

**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-TH
Солонецький Роман Миколайович*

Перевірив:
Тютюнник Петро Богданович

Полтава 2025

Тема: Багатопоковість. Асинхроність. IEnumerables. LINQ.

У межах роботи було модифіковано проект «Smart Home». До класів пристрой 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 безпечною потоків (thread-safe)

```
ICrudServiceAsync.cs
1  using System.Collections.Generic;
2  using System.Threading.Tasks;
3
4  namespace SmartHome.Common;
5
6  // Наслідуємо IEnumerable для підтримки LINQ
7  public interface ICrudServiceAsync<T> : IEnumerable<T>
8  {
9      Task<bool> CreateAsync(T element);
10     Task<T> ReadAsync(Guid id);
11     Task<IEnumerable<T>> ReadAllAsync();
12     Task<IEnumerable<T>> ReadAllAsync(int page, int amount);
13     Task<bool> UpdateAsync(T element);
14     Task<bool> RemoveAsync(T element);
15     Task<bool> SaveAsync();
}
SmartHomeServiceAsync.cs
1  public class SmartHomeServiceAsync<T> : ICrudServiceAsync<T> where T : class
2  {
3      private List<T> _items = new List<T>();
4      private readonly object _lock = new object(); // Примітка синхронізації (lock)
5      private readonly string _path = "data_lab2.json";
6
7      public async Task<bool> CreateAsync(T element)
8      {
9          return await Task.Run(() => {
10              lock (_lock) { _items.Add(element); return true; }
11          });
12      }
13
14      public async Task<IEnumerable<T>> ReadAllAsync(int page, int amount)
15      {
16          return await Task.Run(() => {
17              lock (_lock)
18              {
19                  // Порядок: Skip (пропустити) та Take (зняти)
20                  return _items.Skip((page - 1) * amount).Take(amount).ToList();
21              }
22          });
23      }
24
25      public async Task<bool> SaveAsync()
26      {
27          string json;
28          lock (_lock) { json = JsonSerializer.Serialize(_items); }
29          await File.WriteAllTextAsync(_path, json);
30          return true;
31      }
32  }
```

Робота з багатопотоківістю 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.

The screenshot shows a Visual Studio code editor with three tabs: ICrudServiceAsync.cs, SmartHomeServiceAsync.cs*, and Program.cs. The Program.cs tab contains C# code for creating 1100 devices in parallel, calculating statistics using LINQ, and demonstrating SemaphoreSlim synchronization. The code includes comments explaining each section: 1. Parallel creation using Parallel.For, 2. Calculating statistics with LINQ, and 3. A semaphore example. The status bar at the bottom indicates no problems found and shows the current file is Program.cs.

```
4 Console.OutputEncoding = Encoding.UTF8;
5 var service = new SmartHomeServiceAsync<SmartDevice>();
6
7 Console.WriteLine("Паралельне створення 1100 пристрой...");
8
9 // 1. Багатопоткове 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.");
37
```

Примітиви синхронізації

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

The screenshot shows a Visual Studio code editor with the same Program.cs file as the previous screenshot. It highlights a section of code demonstrating the use of SemaphoreSlim to limit access to a shared resource to two threads at once. The code uses a semaphore to control access and releases it after each task completes.

```
// 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

The screenshot shows the Microsoft Console window with the output of the application. It first displays the parallel creation of 1100 devices, followed by the calculated statistics (max, min, average brightness). Then it shows the demonstration of SemaphoreSlim, where it limits access to the save operation to two threads at once. Finally, it concludes with a message that the work is completed and the data is saved.

```
Паралельне створення 1100 пристрой...
--- 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\
работу с кодом 0 (0x0).
Чтобы автоматически закрывать консоль при остановке отладки, включите
автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно...
```

ДОДАТКОВЕ ЗАВДАННЯ

Реалізувати репозиторій для роботи з нереляційною базою даних (NoSQL) у проекті 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.

The screenshot shows a code editor with several tabs open. The main tab is 'MongoSmart...eService.cs' which contains the implementation of the MongoSmartHomeService class. Other tabs include 'Program.cs', 'ClimateDevice.cs', and 'SmartHomeService.cs'. The code is written in C# and uses MongoDB's C# driver. It includes static registration of a BSON serializer for GUIDs, connection setup, and implementations for CreateAsync, ReadAsync, ReadAllAsync, UpdateAsync, RemoveAsync, SaveAsync, GetEnumerable, and GetQueryable methods.

```
using MongoDB.Bson;
using MongoDB.Bson.Serialization;
using MongoDB.Bson.Serialization.Serializers;
using MongoDB.Driver;
using SmartHome.Common;
using System.Collections;

namespace SmartHome.Nosql;

public class MongoSmartHomeService<T> : ICrudServiceAsync<T> where T : SmartDevice
{
    private readonly IMongoCollection<T> _collection;

    // Статичний конструктор для реєстрації серіалізатора GUID
    static MongoSmartHomeService()
    {
        try
        {
            // Виправляє помилку серіалізації GUID для нових версій драйвера
#pragma warning disable CS0618
            BsonSerializer.RegisterSerializer(new GuidSerializer(GuidRepresentation.Standard));
#pragma warning restore CS0618
        }
        catch { /* Все зареєстровано */ }
    }

    public MongoSmartHomeService(string connectionString, string databaseName)
    {
        var client = new MongoClient(connectionString);
        var database = client.GetDatabase(databaseName);
        _collection = database.GetCollection<T>(typeof(T).Name);
    }

    public async Task<bool> CreateAsync(T element)
    {
        await _collection.InsertOneAsync(element);
        return true;
    }

    public async Task<T?> ReadAsync(Guid id) =>
    {
        var result = await _collection.Find(x => x.Id == id).FirstOrDefaultAsync();
        return result;
    }

    public async Task<T?> ReadAsync() =>
    {
        var result = await _collection.Find(_ > true).ToListAsync();
        return result;
    }

    public async Task<IEnumerable<T>> ReadAllAsync(int page, int amount) =>
    {
        await _collection.Find(_ > true).Skip((page - 1) * amount).Limit(amount).ToListAsync();
    }

    public async Task<bool> UpdateAsync(T element)
    {
        var result = await _collection.ReplaceOneAsync(x => x.Id == element.Id, element);
        return result.IsAcknowledged;
    }

    public async Task<bool> RemoveAsync(T element)
    {
        var result = await _collection.DeleteOneAsync(x => x.Id == element.Id);
        return result.IsAcknowledged;
    }

    public Task<bool> SaveAsync() => Task.FromResult(true);

    public Ienumerator<T> GetEnumerator() => _collection.AsQueryable().GetEnumerator();
    Ienumerator<T> GetEnumerable() => GetEnumerator();
}
```

The screenshot shows the Compass MongoDB interface. On the left, the connections tree shows 'localhost:27017' with a 'SmartHomeDB' database selected, which contains a 'SmartDevice' collection. A single document is listed:

```

_id: UUID('a4506a11-f261-426a-a162-0f557e5c7b7c')
_t: "LightDevice"
Name: "Лампа #73"
.IsEnabled: false
Brightness: 42
Color: "White"
BulbType: "LED"

```

On the right, there is a Microsoft Visual Studio debugger console window titled 'Консоль отладки Microsoft'. It displays the following output:

```

Пристрій Лампа #73 успішно записано в MongoDB!
Макс яскравість: 100
Мін яскравість: 0
Сер. яскравість: 49,31
Дані збережено у локальний JSON файл.

C:\Users\RSolo\source\repos\SmartHome\Console\bin\Debug\net10.0\SmartHome.Console.exe (процесс 23248) завершил работу с кодом 0 (0x0).
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно...

```

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.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 налаштовувати 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 проекту {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.