**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**

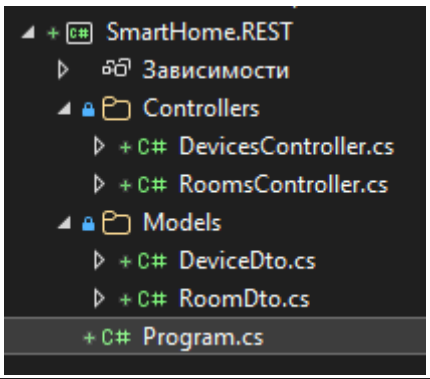
# Лабораторна робота № 4

з навчальної дисципліни
«Технології на платформі NET»

**Виконав:**
*Студент групи 403-ТН*
*Солонецький Роман Миколайович*

**Перевірив:**
*Тютюнник Петро Богданович*

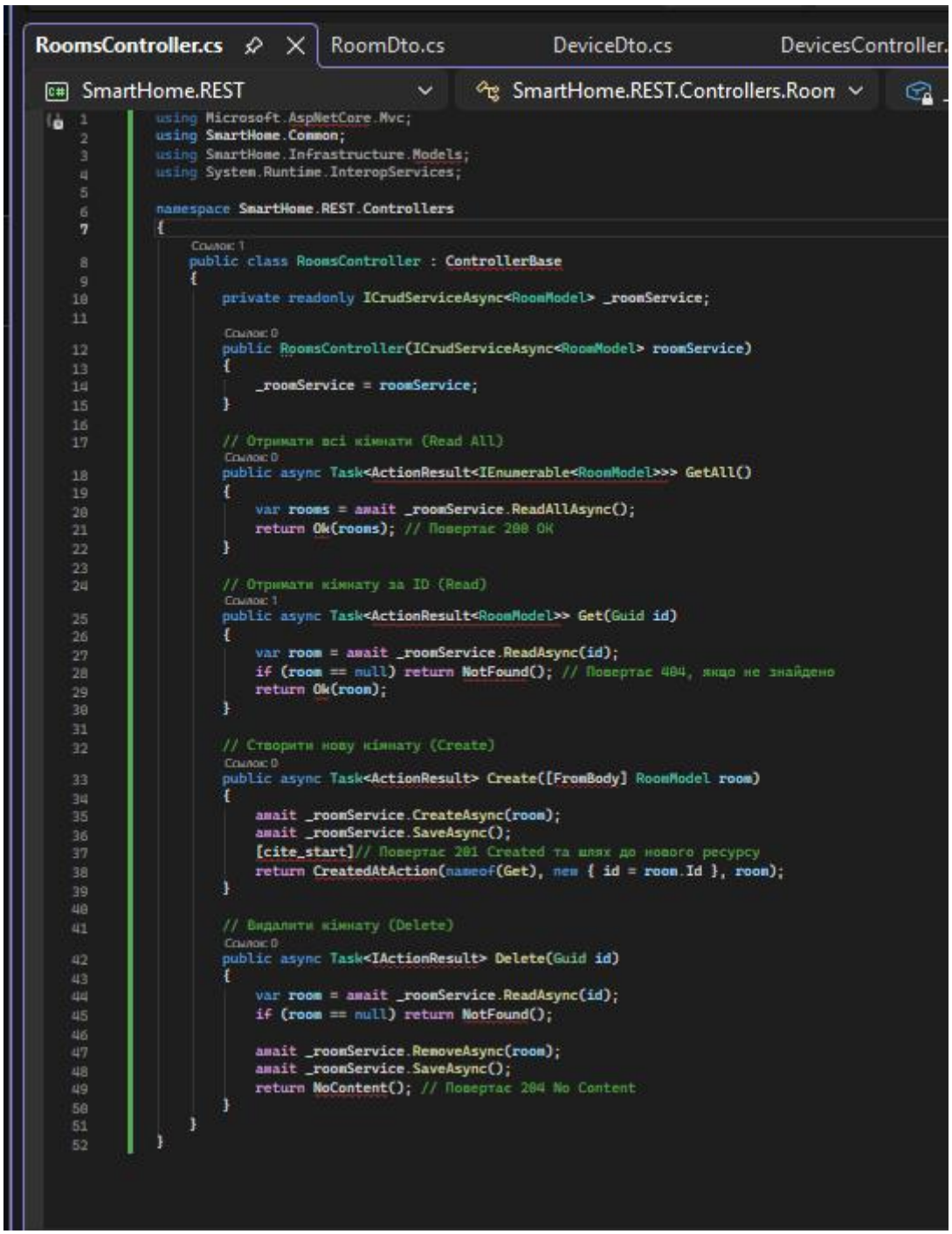
Полтава 2025



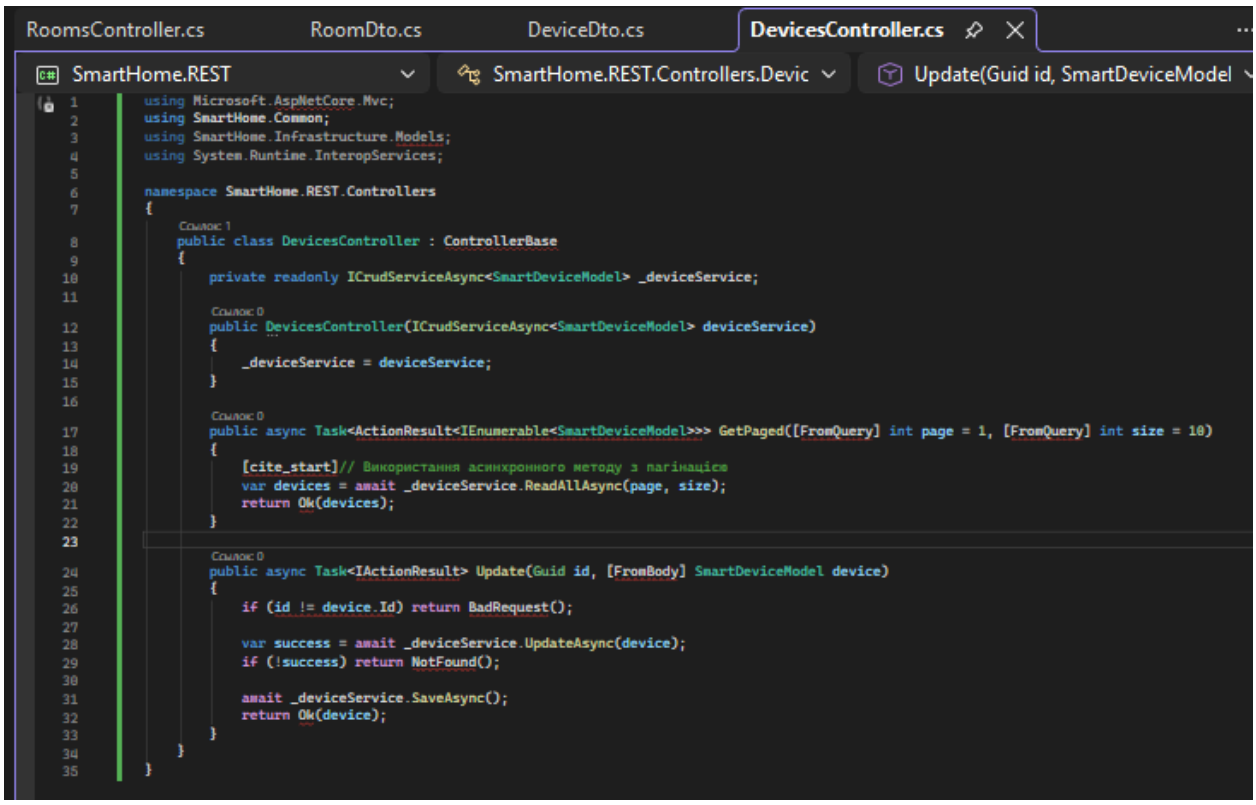

DevicesController.cs | Program.cs | Program.cs | RoomsController.cs | RoomDto.cs

SmartHome.REST | Program | <top-level-statements-entry-point

```
using Microsoft.EntityFrameworkCore;
using SmartHome.Common;
using SmartHome.Infrastructure;
using SmartHome.Infrastructure.Models;

var builder = WebApplication.CreateBuilder(args);

// 1. Реєстрація DbContext (SQLite)
builder.Services.AddDbContext<SmartHomeContext>(options =>
    options.UseSqlite("Data Source=../SmartHome.Console/smart_home.db")); // Шлях до БД з

// 2. Реєстрація Репозиторію та Сервісу (Dependency Injection)
// Використовуємо AddScoped, щоб об'єкти створювалися на кожен HTTP-запит
builder.Services.AddScoped<IRepository<SmartDeviceModel>, Repository<SmartDeviceModel>>();
builder.Services.AddScoped<ICrudServiceAsync<SmartDeviceModel>, SmartHomeServiceAsync<Smar

builder.Services.AddScoped<IRepository<RoomModel>, Repository<RoomModel>>();
builder.Services.AddScoped<ICrudServiceAsync<RoomModel>, SmartHomeServiceAsync<RoomModel>>

// 3. Додавання контролерів та Swagger
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// 4. Налаштування Middleware
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseAuthorization();
app.MapControllers();

app.Run();
```




```cs
using Microsoft.AspNetCore.Mvc;
using SmartHome.Common;
using SmartHome.Infrastructure.Models;
using System.Runtime.InteropServices;

namespace SmartHome.REST.Controllers
{
    Ссылок: 1
    public class RoomsController : ControllerBase
    {
        private readonly ICrudServiceAsync<RoomModel> _roomService;

        Ссылок: 0
        public RoomsController(ICrudServiceAsync<RoomModel> roomService)
        {
            _roomService = roomService;
        }

        // Отримати всі кімнати (Read All)
        Ссылок: 0
        public async Task<ActionResult<IEnumerable<RoomModel>>> GetAll()
        {
            var rooms = await _roomService.ReadAllAsync();
            return Ok(rooms); // Повертає 200 OK
        }

        // Отримати кімнату за ID (Read)
        Ссылок: 1
        public async Task<ActionResult<RoomModel>> Get(Guid id)
        {
            var room = await _roomService.ReadAsync(id);
            if (room == null) return NotFound(); // Повертає 404, якщо не знайдено
            return Ok(room);
        }

        // Створити нову кімнату (Create)
        Ссылок: 0
        public async Task<ActionResult> Create([FromBody] RoomModel room)
        {
            await _roomService.CreateAsync(room);
            await _roomService.SaveAsync();
            [cite_start]// Повертає 201 Created та шлях до нового ресурсу
            return CreatedAtAction(nameof(Get), new { id = room.Id }, room);
        }

        // Видалити кімнату (Delete)
        Ссылок: 0
        public async Task<IActionResult> Delete(Guid id)
        {
            var room = await _roomService.ReadAsync(id);
            if (room == null) return NotFound();

            await _roomService.RemoveAsync(room);
            await _roomService.SaveAsync();
            return NoContent(); // Повертає 204 No Content
        }
    }
}
```




## Контрольні запитання

**1. Які ключові особливості шаблону ASP.NET Core Web API та чим він відрізняється від Blazor Web App або ASP.NET MVC?**

ASP.NET Core Web API фокусується на створенні RESTful сервісів, які повертають дані (зазвичай у форматі JSON), а не готові інтерфейси користувача. На відміну від нього, MVC та Blazor призначені для повноцінної веброзробки, де сервер повертає відрендерені HTML-сторінки або інтерактивні компоненти.

**2. Поясніть, як реалізується трирівнева архітектура у вашому проєкті {Назва тематики}.REST та які переваги вона має.**

Архітектура реалізується шляхом поділу проєкту на рівень представлення (контролери API), рівень бізнес-логіки (CRUD-сервіси) та рівень доступу до даних (репозиторії та контекст БД). Це забезпечує легкість підтримки, тестування та можливість незалежної зміни окремих шарів застосунку.

**3. Навіщо створювати окремі моделі у папці Models та в чому різниця між цими моделями й сутностями, що зберігаються в базі даних?**

Окремі моделі (DTO) створюються для того, щоб передавати клієнту лише необхідні дані, приховуючи внутрішню структуру бази даних. Сутності БД

описують таблиці, тоді як моделі в контролерах адаптовані під потреби конкретних HTTP-запитів.

**4. Що таке REST і як у вашому API забезпечується дотримання принципу однорідності інтерфейсу (uniform interface)?**

REST — це архітектурний стиль побудови веблежних систем, а однорідність інтерфейсу забезпечується використанням унікальних назв сутностей, правильних HTTP-методів та стандартних кодів відповідей. Це робить взаємодію з API передбачуваною та уніфікованою для будь-яких клієнтів.

**5. Які HTTP-методи використовуються для CRUD-операцій? Наведіть приклади відповідності методів діям у контролерах.**

Для операцій використовуються методи POST (створення), GET (читання), PUT (оновлення) та DELETE (видалення). Наприклад, метод GET у контролері відповідає дії ReadAllAsync для отримання списку всіх сутностей.

**6. Які HTTP-коди відповідей є коректними для сценаріїв успіху та обробки помилок у вашому API (створення, оновлення, видалення, не знайдено тощо)?**

Для успішного створення використовується код 201 Created, для загального успіху — 200 OK, а для видалення — 204 No Content. У разі помилок повертається 404 Not Found, якщо ресурс відсутній, або інші коди помилок клієнта/сервера.

**7. Опишіть інтерфейс ICrudServiceAsync<T>: для чого потрібні асинхронні методи та як вони впливають на продуктивність застосунку?**

Цей інтерфейс визначає асинхронні методи, такі як CreateAsync та ReadAsync, для виконання операцій без блокування потоків сервера. Асинхронність значно підвищує продуктивність, дозволяючи обробляти більше одночасних запитів за рахунок ефективного очікування операцій вводу-виводу.

**8. Як працює механізм впровадження залежностей (Dependency Injection) в ASP.NET Core і як ви зареєстрували свої сервіси та репозиторії?**

DI автоматично надає необхідні об'єкти (залежності) класам через їхні конструктори, що спрощує керування життєвим циклом компонентів. Реєстрація сервісів, репозиторіїв та контексту БД відбувається у файлі конфігурації проєкту за допомогою вбудованих методів контейнера залежностей.

**9. Як організована пагінація у вашому CRUD-сервісі (ReadAllAsync(int page, int amount)) та для чого вона потрібна?**

Пагінація реалізована через параметри номера сторінки та кількості елементів, що дозволяє викачувати дані невеликими порціями. Вона потрібна для зменшення навантаження на сервер і мережу при роботі з великими масивами даних.

**10. Яким чином ви тестували роботу свого API, які інструменти або сторінки використовували, та які результати следует включити в PDF-файл для PR?**

Тестування проводилося за допомогою Swagger або Postman шляхом відправки HTTP-запитів до створених контролерів. У PDF-файл необхідно включити скріншоти результатів виконання запитів, що демонструють коректні дані та HTTP-коди відповідей.

**11. Які кроки потрібно виконати, щоб створити проєкт {Назва тематики}.MVC, та чим відрізняється його структура від REST API?**

Необхідно створити проєкт за шаблоном ASP.NET Core Web App (MVC) та реалізувати вебінтерфейс, використовуючи контролери та представлення (Views). На відміну від REST API, структура MVC проєкту включає папки для моделей, контролерів та сторінок Razor для візуалізації даних користувачу.

**12. Як ви повторно використали моделі та контекст бази даних з третьої лабораторної у новому MVC/Blazor-застосунку?**

Контекст БД та сутності підключаються як посилання на проєкт із попередньої лабораторної роботи, що дозволяє використовувати вже існуючу схему даних. На основі цих моделей за допомогою вбудованої генерації (scaffolding) створюються сторінки для автоматичної реалізації CRUD-операцій у вебінтерфейсі.