МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

**Практична робота № 3**

з дисципліни «Архітектура та проектування програмного забезпечення .Net»

*назва дисципліни*

на тему: «Розробка RESTful Web API (Service Layer / API)»

Виконав: студент 2 курсу групи № 621п

освітньої програми

121 інженерія програмного забезпечення

(шифр і назва ОП)

Головкіна О. А.

(прізвище й ініціали студента)

Прийняв: к.т.н., доцент каф. 603

Лучшев П. О.

(посада, науковий ступінь, прізвище й ініціали)

Кількість балів:

Харків – 2025

ПРАКТИЧНА РОБОТА № 3

Розробка RESTful Web API (Service Layer / API)

ЗМІСТ

[МЕТА РОБОТИ 3](#_Toc199677689)

[ПОСТАНОВКА ЗАДАЧІ 3](#_Toc199677690)

[ВИКОНАННЯ РОБОТИ 4](#_Toc199677691)

[Створення RESTful Web API 4](#_Toc199677692)

[CRUD-контролери 5](#_Toc199677693)

[Swagger (OpenAPI) 10](#_Toc199677694)

[GitHub 11](#_Toc199677695)

[Система оцінювання роботи 11](#_Toc199677696)

[ВИСНОВКИ 12](#_Toc199677697)

МЕТА РОБОТИ

Вивчення особливостей створення та використання Web API на базі ASP.NET Core, його інтеграція з модулем доступу до даних (Data Access Layer), а також ознайомлення з набором інструментів Swagger та написанням інтеграційних тестів.

ПОСТАНОВКА ЗАДАЧІ

У цій практичній роботі необхідно розробити Web API-застосунок, який забезпечує доступ до бази даних за допомогою HTTP-запитів у стилі REST. Для роботи з БД і рівнем доступу до даних (Data Access Layer) використовуемо модуль який був створений в рамках попередньої практичної роботи (моделі сутностей предметної області, контекст DbContext, використання Entity Framework Core + SQLite). Таким чином у межах поточної роботи необхідно:

1. Створити Web API-проєкт (ASP.NET Core Web API, .NET 6+)
2. Підключити проєкт із Data Access Layer (проєкт із моделями та DbContext);
3. Реалізувати контролери (Controllers) з повним CRUD-функціоналом для трьох сутностей предметної області, використовуючи Scaffolded Items або ручне створення;
4. Інтегрувати Swagger (OpenAPI) для автоматичної генерації документації та ручного тестування API;
5. Реалізувати інтеграційне тестування API за допомогою створення окремого набору тестів у окремому проєкту Tests (з попередньої практичної роботи);
6. Виконати тестування основних HTTP-запитів (GET, POST, PUT, DELETE).

В якості варіативної частини необхідно використовувати модель даних з трьома сутностями відповідно до своєї предметної області, яку було реалізовано у попередній практичній роботі.

ВИКОНАННЯ РОБОТИ

Створення RESTful Web API

Було створено проєкт ASP.NET Core Web API, що використовує фреймворк .NET 8 для Windows. В основі проєкту лежать моделі предметної області: Detail, Operation та Production. Ці моделі було попередньо реалізовано та перенесено в новий проєкт. Було налаштовано підключення до бази даних через Entity Framework Core, із використанням контексту AppDbContext.

На рисунку 1 продемонстрована структура проєкту:

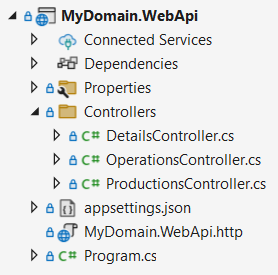


Рисунок 1 – Структура проєкту

Вміст файлу Program.cs:

using DataAccessLayer;

using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

if (builder.Environment.IsEnvironment("Testing"))

{

builder.Services.AddDbContext<AppDbContext>(options =>

options.UseInMemoryDatabase("TestDb"));

}

else

{

builder.Services.AddDbContext<AppDbContext>(options =>

options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));

}

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

CRUD-контролери

Для кожної з трьох сутностей (Detail, Operation, Production) були створені контролери з повним набором CRUD-операцій. Вони забезпечують можливість виконання запитів GET для отримання даних, POST для створення нових записів, PUT для оновлення існуючих даних та DELETE для їх видалення. Контролери були розроблені вручну, з урахуванням структури ключів і навігаційних властивостей.

Вміст файлів:

* DetailsController.cs:

using DataAccessLayer;

using DomainTables;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

[ApiController]

[Route("api/[controller]")]

public class DetailsController : ControllerBase

{

private readonly AppDbContext \_context;

public DetailsController(AppDbContext context)

{

\_context = context;

}

[HttpGet]

public async Task<ActionResult<IEnumerable<Detail>>> GetDetails()

{

return await \_context.Details.ToListAsync();

}

[HttpGet("{id}")]

public async Task<ActionResult<Detail>> GetDetail(int id)

{

var detail = await \_context.Details.FindAsync(id);

if (detail == null)

{

return NotFound();

}

return detail;

}

[HttpPost]

public async Task<ActionResult<Detail>> PostDetail(Detail detail)

{

\_context.Details.Add(detail);

await \_context.SaveChangesAsync();

return CreatedAtAction(nameof(GetDetail), new { id = detail.CodeDetail }, detail);

}

[HttpPut("{id}")]

public async Task<IActionResult> PutDetail(int id, Detail detail)

{

if (id != detail.CodeDetail)

{

return BadRequest();

}

\_context.Entry(detail).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!\_context.Details.Any(e => e.CodeDetail == id))

{

return NotFound();

}

throw;

}

return NoContent();

}

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteDetail(int id)

{

var detail = await \_context.Details.FindAsync(id);

if (detail == null)

{

return NotFound();

}

\_context.Details.Remove(detail);

await \_context.SaveChangesAsync();

return NoContent();

}

}

* OperationsController.cs:

using DataAccessLayer;

using DomainTables;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

[ApiController]

[Route("api/[controller]")]

public class OperationsController : ControllerBase

{

private readonly AppDbContext \_context;

public OperationsController(AppDbContext context)

{

\_context = context;

}

[HttpGet]

public async Task<ActionResult<IEnumerable<Operation>>> GetOperations()

{

return await \_context.Operations.ToListAsync();

}

[HttpGet("{id}")]

public async Task<ActionResult<Operation>> GetOperation(int id)

{

var operation = await \_context.Operations.FindAsync(id);

if (operation == null)

{

return NotFound();

}

return operation;

}

[HttpPost]

public async Task<ActionResult<Operation>> PostOperation(Operation operation)

{

\_context.Operations.Add(operation);

await \_context.SaveChangesAsync();

return CreatedAtAction(nameof(GetOperation), new { id = operation.CodeOperation }, operation);

}

[HttpPut("{id}")]

public async Task<IActionResult> PutOperation(int id, Operation operation)

{

if (id != operation.CodeOperation)

{

return BadRequest();

}

\_context.Entry(operation).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!\_context.Operations.Any(e => e.CodeOperation == id))

{

return NotFound();

}

throw;

}

return NoContent();

}

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteOperation(int id)

{

var operation = await \_context.Operations.FindAsync(id);

if (operation == null)

{

return NotFound();

}

\_context.Operations.Remove(operation);

await \_context.SaveChangesAsync();

return NoContent();

}

}

* ProductionController.cs:

using DataAccessLayer;

using DomainTables;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

[ApiController]

[Route("api/[controller]")]

public class ProductionsController : ControllerBase

{

private readonly AppDbContext \_context;

public ProductionsController(AppDbContext context)

{

\_context = context;

}

[HttpGet]

public async Task<ActionResult<IEnumerable<Production>>> GetProductions()

{

return await \_context.Productions

.Include(p => p.Detail)

.Include(p => p.Operation)

.ToListAsync();

}

[HttpGet("{codeDetail}/{operationNumber}/{codeOperation}")]

public async Task<ActionResult<Production>> GetProduction(short codeDetail, short operationNumber, short codeOperation)

{

var production = await \_context.Productions

.Include(p => p.Detail)

.Include(p => p.Operation)

.FirstOrDefaultAsync(p =>

p.CodeDetail == codeDetail &&

p.OperationNumber == operationNumber &&

p.CodeOperation == codeOperation);

if (production == null)

{

return NotFound();

}

return production;

}

[HttpPost]

public async Task<ActionResult<Production>> PostProduction(Production production)

{

\_context.Productions.Add(production);

await \_context.SaveChangesAsync();

return CreatedAtAction(nameof(GetProduction),

new

{

codeDetail = production.CodeDetail,

operationNumber = production.OperationNumber,

codeOperation = production.CodeOperation

},

production);

}

[HttpPut("{codeDetail}/{operationNumber}/{codeOperation}")]

public async Task<IActionResult> PutProduction(short codeDetail, short operationNumber, short codeOperation, Production production)

{

if (codeDetail != production.CodeDetail ||

operationNumber != production.OperationNumber ||

codeOperation != production.CodeOperation)

{

return BadRequest("Composite key mismatch.");

}

\_context.Entry(production).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

var exists = await \_context.Productions.AnyAsync(p =>

p.CodeDetail == codeDetail &&

p.OperationNumber == operationNumber &&

p.CodeOperation == codeOperation);

if (!exists)

{

return NotFound();

}

throw;

}

return NoContent();

}

[HttpDelete("{codeDetail}/{operationNumber}/{codeOperation}")]

public async Task<IActionResult> DeleteProduction(short codeDetail, short operationNumber, short codeOperation)

{

var production = await \_context.Productions.FirstOrDefaultAsync(p =>

p.CodeDetail == codeDetail &&

p.OperationNumber == operationNumber &&

p.CodeOperation == codeOperation);

if (production == null)

{

return NotFound();

}

\_context.Productions.Remove(production);

await \_context.SaveChangesAsync();

return NoContent();

}

}

Swagger (OpenAPI)

Було підключено й налаштовано Swagger (OpenAPI), що дозволило автоматично генерувати документацію до API та забезпечити можливість ручного тестування ендпоінтів.

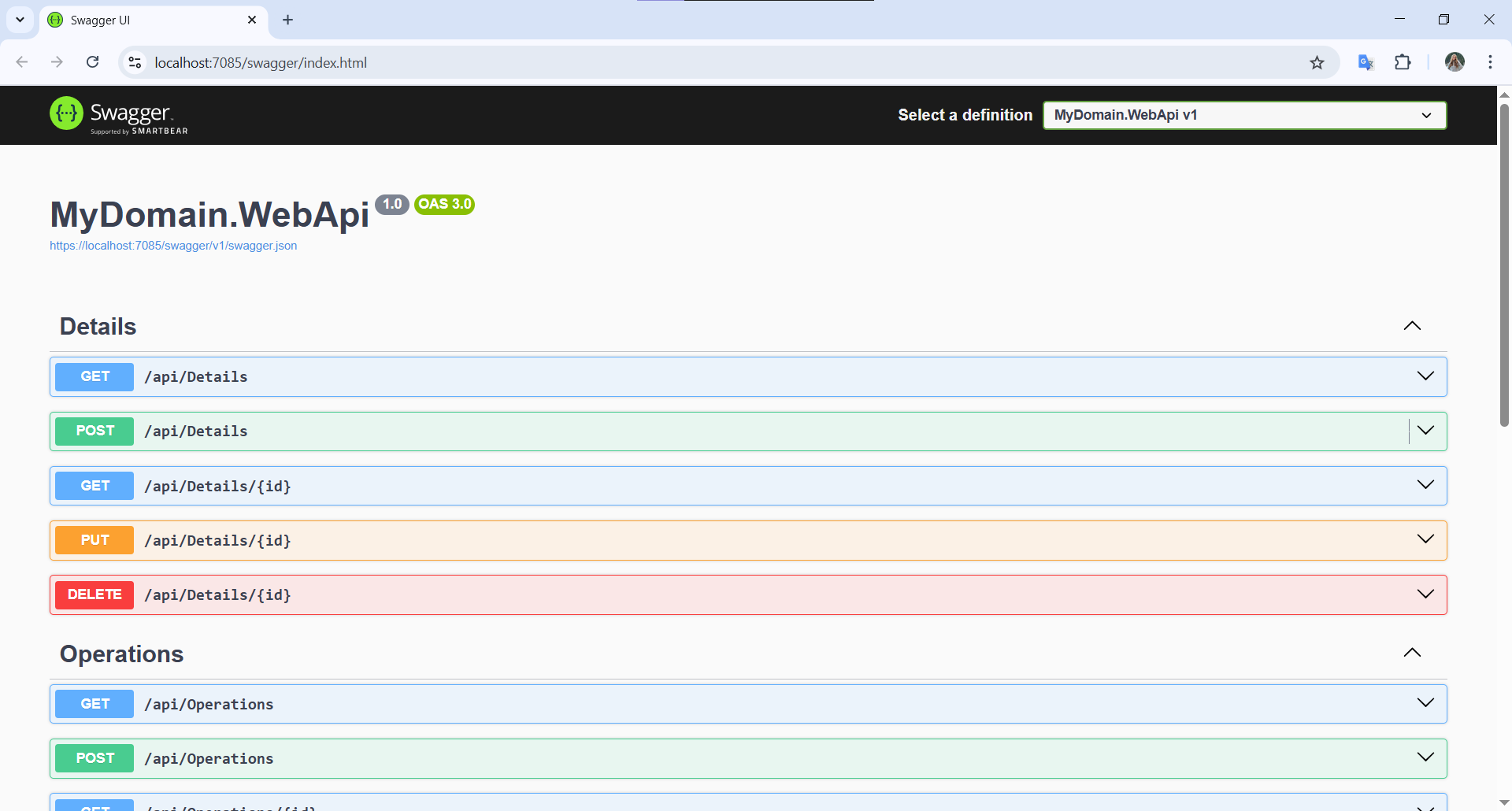


Рисунок 2 – Підключений і налаштований Swagger (OpenAPI)

****GitHub****

Усі файли проєкту були завантажені до репозиторію на GitHub, що забезпечує можливість віддаленого доступу, версіонування й контролю змін. Посилання на репозиторій: <https://github.com/o-a-golovkina/03-o-a-golovkina>.

Система оцінювання роботи

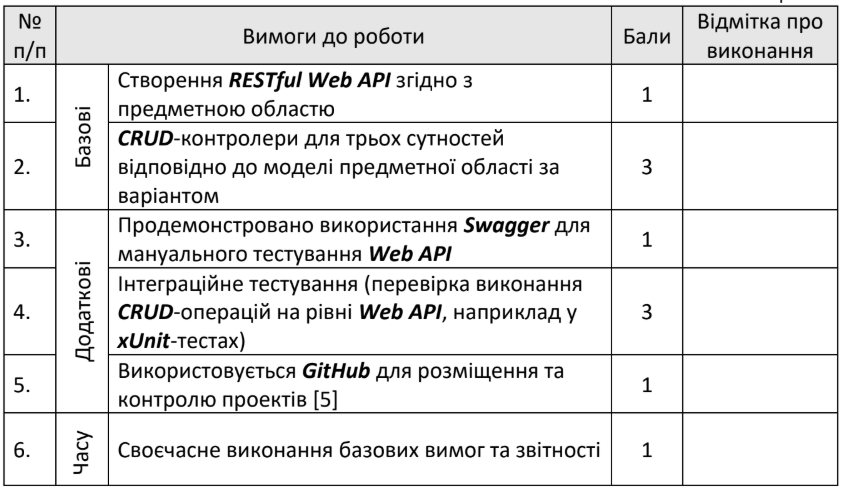


Рисунок 3 – Таблиця з системи оцінювання

ВИСНОВКИ

Під час виконання цієї роботи було отримано практичні навички зі створення RESTful Web API на ASP.NET Core, побудови CRUD-контролерів, налаштування й використання Swagger для документування та тестування ендпоінтів, а також реалізації інтеграційних тестів на рівні API. Окремо варто відзначити роботу з InMemoryDatabase, що дозволила тестувати API без взаємодії з реальною базою даних, та використання GitHub як засобу керування версіями проєкту. Серед труднощів, що виникали, були налаштування середовища тестування, вирішення проблеми сумісності .NET-версій та забезпечення правильного копіювання залежностей для тестового середовища. Завдяки наполегливості всі проблеми були вирішені, а всі базові та додаткові вимоги до роботи виконані.