Дані збережено.

C:\Users\RSolo\source\repos\SmartHome\SmartHome.Console\bin\Debug\
роботу с кодом 0 (0x0).
Чтобы автоматически закрывать консоль при остановке отладки, включ
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно.
```

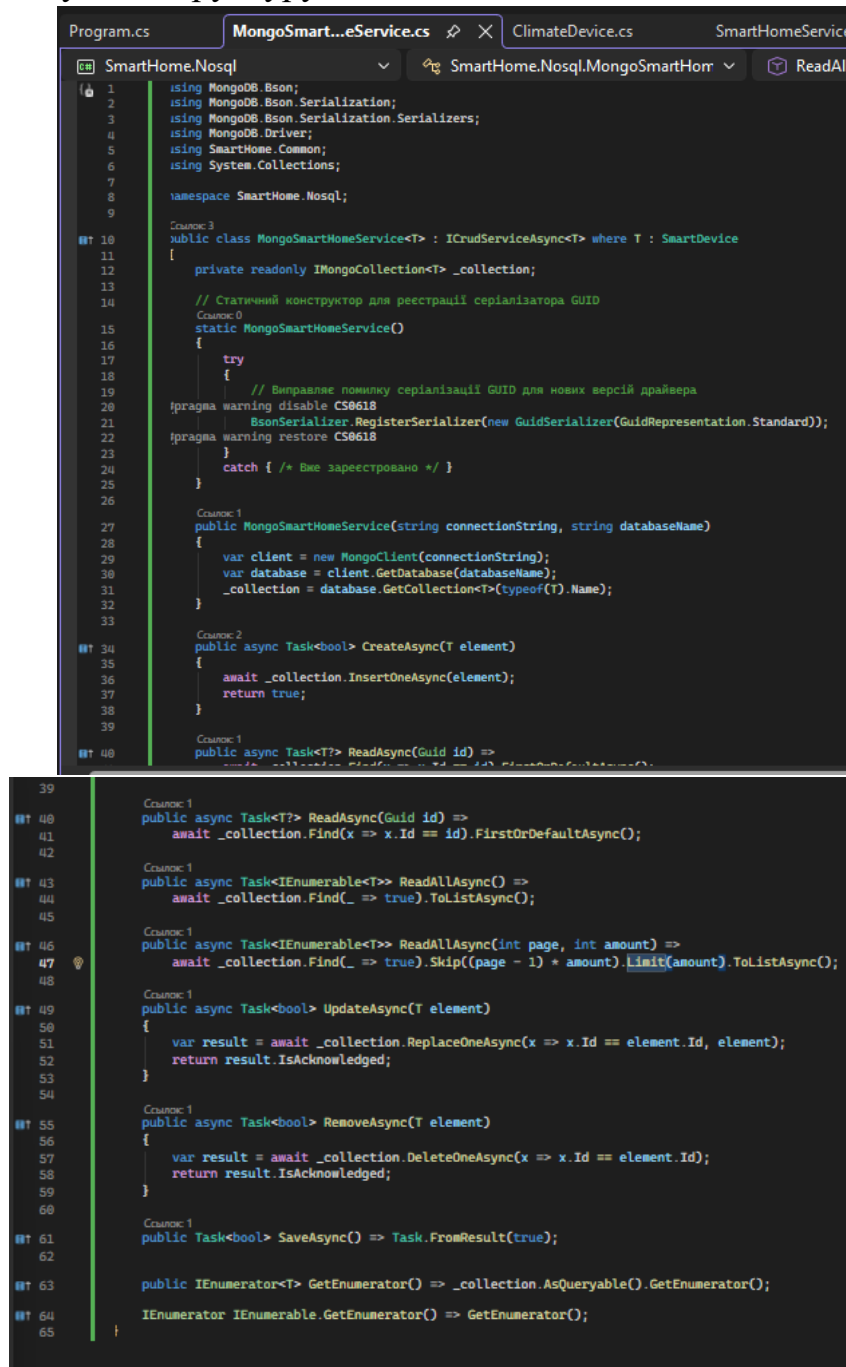
## ДОДАТКОВЕ ЗАВДАННЯ

### Реалізувати репозиторій для роботи з нереляційною базою даних (NoSQL) у проєкті SmartHome.Nosql

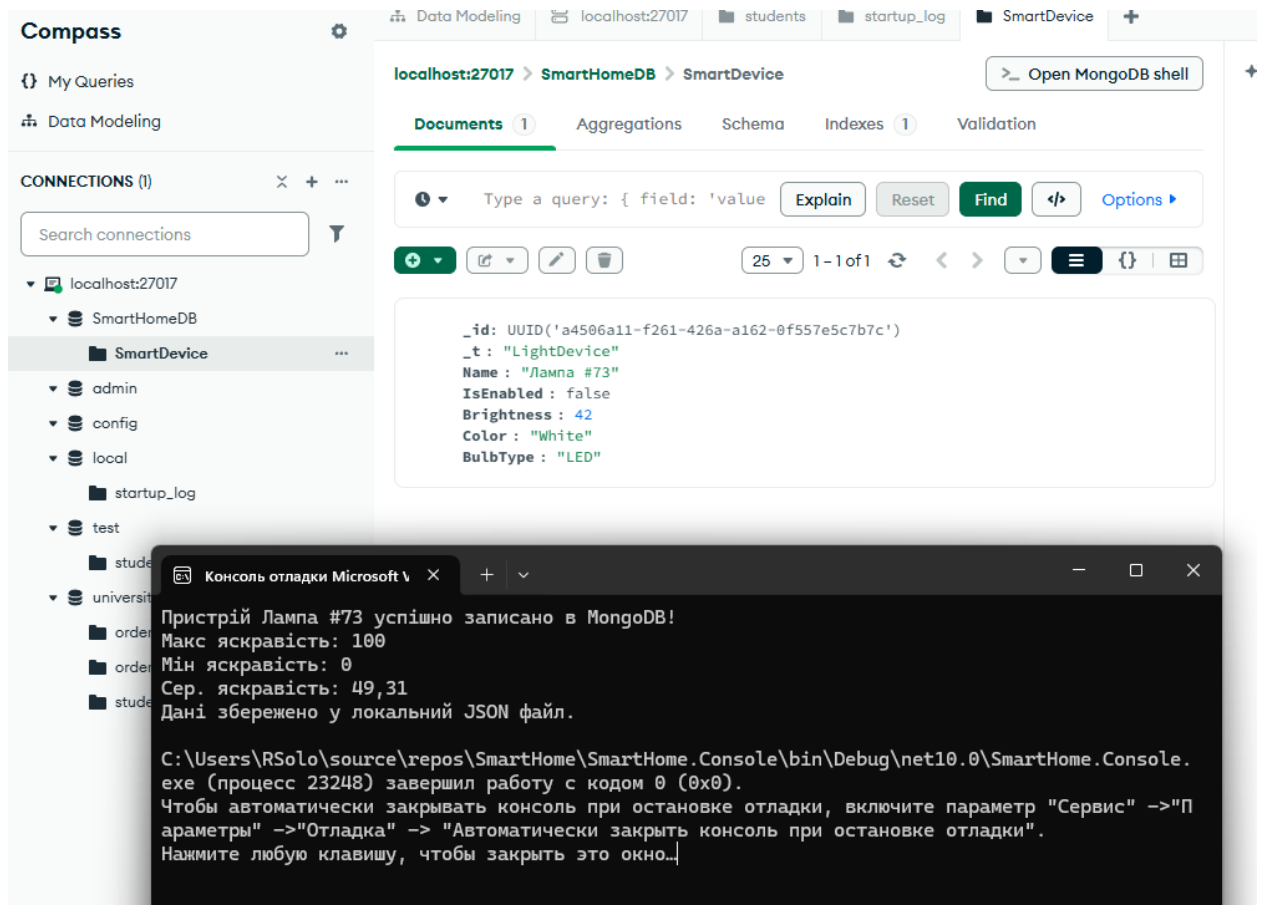
Було створено окрему бібліотеку класів SmartHome.Nosql, яка містить логіку взаємодії з базою даних MongoDB. Використано драйвер MongoDB.Driver для взаємодії з базою без використання Entity Framework.

Клас MongoSmartHomeService<T> реалізує раніше визначений інтерфейс ICrudServiceAsync<T>.

Оскільки MongoDB є документоорієнтованою БД, об'єкти LightDevice та ClimateDevice зберігаються як JSON-подібні документи (BSON), що дозволяє легко масштабувати структуру даних.



```
Program.cs  MongoSmart...eService.cs  ClimateDevice.cs  SmartHomeService.cs
SmartHome.Nosql
1 using MongoDB.Bson;
2 using MongoDB.Bson.Serialization;
3 using MongoDB.Bson.Serialization.Serializers;
4 using MongoDB.Driver;
5 using SmartHome.Common;
6 using System.Collections;
7
8 namespace SmartHome.Nosql;
9
10 // Ссылка 3
11 public class MongoSmartHomeService<T> : ICrudServiceAsync<T> where T : SmartDevice
12 {
13     private readonly IMongoCollection<T> _collection;
14
15     // Статичний конструктор для реєстрації серіалізатора GUID
16     // Ссылка 0
17     static MongoSmartHomeService()
18     {
19         try
20         {
21             // Виправляє помилку серіалізації GUID для нових версій драйвера
22            #pragma warning disable CS0618
23             BsonSerializer.RegisterSerializer(new GuidSerializer(GuidRepresentation.Standard));
24            #pragma warning restore CS0618
25         }
26         catch { /* Все зареєстровано */ }
27
28     }
29
30     // Ссылка 1
31     public MongoSmartHomeService(string connectionString, string databaseName)
32     {
33         var client = new MongoClient(connectionString);
34         var database = client.GetDatabase(databaseName);
35         _collection = database.GetCollection<T>(typeof(T).Name);
36     }
37
38     // Ссылка 2
39     public async Task<bool> CreateAsync(T element)
40     {
41         await _collection.InsertOneAsync(element);
42         return true;
43     }
44
45     // Ссылка 1
46     public async Task<T> ReadAsync(Guid id) =>
47     {
48         var element = await _collection.Find(x => x.Id == id).FirstOrDefaultAsync();
49         return element;
50     }
51
52     // Ссылка 1
53     public async Task<IEnumerable<T>> ReadAllAsync() =>
54     {
55         return await _collection.Find(_ => true).ToListAsync();
56     }
57
58     // Ссылка 1
59     public async Task<IEnumerable<T>> ReadAllAsync(int page, int amount) =>
60     {
61         return await _collection.Find(_ => true).Skip((page - 1) * amount).Limit(amount).ToListAsync();
62     }
63
64     // Ссылка 1
65     public async Task<bool> UpdateAsync(T element)
66     {
67         var result = await _collection.ReplaceOneAsync(x => x.Id == element.Id, element);
68         return result.IsAcknowledged;
69     }
70
71     // Ссылка 1
72     public async Task<bool> RemoveAsync(T element)
73     {
74         var result = await _collection.DeleteOneAsync(x => x.Id == element.Id);
75         return result.IsAcknowledged;
76     }
77
78     // Ссылка 1
79     public Task<bool> SaveAsync() => Task.FromResult(true);
80
81     public IEnumerable<T> GetEnumerator() => _collection.AsQueryable().GetEnumerator();
82     IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();
83 }
```



### Контрольні питання

**1. Що таке реляційні бази даних? Що таке СУБД? Які СУБД ви знаєте?**

Реляційні БД - це сховища, де дані організовані у вигляді пов'язаних таблиць. СУБД (Система управління базами даних) - ПЗ для створення та роботи з БД. Приклади: PostgreSQL, MS SQL Server, MySQL, SQLite, Oracle.

**2. Що позначає термін таблиця у реляційній БД? Які існують зв'язки у реляційній БД?**

Таблиця - структура з рядків (записів) та стовпців (полів).

Зв'язки: Один-до-одного (1:1), Один-до-багатьох (1:N), Багато-до-багатьох (N:M).

**3. Що таке ERD-діаграма? Як її створити та для чого вона потрібна?**

Це графічна схема сутностей та зв'язків між ними. Потрібна для візуального проектування структури бази перед написанням коду.

**4. Що таке DbContext і яку роль він відіграє в роботі з базою даних?**  
Головний клас в EF Core, що діє як міст між вашим кодом і базою даних. Він керує підключенням та відстежує зміни в об'єктах для їх збереження.

**5. Що таке зв'язки один-до-одного, один-до-багатьох і багато-до-багатьох у контексті Entity Framework Core?**

Це спосіб опису того, як класи посилаються один на одного (наприклад, через навігаційні властивості `public ICollection<Device> Devices { get; set; }`).

#### **6. Як реалізувати зв'язок один-до-одного за допомогою Fluent API?**

Реалізується в методі `OnModelCreating` за допомогою конструкції:  
`modelBuilder.Entity<A>().HasOne(a => a.B).WithOne(b => b.A).HasForeignKey<B>(b => b.AId);`.

#### **7. Наведіть приклад реалізації зв'язку багато-до-багатьох у EF Core.**

Створення колекцій в обох класах. EF Core 5.0+ автоматично створить проміжну таблицю: `public class User { public ICollection<Role> Roles { get; set; } }` та `public class Role { public ICollection<User> Users { get; set; } }`.

#### **8. У чому різниця між анотаціями (Data Annotations) та Fluent API в конфігурації моделей?**

Анотації - атрибути прямо в коді класу (наприклад, `[Key]`). Прості, але засмічують модель.

Fluent API - налаштування всередині `DbContext`. Потужніші, дозволяють налаштувати те, що неможливо зробити атрибутами.

#### **9. Як створюється та застосовується міграція в EF Core?**

Створення: `dotnet ef migrations add Name` (генерує код змін).

Застосування: `dotnet ef database update` (вносить зміни безпосередньо в базу).

#### **10. Яка мета проєкту {Назва тематики}.Infrastructure? Яку роль він відіграє у загальній структурі застосунку?**

Містить технічну реалізацію доступу до даних: налаштування контексту БД, репозиторії, роботу із зовнішніми API. Відокремлює "техніку" від бізнес-логіки.

#### **11. У чому різниця між доменною моделлю (з попередньої лабораторної) та моделлю бази даних?**

Доменна модель - об'єкт з бізнес-логікою.

Модель БД (Entity) - об'єкт, структура якого точно повторює таблицю в базі даних.

#### **12. Яку проблему вирішує шаблон проектування Репозиторій?**

Вирішує проблему прямого доступу до БД із бізнес-логіки. Він "ховає" складні запити за простими методами (`GetById`, `Add`), роблячи код незалежним від конкретної СУБД.

#### **13. У чому принципова різниця між реляційними та нереляційними базами даних?**

SQL: Жорстка структура (таблиці), зв'язки через ключі, транзакції (ACID).

NoSQL: Гнучка структура (документи JSON/BSON), висока швидкість запису, відсутність жорстких зв'язків, легке масштабування.