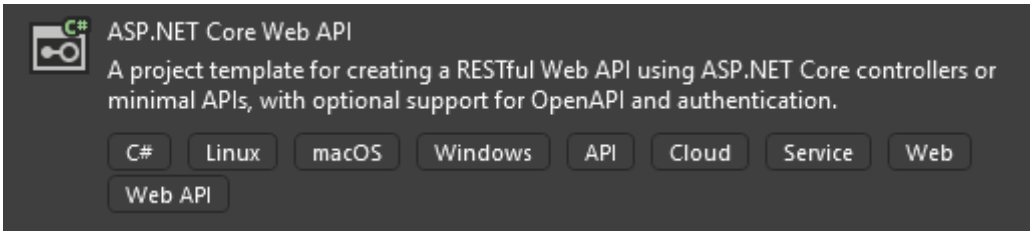
Лабораторна робота №4

Розробка вебсерверів засобами фреймворку ASP.NET.

Виконав студент групи 302-ТК

Писаренко Дмитрій

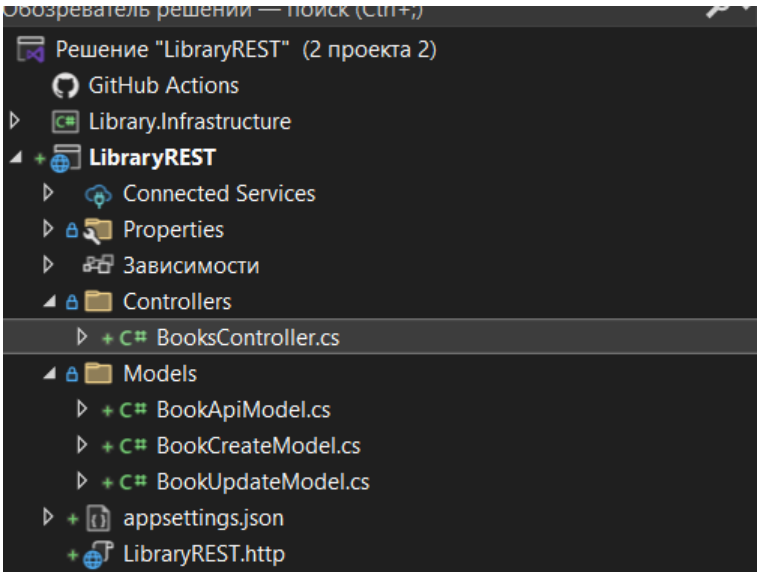
1. Створити проєкт за шаблоном ASP.NET Core Web API {Назва тематики}.REST використовуючи Visual Studio або .NET SDK: dotnet new webapi -n {Назва тематики}.REST



2. Створити папку Models, у якій будуть зберігатися моделі, які будуть використовуватися контролерами у проєкті {Назва тематики}.REST.

3. Створіть контролери у папці Controllers, які будуть описувати REST API для виконання CRUD операцій (створення, читання, оновлення, видалення) над моделями у вашій тематиці, які ви зберігали у базі даних у третій лабораторній, при цьому моделі, що використовуються у контролерах, повинні бути описані у проєкті {Назва тематики}.REST, щоб відповідати 3-рівневій архітектурі. Наприклад, якщо у вас є сутність Bus, що зберігається у БД, ви створите новий клас BusModel у папці Models нового проєкту, який використовуватиметься у контролерах. Також зверніть увагу, щоб моделі у контролерах, мали лишень необхідні властивості, та якщо якісь властивості зайві, для якихось із методів - створіть нову модель без цих властивостей.

Архітектура:



Контролер:


BookUpdateModel.cs*
BookCreateModel.cs*
BooksController.cs*
BookApiM
LibraryREST
LibraryREST.Controllers.BooksContro
using Microsoft.AspNetCore.Mvc;
using Library.Infrastructure.Models;
using LibraryREST.Models;
namespace LibraryREST.Controllers
{
[ApiController]
[Route("api/[controller]")]
Ссылок: 0
public class BooksController : ControllerBase
{
private static List<BookModel> _books = new()
{
new BookModel
{
Id = 1,
Title = "1984",
Author = "George Orwell",
LibraryMemberId = 1,
BookTags = new List<BookTagModel>()
},
new BookModel
{
Id = 2,
Title = "The Hobbit",
Author = "J.R.R. Tolkien",
LibraryMemberId = 2,
BookTags = new List<BookTagModel>()
}
};
// GET: api/Books
[HttpGet]
Ссылок: 0
public ActionResult<IEnumerable<BookApiModel>> GetAll()
{
var result = _books.Select(book => new BookApiModel
{
Id = book.Id,
Title = book.Title,
Author = book.Author
});
return Ok(result);
}
// GET: api/Books/1
[HttpGet("{id}")]
Ссылок: 1
public ActionResult<BookApiModel> Get(int id)
{
var book = _books.FirstOrDefault(b => b.Id == id);

4. При проєктуванні Web API зверніть увагу, щоб створений API відповідав 4-ій умові побудови REST-додатку по Філдингу - однорідності інтерфейсу (див. 31 слайд), тобто використовувалися унікальні назви сутностей, правильні HTTP методи та поверталися правильні HTTP коди у HTTP відповідях, наприклад 404 якщо сутність не знайдена, або 201 якщо нова сутність створена.

HTTP методи:

- GET — для отримання (список і по ID).
- POST — для створення.
- PUT — для оновлення.
- DELETE — для видалення.

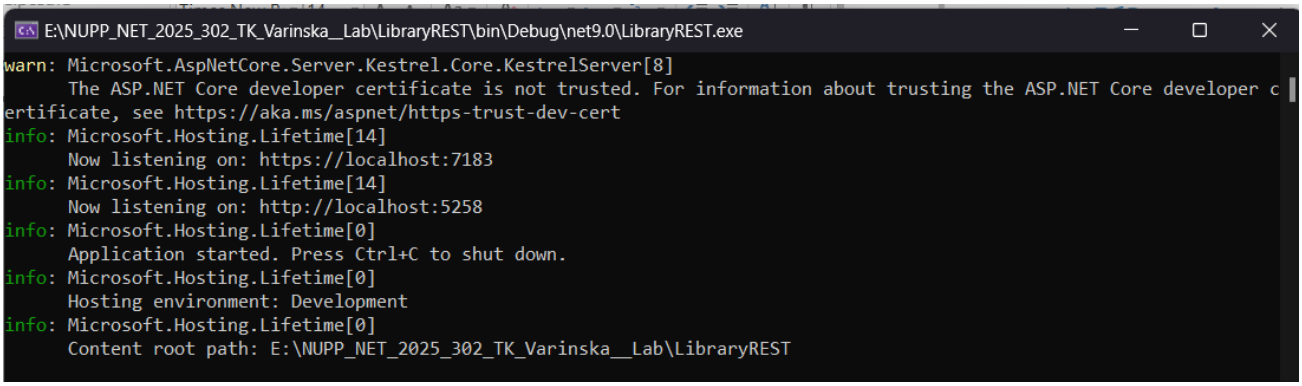
HTTP статуси:

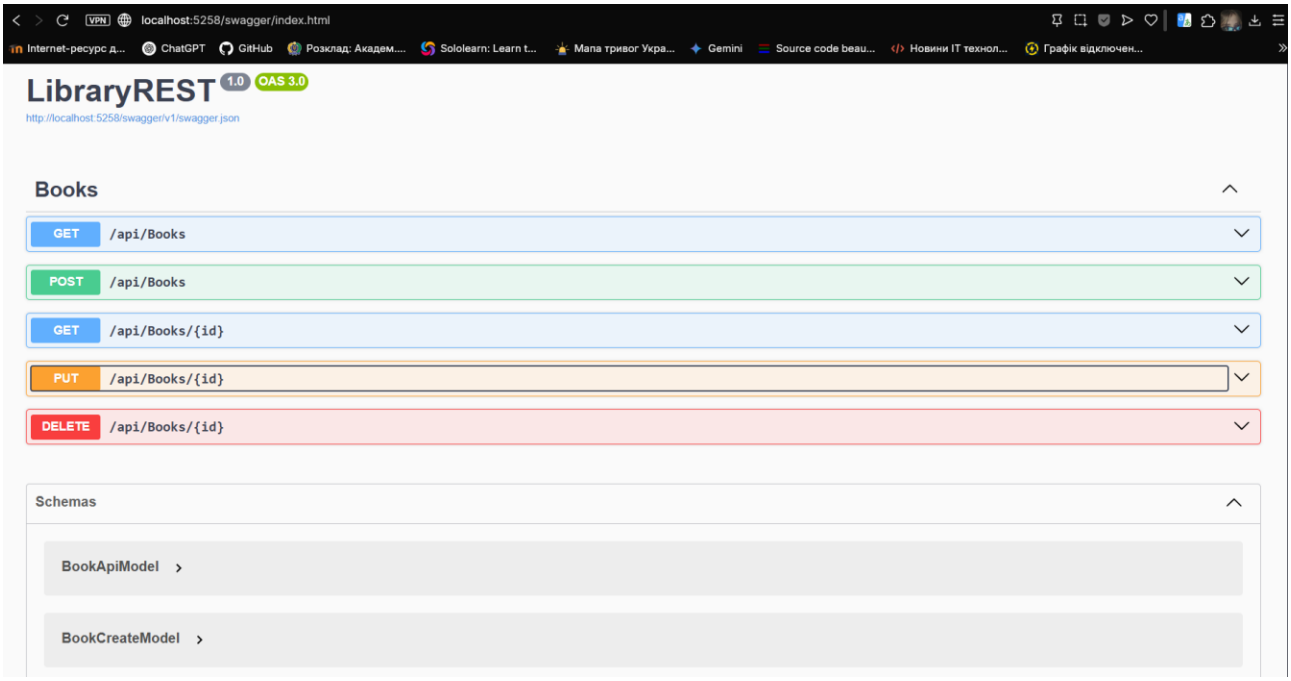
- 404 Not Found — коли книга не знайдена.
- 201 Created — коли створено нову книгу.
- 204 No Content — після оновлення або видалення без тіла відповіді.

Унікальні URI для ресурсів:

- api/books — колекція.
- api/books/{id} — конкретний ресурс.

Результат:





5. Використовуйте асинхрону версію дженерік CRUD сервісу, що використовує репозиторій для доступу до даних та який був реалізований у 3ій лабораторній роботі, у створених контролерах.:

```
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();
}
```

Імплементацію CRUD сервісу, репозиторіїв та контексту, додавайте у контролери використовуючи вбудовану у ASP.NET Core підтриму впровадження залежностей(Dependency Injection).

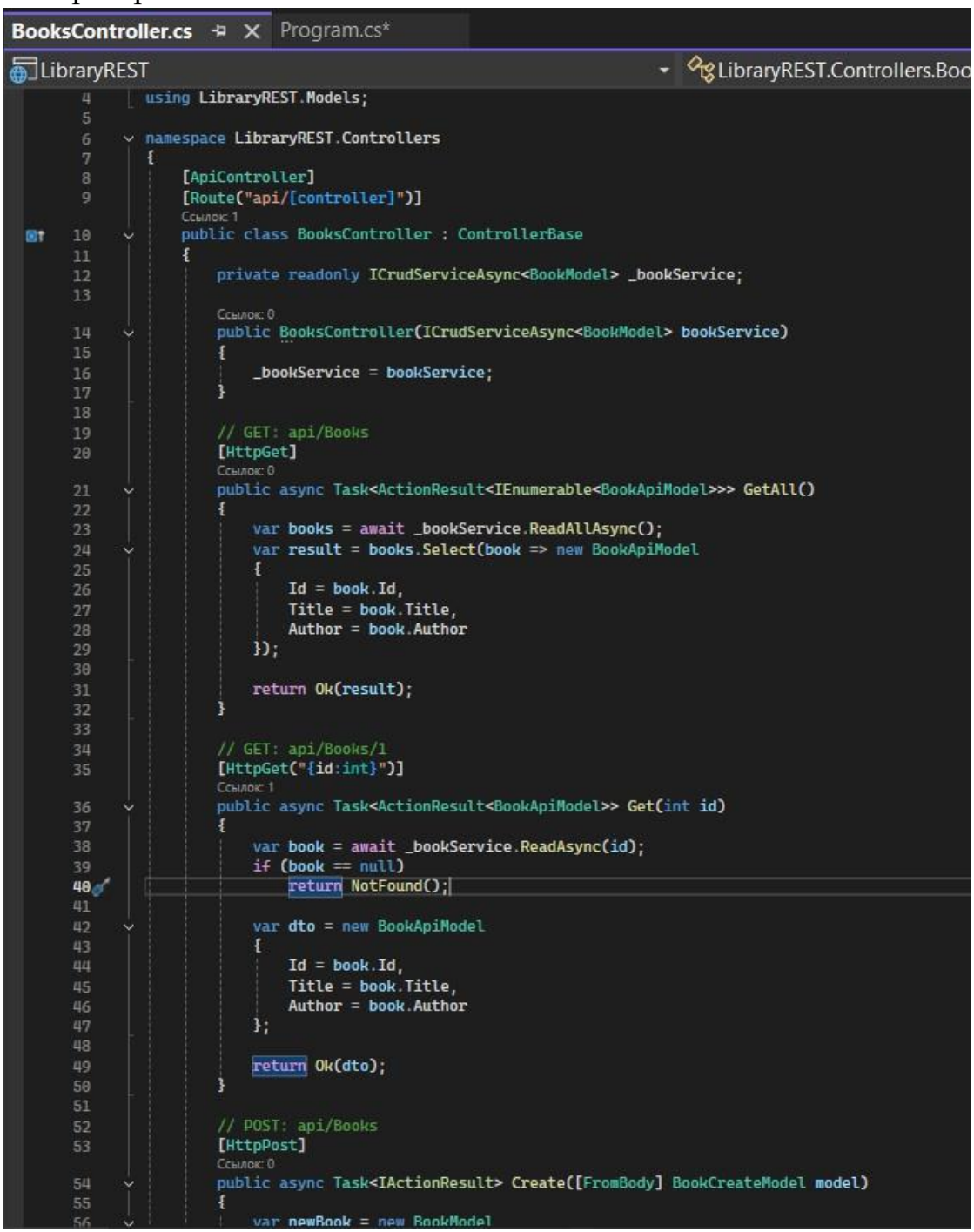
Інтерфейс:

CrudServiceAsync.cs | Program.cs*

Library.Infrastructure

```
using Library.Infrastructure.Repositories;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Library.Infrastructure.Services
{
    Ссылок: 3
    public class CrudServiceAsync<T> : ICrudServiceAsync<T> where T : class
    {
        private readonly IRepository<T> _repository;

        Ссылок: 0
        public CrudServiceAsync(IRepository<T> repository)
        {
            _repository = repository;
        }

        Ссылок: 2
        public async Task<bool> CreateAsync(T element)
        {
            await _repository.AddAsync(element);
            await _repository.SaveChangesAsync();
            return true;
        }

        Ссылок: 4
        public async Task<T?> ReadAsync(int id)
        {
            var all = await _repository.GetAllAsync();
            return all.FirstOrDefault(e => (e as dynamic).Id == id);
        }

        Ссылок: 2
        public async Task<IEnumerable<T>> ReadAllAsync()
        {
            return await _repository.GetAllAsync();
        }

        Ссылок: 1
        public async Task<IEnumerable<T>> ReadAllAsync(int page, int amount)
        {
            var all = await _repository.GetAllAsync();
            return all.Skip((page - 1) * amount).Take(amount);
        }

        Ссылок: 2
        public async Task<bool> UpdateAsync(T element)
        {
            await _repository.UpdateAsync(element);
            await _repository.SaveChangesAsync();
            return true;
        }

        Ссылок: 2
        public async Task<bool> RemoveAsync(T element)
        {
            await _repository.DeleteAsync(element);
            await _repository.SaveChangesAsync();
            return true;
        }

        Ссылок: 4
        public async Task<bool> SaveAsync()
        {
            await _repository.SaveChangesAsync();
            return true;
        }
    }
```

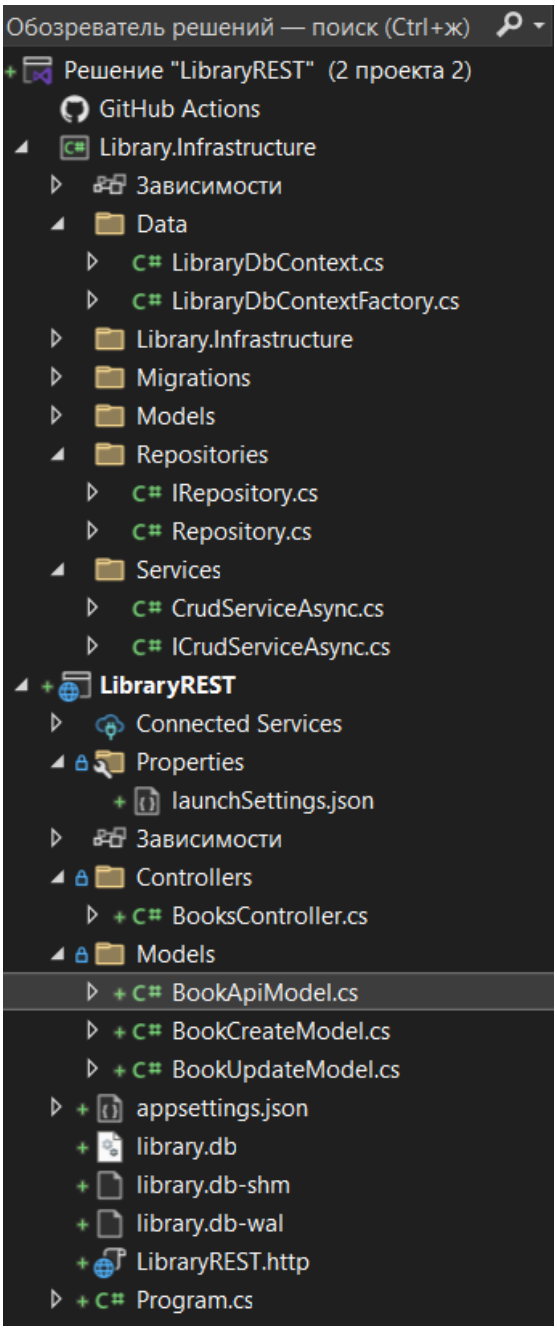
Контролер:


BooksController.cs
Program.cs*
LibraryREST
LibraryREST.Controllers.Boo
using LibraryREST.Models;
namespace LibraryREST.Controllers
{
[ApiController]
[Route("api/[controller]")]
Ссылок: 1
public class BooksController : ControllerBase
{
private readonly ICrudServiceAsync<BookModel> _bookService;
Ссылок: 0
public BooksController(ICrudServiceAsync<BookModel> bookService)
{
_bookService = bookService;
}
// GET: api/Books
[HttpGet]
Ссылок: 0
public async Task<ActionResult<IEnumerable<BookApiModel>>> GetAll()
{
var books = await _bookService.ReadAllAsync();
var result = books.Select(book => new BookApiModel
{
Id = book.Id,
Title = book.Title,
Author = book.Author
});
return Ok(result);
}
// GET: api/Books/1
[HttpGet("{id:int}")]
Ссылок: 1
public async Task<ActionResult<BookApiModel>> Get(int id)
{
var book = await _bookService.ReadAsync(id);
if (book == null)
return NotFound();
var dto = new BookApiModel
{
Id = book.Id,
Title = book.Title,
Author = book.Author
};
return Ok(dto);
}
// POST: api/Books
[HttpPost]
Ссылок: 0
public async Task<IActionResult> Create([FromBody] BookCreateModel model)
{
var newBook = new BookModel

BooksController.cs | Program.cs*

LibraryREST | LibraryREST.Controllers.BooksCont

```
52          // POST: api/Books
53          [HttpPost]
            Ссылок: 0
54          public async Task<IActionResult> Create([FromBody] BookCreateModel model)
55          {
56              var newBook = new BookModel
57              {
58                  Title = model.Title,
59                  Author = model.Author,
60                  LibraryMemberId = model.LibraryMemberId,
61                  BookTags = new List<BookTagModel>()
62              };
63
64              var created = await _bookService.CreateAsync(newBook);
65              if (!created) return BadRequest("Could not create book.");
66
67              await _bookService.SaveAsync();
68
69              return CreatedAtAction(nameof(Get), new { id = newBook.Id }, newBook);
70          }
71
72          // PUT: api/Books/1
73          [HttpPut("{id:int}")]
            Ссылок: 0
74          public async Task<IActionResult> Update(int id, [FromBody] BookUpdateModel model)
75          {
76              var book = await _bookService.ReadAsync(id);
77              if (book == null)
78                  return NotFound();
79
80              book.Title = model.Title;
81              book.Author = model.Author;
82              book.LibraryMemberId = model.LibraryMemberId;
83
84              var updated = await _bookService.UpdateAsync(book);
85              if (!updated) return BadRequest("Update failed.");
86
87              await _bookService.SaveAsync();
88
89              return NoContent();
90          }
91
92          // DELETE: api/Books/1
93          [HttpDelete("{id:int}")]
            Ссылок: 0
94          public async Task<IActionResult> Delete(int id)
95          {
96              var book = await _bookService.ReadAsync(id);
97              if (book == null)
98                  return NotFound();
99
100             var deleted = await _bookService.RemoveAsync(book);
101             if (!deleted) return BadRequest("Delete failed.");
102
103             await _bookService.SaveAsync();
104
105             return NoContent();
```



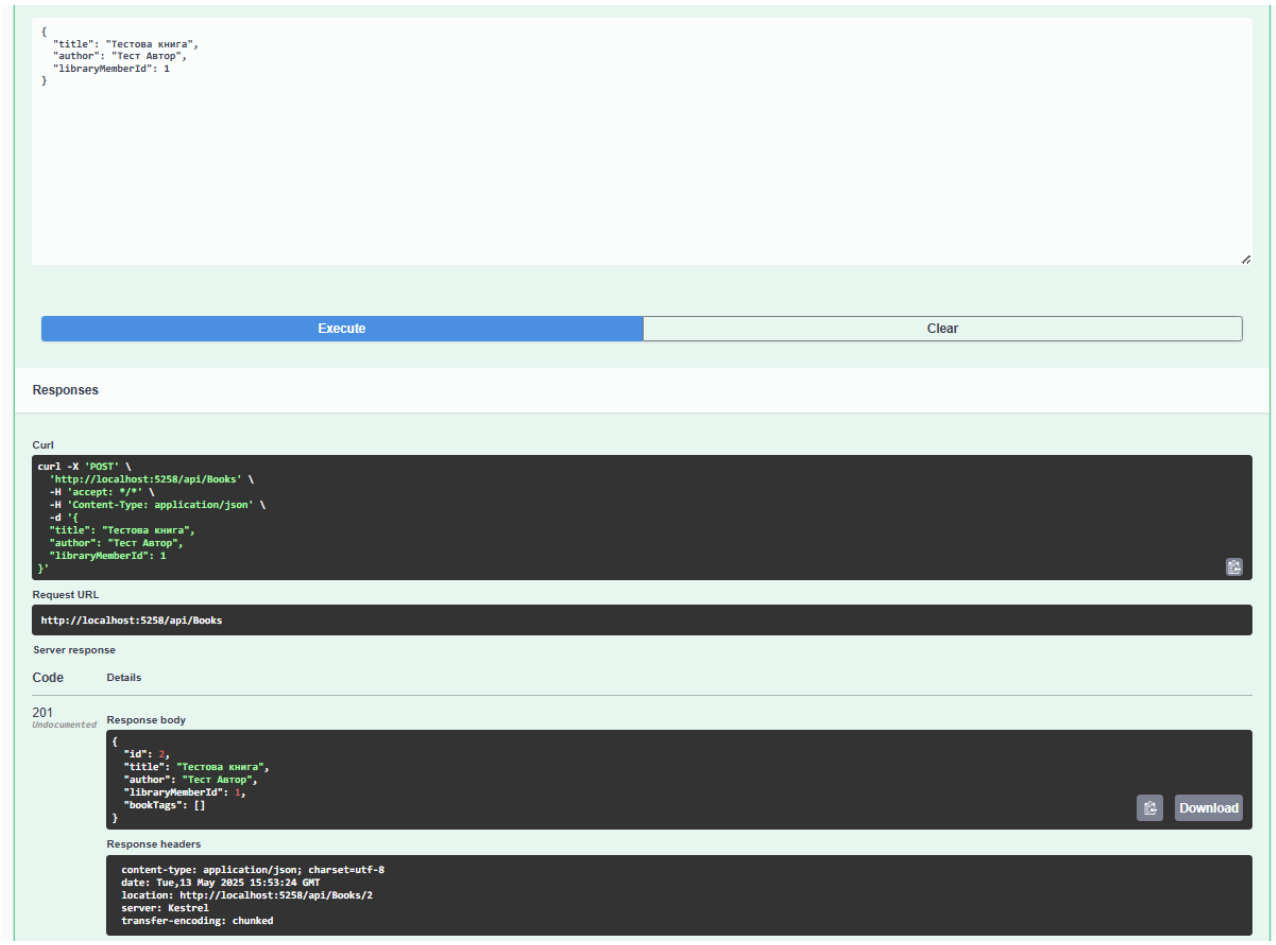
Код програми:

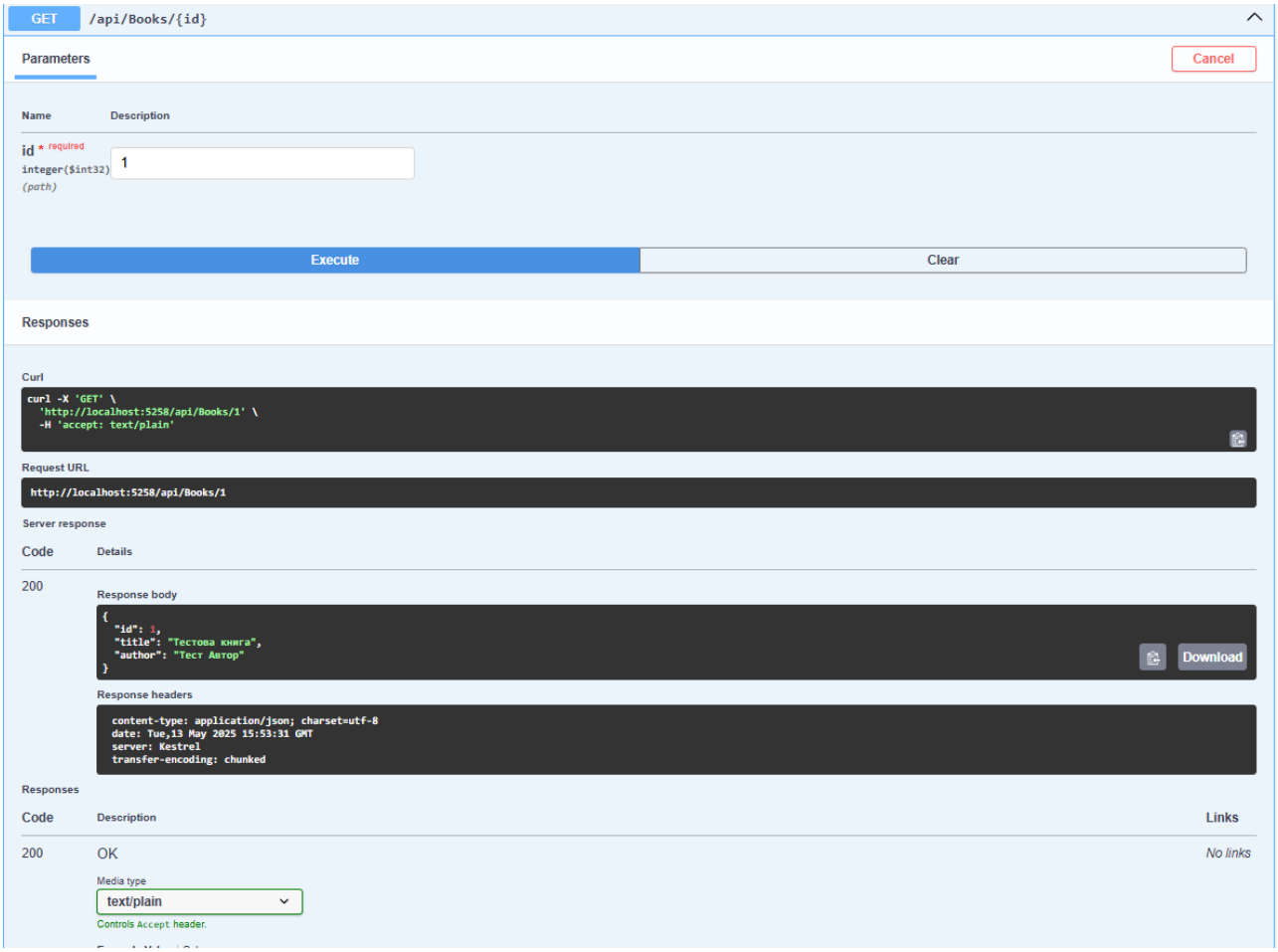
Program.cs*

LibraryREST

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

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<LibraryDbContext>(options =>
    options.UseSqlite("Data Source=library.db"));

builder.Services.AddScoped<IRepository<BookModel>, Repository<BookModel>>();
builder.Services.AddScoped<ICrudServiceAsync<BookModel>, CrudServiceAsync<BookModel>>();

builder.Services.AddScoped<IRepository<LibraryMemberModel>, Repository<LibraryMemberModel>>();
builder.Services.AddScoped<ICrudServiceAsync<LibraryMemberModel>, CrudServiceAsync<LibraryMemberModel>>();

builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

app.UseSwagger();
app.UseSwaggerUI();

app.UseHttpsRedirection();
app.UseAuthorization();

app.MapControllers();

using (var scope = app.Services.CreateScope())
{
    var db = scope.ServiceProvider.GetRequiredService<LibraryDbContext>();

    db.Database.Migrate();

    if (!db.LibraryMembers.Any())
    {
        var testMember = new LibraryMemberModel
        {
            FullName = "Тестовий Користувач",
            Email = "test@example.com"
        };
        db.LibraryMembers.Add(testMember);
        db.SaveChanges();

        var testBook = new BookModel
        {
            Title = "Тестова книга",
            Author = "Тест Автор",
            LibraryMemberId = testMember.Id,
            BookTags = new List<BookTagModel>()
        };
        db.Books.Add(testBook);
        db.SaveChanges();
    }
}

app.Run();
```

Результат Swagger:


{
"title": "Тестова книга",
"author": "Тест Автор",
"libraryMemberId": 1
}
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://localhost:5258/api/Books' \
-H 'accept: */*' \
-H 'Content-Type: application/json' \
-d '{
"title": "Тестова книга",
"author": "Тест Автор",
"libraryMemberId": 1
}'
Request URL
http://localhost:5258/api/Books
Server response
Code
Details
201
Undocumented
Response body
{
"id": 2,
"title": "Тестова книга",
"author": "Тест Автор",
"libraryMemberId": 1,
"bookTags": []
}
Download
Response headers
content-type: application/json; charset=utf-8
date: Tue,13 May 2025 15:53:24 GMT
location: http://localhost:5258/api/Books/2
server: Kestrel
transfer-encoding: chunked


GET
/api/Books/{id}
Parameters
Cancel
Name
Description
id * required
integer($int32)
(path)
1
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://localhost:5258/api/Books/1' \
-H 'accept: text/plain'
Request URL
http://localhost:5258/api/Books/1
Server response
Code
Details
200
Response body
{
"id": 1,
"title": "Тестова книга",
"author": "Тест Автор"
}
Download
Response headers
content-type: application/json; charset=utf-8
date: Tue,13 May 2025 15:53:31 GMT
server: Kestrel
transfer-encoding: chunked
Responses
Code
Description
Links
200
OK
No links
Media type
text/plain
Controls Accept header.

Додаткове завдання

Створіть проєкт {Назва тематики}.MVC за шаблоном Blazor Web App або ASP.NET MVC. У цьому проєкті реалізуйте вебзастосунок, що використовує контекст БД та моделі із 3 лабораторної, використовуючи генерацію сторінок вбудовану у ASP.NET.