WYŻSZA SZKOŁA
INFORMATYKI i ZARZĄDZANIA
z siedzibą w Rzeszowie

KOLEGIUM INFORMATYKI STOSOWANEJ

Kierunek: **INFORMATYKA**
Specjalność: **Technologie internetowe i mobilne**
Semestr: **6IIZ**
Grupa: **SP01**

Izabella Nosek
Nr albumu studenta: w71345

API „Biblioteka szkolna”

Prowadzący: inż. Eryk Chrustek

ZARZĄDZANIE DANYMI

Rzeszów 2025

Spis treści

1. Opis projektu

1.1. Założenia i cele projektu

Projekt ma na celu stworzenie API do obsługi systemu biblioteki szkolnej. API ma umożliwiać zarządzanie zasobami książkowymi oraz umożliwiać dodawanie danych dodatkowych np. egzemplarze książek.

Celem projektu jest:

- Utworzenie modelu bazy danych
- Stworzenie API do zarządzania danymi
- Prezentacja wizualna rozwiązania przy pomocy Swagger

1.2. Wykorzystane technologie

Projekt utworzono w technologii ASP.NET Core 9 API. Całość działa w środowisku lokalnym z lokalną bazą danych Microsoft SQL Server. Projekt rozwijany był w Visual Studio 2022.

Do obsługi połączenia z bazą zainstalowano package *Entity Framework Core*. Jest to uproszczona, rozszerzalna i wieloplatformowa wersja open source popularnej technologii dostępu do danych Entity Framework.

Do prezentacji działania aplikacji użyto narzędzia *Swagger* - interaktywnego narzędzia służącego do dokumentowania i testowania REST API. Automatycznie generuje czytelny interfejs użytkownika na podstawie kodu aplikacji, umożliwiając przeglądanie dostępnych endpointów, ich parametrów oraz wykonywanie zapytań bezpośrednio z poziomu przeglądarki.

W projekcie zaimplementowano trzy kontrolery: BookController, CatalogController oraz CopyController. Odpowiadają one kolejno za obsługę operacji CRUD na książkach, operacje GET wykorzystywane w widoku katalogu książek (dla czytelnika) oraz operacje CRUD na kopiach (fizycznych egzemplarzach) książek. Każdy z kontrolerów realizuje określone metody http.

2. Struktura bazy danych

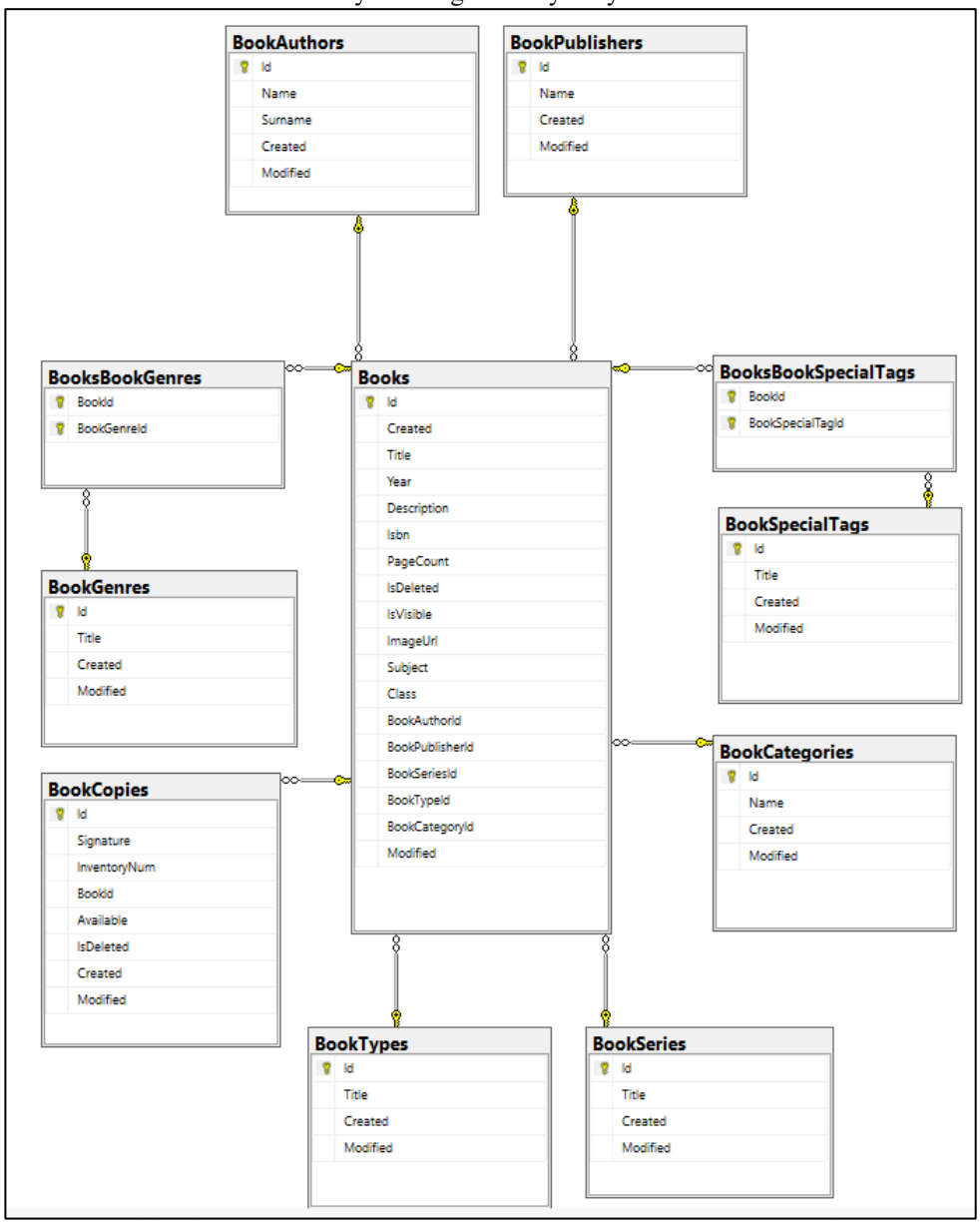
Baza danych zawiera tabele i relacje dotyczące zasobów bibliotecznych (książki, autorzy, egzemplarze, gatunki itd.). Struktura bazy dot. zasobów została utworzona programowo, korzystając z kodu C#.

Dostęp do danych opiera się o obiekty transferu danych DTO (Data Transfer Object) – dzięki temu użytkownik nie ma bezpośredniego dostępu do bazy oraz ma dostęp tylko do wybranej zawartości modelu.

2.1. Diagram bazy

Diagram został wygenerowany w *SQL Server Management Studio 20* i przedstawia tabele oraz ich relacje między sobą.

Rys. 1 Diagram bazy danych



Źródło 1: Praca własna

2.2. Encje i relacje

Encje i relacje między nimi zostały umieszczone w folderze *Models* projektu, razem z dodatkowymi klasami wspomagającymi (do obsługi paginacji, filtrów i sortowania wyników – więcej w punkcie 2.4). Wszystkie encje zawierają ID, datę utworzenia i datę modyfikacji.

Book – książka
BookAuthor – autor książki
BookCategory – kategoria książki
BookCopy – egzemplarz (fizyczna kopia) książki
BookGenre – gatunek książki
BookPublisher – wydawca książki
BookSeries – seria wydawnicza książki
BookSpecialTag – specjalne oznaczenie książki
BookType – typ książki

Zdj. 1 Encja Book

```
public class Book
{
    10 references
    public int Id { get; set; }
    0 references
    public DateTime Created { get; set; } = DateTime.Now;
    2 references
    public DateTime? Modified { get; set; }
    24 references
    public string Title { get; set; } = null!;
    13 references
    public int Year { get; set; }
    7 references
    public string Description { get; set; } = null!;//krótki opis ksiazki
    14 references
    public string Isbn { get; set; } = null!; //numer ISBN
    7 references
    public int PageCount { get; set; } //ilość stron
    8 references
    public bool IsDeleted { get; set; } = false;//do soft delete
    6 references
    public bool IsVisible { get; set; } = true;//do widoku w katalogu
    9 references
    public string? ImageUrl { get; set; } //okładka książki - URL do folderu
    4 references
    public string? Subject { get; set; } //przedmiot (dla podrecznikow)
    4 references
    public string? Class { get; set; } //klasa (dla lektur)

    #region Odwołania do innych
    3 references
    public int BookAuthorId { get; set; }
    [ForeignKey(nameof(BookAuthorId))]
    46 references
    public BookAuthor BookAuthor { get; set; } = null!;//autor

    3 references
    public int BookPublisherId { get; set; }
    [ForeignKey(nameof(BookPublisherId))]
    19 references
    public BookPublisher BookPublisher { get; set; } = null!;//wydawca

    3 references
    public int BookSeriesId { get; set; }
    [ForeignKey(nameof(BookSeriesId))]
    22 references
    public BookSeries BookSeries { get; set; } = null!;//seria np. Harry Potter

    3 references
    public int BookTypeId { get; set; }
    [ForeignKey(nameof(BookTypeId))]
    19 references
    public BookType BookType { get; set; } = null!;//typ ksiazki, rodzaj np. powieść

    3 references
    public int BookCategoryId { get; set; }
    [ForeignKey(nameof(BookCategoryId))]
    19 references
    public BookCategory BookCategory { get; set; } = null!;//jedno z trzech: lektura, podręcznik, pozostałe

    18 references
    public ICollection<BookBookGenre> BookBookGenres { get; set; } = new List<BookBookGenre>();//gatunek ksiazki np. fantasy - moze byc wiele

    16 references
    public ICollection<BookBookSpecialTag> BookBookSpecialTags { get; set; } = new List<BookBookSpecialTag>(); //special tag

    19 references
    public ICollection<BookCopy> BookCopies { get; set; } = new List<BookCopy>(); //lista kopii
    0 references
    public int CopyCount => BookCopies?.Count ?? 0; //liczba kopii
    0 references
    public int AvailableCopyCount => BookCopies?.Count(c => c.Available) ?? 0; //liczba dostępnych
    #endregion
}
```

Źródło 2 Praca własna

2.3. Obiekty transferu danych

Na ten moment projekt zawiera dwa rodzaje DTO – dotyczące książek i ich fizycznych kopii, z możliwością przyszłej rozbudowy. Każde DTO dzieli się na cztery rodzaje – Create, Edit, Get oraz GetDetailed i oferuje różne zestawy pobieranych danych.

BookCreateDTO - do tworzenia nowych obiektów typu *Book*.

BookEditDTO - do edycji obiektów typu *Book*.

BookGetDTO - do pobierania informacji z obiektów typu *Book*.

BookGetDetailedDTO - do pobierania szczegółowych informacji z obiektów typu *Book*.

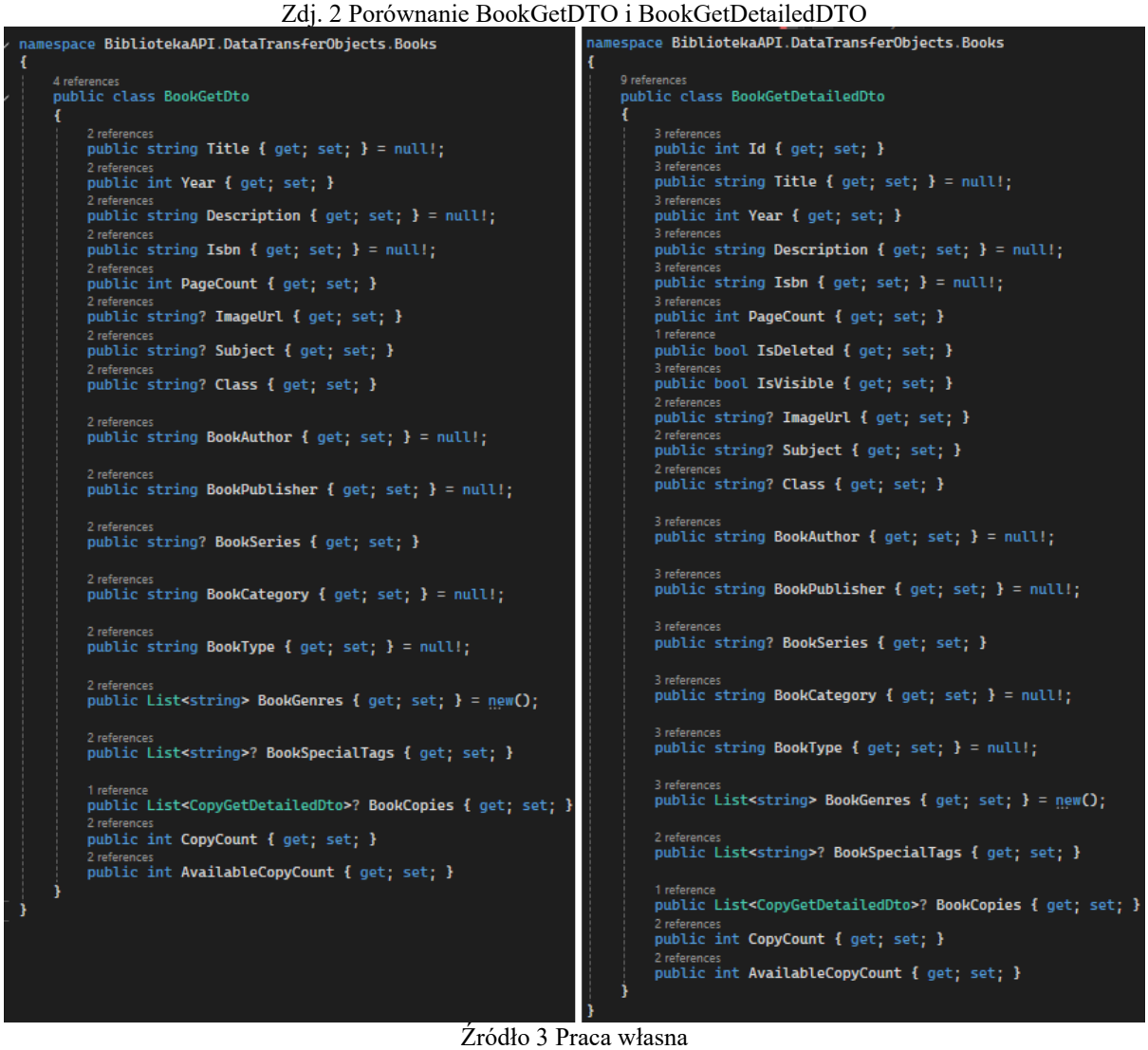
CopyCreateDTO - do tworzenia nowych obiektów typu *BookCopy*.

CopyEditDTO - do edycji obiektów typu *BookCopy*.

CopyGetDTO - do pobierania informacji z obiektów typu *BookCopy*.

CopyGetDetailedDTO - do pobierania szczegółowych informacji z obiektów typu *BookCopy*.

Zdj. 2 Porównanie BookGetDTO i BookGetDetailedDTO



```
namespace BibliotekaAPI.DataTransferObjects.Books
{
    4 references
    public class BookGetDto
    {
        2 references
        public string Title { get; set; } = null!;
        2 references
        public int Year { get; set; }
        2 references
        public string Description { get; set; } = null!;
        2 references
        public string Isbn { get; set; } = null!;
        2 references
        public int PageCount { get; set; }
        2 references
        public string? ImageUrl { get; set; }
        2 references
        public string? Subject { get; set; }
        2 references
        public string? Class { get; set; }

        2 references
        public string BookAuthor { get; set; } = null!;

        2 references
        public string BookPublisher { get; set; } = null!;

        2 references
        public string? BookSeries { get; set; }

        2 references
        public string BookCategory { get; set; } = null!;

        2 references
        public string BookType { get; set; } = null!;

        2 references
        public List<string> BookGenres { get; set; } = new();

        2 references
        public List<string>? BookSpecialTags { get; set; }

        1 reference
        public List<CopyGetDetailedDto>? BookCopies { get; set; }
        2 references
        public int CopyCount { get; set; }
        2 references
        public int AvailableCopyCount { get; set; }
    }
}
```

```
namespace BibliotekaAPI.DataTransferObjects.Books
{
    9 references
    public class BookGetDetailedDto
    {
        3 references
        public int Id { get; set; }
        3 references
        public string Title { get; set; } = null!;
        3 references
        public int Year { get; set; }
        3 references
        public string Description { get; set; } = null!;
        3 references
        public string Isbn { get; set; } = null!;
        3 references
        public int PageCount { get; set; }
        1 reference
        public bool IsDeleted { get; set; }
        3 references
        public bool IsVisible { get; set; }
        2 references
        public string? ImageUrl { get; set; }
        2 references
        public string? Subject { get; set; }
        2 references
        public string? Class { get; set; }

        3 references
        public string BookAuthor { get; set; } = null!;

        3 references
        public string BookPublisher { get; set; } = null!;

        3 references
        public string? BookSeries { get; set; }

        3 references
        public string BookCategory { get; set; } = null!;

        3 references
        public string BookType { get; set; } = null!;

        3 references
        public List<string> BookGenres { get; set; } = new();

        2 references
        public List<string>? BookSpecialTags { get; set; }

        1 reference
        public List<CopyGetDetailedDto>? BookCopies { get; set; }
        2 references
        public int CopyCount { get; set; }
        2 references
        public int AvailableCopyCount { get; set; }
    }
}
```

Źródło 3 Praca własna

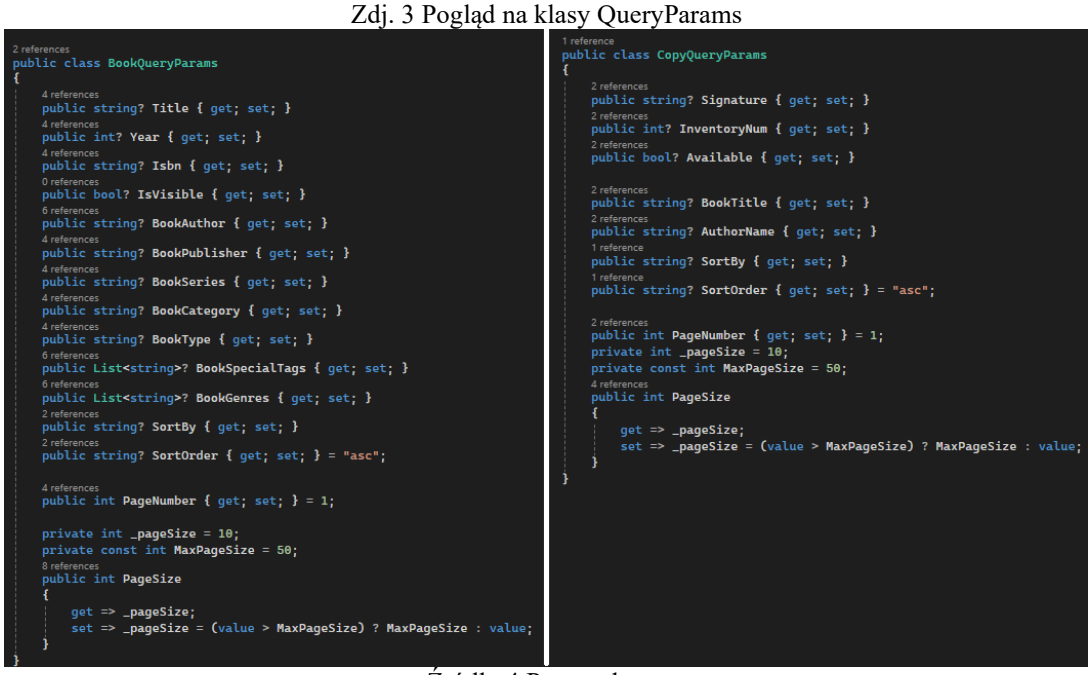
2.4. Paginacja, filtrowanie i sortowanie wyników

Do obsługi filtrów, sortowania i stronicowania wyników utworzono dodatkowe klasy wspomagające, zawierające wybrane parametry do filtrowania i sortowania wyników.

BookQueryParams – klasa zastosowana do obsługi zapytań dot. książek w kontrolerach *BookController* oraz *CatalogController* w metodach GET.

CopyQueryParams - klasa zastosowana do obsługi zapytań dot. egzemplarzy książek w kontrolerach *BookController* oraz *CatalogController* w metodach GET.

Zdj. 3 Pogląd na klasy QueryParams



```
2 references
public class BookQueryParams
{
    4 references
    public string? Title { get; set; }
    4 references
    public int? Year { get; set; }
    4 references
    public string? Isbn { get; set; }
    0 references
    public bool? IsVisible { get; set; }
    6 references
    public string? BookAuthor { get; set; }
    4 references
    public string? BookPublisher { get; set; }
    4 references
    public string? BookSeries { get; set; }
    4 references
    public string? BookCategory { get; set; }
    4 references
    public string? BookType { get; set; }
    6 references
    public List<string>? BookSpecialTags { get; set; }
    6 references
    public List<string>? BookGenres { get; set; }
    2 references
    public string? SortBy { get; set; }
    2 references
    public string? SortOrder { get; set; } = "asc";

    4 references
    public int PageNumber { get; set; } = 1;

    private int _pageSize = 10;
    private const int MaxPageSize = 50;
    8 references
    public int PageSize
    {
        get => _pageSize;
        set => _pageSize = (value > MaxPageSize) ? MaxPageSize : value;
    }
}
```

```
1 reference
public class CopyQueryParams
{
    2 references
    public string? Signature { get; set; }
    2 references
    public int? InventoryNum { get; set; }
    2 references
    public bool? Available { get; set; }

    2 references
    public string? BookTitle { get; set; }
    2 references
    public string? AuthorName { get; set; }
    1 reference
    public string? SortBy { get; set; }
    1 reference
    public string? SortOrder { get; set; } = "asc";

    2 references
    public int PageNumber { get; set; } = 1;
    private int _pageSize = 10;
    private const int MaxPageSize = 50;
    4 references
    public int PageSize
    {
        get => _pageSize;
        set => _pageSize = (value > MaxPageSize) ? MaxPageSize : value;
    }
}
```

Źródło 4 Praca własna

3. Kontrolery

CatalogController – kontroler do obsługi widoków ze strony użytkownika niezalogowanego i czytelnika. Oferuje operacje GET, wyświetlające podstawowe informacje o wybranych rekordach. Obsługuje filtry, sortowanie i paginację.

Zdj. 4 Pogląd na kontroler katalogu

```
[Route("api/catalog/[controller]")]
[ApiController]
0 references
public class CatalogController(ApplicationDbContext _context) : ControllerBase
{
    #region BOOKS
    [HttpGet("books")]
    0 references
    public async Task<ActionResult<BookGetDto>> GetBooks([FromQuery] BookQueryParams query)...

    [HttpGet("details/{id}")]
    0 references
    public async Task<ActionResult<BookGetDto>> GetBookById(int id)...

    #endregion

    #region COPIES
    [HttpGet("details/{bookId}/copies")]
    0 references
    public async Task<ActionResult<CopyGetDto>> GetCopiesForBook(int bookId)...

    [HttpGet("details/{bookId}/copies/{copyId}")]
    0 references
    public async Task<ActionResult<CopyGetDto>> GetCopyDetails(int bookId, int copyId)...
    #endregion
}
```

Źródło 5 Praca własna

BookController – kontroler do obsługi operacji na książkach takich jak pobieranie książek, pobieranie książek w koszu, pobieranie konkretnej książki, dodawanie nowej książki, edycja książki, miękkie usuwanie (przeniesienie do kosza) i usuwanie z bazy danych. Widok pełnej listy książek oferuje filtrowanie, sortowanie i stronicowanie wyników. Kontroler ten domyślnie ma być używany po stronie administracyjnej systemu, do operacji wykonywanych przez użytkowników z uprawnieniami administratora lub bibliotekarza.

Zdj. 5 Pogląd na kontroler Book

```
[Route("api/manage/[controller]")]
[ApiController]
0 references
public class BookController(ApplicationDbContext _context) : ControllerBase
{
    #region GET
    [HttpGet("books")]
    0 references
    public async Task<ActionResult<BookGetDetailedDto>> GetBooksDetailed([FromQuery] BookQueryParams query)...

    [HttpGet("bin")]
    0 references
    public async Task<ActionResult<BookGetDetailedDto>> GetBooksDeleted()...

    [HttpGet("details/{id}")]
    1 reference
    public async Task<ActionResult<BookGetDetailedDto>> GetBookByIdDetailed(int id)...
    #endregion

    #region POST
    [HttpPost("create")]
    0 references
    public async Task<ActionResult<BookGetDetailedDto>> CreateBook([FromBody] BookCreateDto dto)...
    #endregion

    #region PUT
    [HttpPut("edit/{id}")]
    0 references
    public async Task<ActionResult<BookGetDetailedDto>> EditBook(int id, [FromBody] BookEditDto dto)...

    // Soft delete - przeniesienie książki do "kosza"
    [HttpPut("bin/{id}")]
    0 references
    public async Task<ActionResult<BookGetDetailedDto>> BinBook(int id)...
    #endregion

    #region DELETE
    //hard delete - usuwa z bazy, ale tylko jeśli jest w koszu
    [HttpDelete("delete/{id}")]
    0 references
    public async Task<IActionResult> DeleteBook(int id)...
    #endregion
}
```

Źródło 6 Praca własna

Zdj. 6 Podgląd metody GET dla pobrania książek

```
[HttpGet("books")]
0 references
public async Task<ActionResult<BookGetDetailedDto>> GetBooksDetailed([FromQuery] BookQueryParams query)
{
    var booksQuery = _context.Books
        .Where(b => !b.IsDeleted)
        .Include(b => b.BookAuthor)
        .Include(b => b.BookPublisher)
        .Include(b => b.BookSeries)
        .Include(b => b.BookType)
        .Include(b => b.BookCategory)
        .Include(b => b.BookBookGenres).ThenInclude(bb => bb.BookGenre)
        .Include(b => b.BookBookSpecialTags).ThenInclude(bb => bb.BookSpecialTag)
        .Include(b => b.BookCopies)
        .AsQueryable();

    #region Filtry
    if (!string.IsNullOrWhiteSpace(query.Title))
        booksQuery = booksQuery.Where(b => b.Title != null && b.Title.Contains(query.Title));

    if (query.Year.HasValue)
        booksQuery = booksQuery.Where(b => b.Year == query.Year);

    if (!string.IsNullOrWhiteSpace(query.Isbn))
        booksQuery = booksQuery.Where(b => b.Isbn != null && b.Isbn.Contains(query.Isbn));

    if (!string.IsNullOrWhiteSpace(query.BookAuthor))
        booksQuery = booksQuery.Where...;

    if (!string.IsNullOrWhiteSpace(query.BookPublisher))
        booksQuery = booksQuery.Where...;

    if (!string.IsNullOrWhiteSpace(query.BookSeries))
        booksQuery = booksQuery.Where...;

    if (!string.IsNullOrWhiteSpace(query.BookCategory))
        booksQuery = booksQuery.Where...;

    if (!string.IsNullOrWhiteSpace(query.BookType))
        booksQuery = booksQuery.Where...;

    if (query.BookGenres != null && query.BookGenres.Count > 0)
        booksQuery = booksQuery.Where...;

    if (query.BookSpecialTags != null && query.BookSpecialTags.Count > 0)...
    #endregion

    #region Sortowanie
    bool descending = string.Equals(query.SortOrder, "desc", StringComparison.OrdinalIgnoreCase);

    booksQuery = (query.SortBy?.ToLower()) switch
    {
        "title" => descending ? booksQuery.OrderByDescending(b => b.Title) : booksQuery.OrderBy(b => b.Title),
        "year" => descending ? booksQuery.OrderByDescending(b => b.Year) : booksQuery.OrderBy(b => b.Year),
        "isbn" => descending ? booksQuery.OrderByDescending(b => b.Isbn) : booksQuery.OrderBy(b => b.Isbn),
        "author" => descending ? booksQuery.OrderByDescending(b => b.BookAuthor!.Surname) : booksQuery.OrderBy(b => b.BookAuthor!.Surname),
        "publisher" => descending ? booksQuery.OrderByDescending(b => b.BookPublisher!.Name) : booksQuery.OrderBy(b => b.BookPublisher!.Name),
        "series" => descending ? booksQuery.OrderByDescending(b => b.BookSeries!.Title) : booksQuery.OrderBy(b => b.BookSeries!.Title),
        "category" => descending ? booksQuery.OrderByDescending(b => b.BookCategory!.Name) : booksQuery.OrderBy(b => b.BookCategory!.Name),
        "type" => descending ? booksQuery.OrderByDescending(b => b.BookType!.Title) : booksQuery.OrderBy(b => b.BookType!.Title),
        _ => booksQuery.OrderBy(b => b.Title)
    };
    #endregion

    #region Wyniki z paginacją
    var totalCount = await booksQuery.CountAsync();
    var items = await booksQuery
        .Skip((query.PageNumber - 1) * query.PageSize)
        .Take(query.PageSize)
        .Select(b => new BookGetDetailedDto
        {
            Id = b.Id,
            Title = b.Title,
            Year = b.Year,
            Description = b.Description,
            Isbn = b.Isbn,
            PageCount = b.PageCount,
            IsDeleted = b.IsDeleted,
            IsVisible = b.IsVisible,
            ImageUrl = b.ImageUrl,
            Subject = b.Subject,
            Class = b.Class,
            BookAuthor = b.BookAuthor.Surname + ", " + b.BookAuthor.Name,
            BookPublisher = b.BookPublisher.Name,
            BookSeries = b.BookSeries != null ? b.BookSeries.Title : null,
            BookCategory = b.BookCategory.Name,
            BookType = b.BookType.Title,
            BookGenres = b.BookBookGenres.Select(bg => bg.BookGenre.Title).ToList(),
            BookSpecialTags = b.BookBookSpecialTags != null
               ? b.BookBookSpecialTags
                    .Where(bb => bb.BookSpecialTag != null)
                    .Select(bb => bb.BookSpecialTag!.Title)
                    .ToList()
               : new List<string>(),
           CopyCount = b.BookCopies.Count(),
            AvailableCopyCount = b.BookCopies.Count(c => c.Available)
        })
        .ToListAsync();
    #endregion

    return Ok(new
    {
        query.PageNumber,
        query.PageSize,
        TotalCount = totalCount,
        TotalPages = (int)Math.Ceiling(totalCount / (double)query.PageSize),
        Items = items
    });
}
```

Źródło 7 Praca własna

CopyController – kontroler do obsługi operacji na egzemplarzach książek: pobieranie egzemplarzy, pobieranie egzemplarzy w koszu, pobieranie konkretnego egzemplarza, dodawanie nowego egzemplarza, edycja egzemplarza, miękkie usuwanie (przeniesienie do kosza) i usuwanie z bazy danych. Widok pełnej listy książek oferuje filtrowanie, sortowanie i stronicowanie wyników. Kontroler ten domyślnie ma być używany po stronie administracyjnej systemu, do operacji wykonywanych przez użytkowników z uprawnieniami administratora lub bibliotekarza.

Zdj. 7 Pogląd na kontroler CopyBook

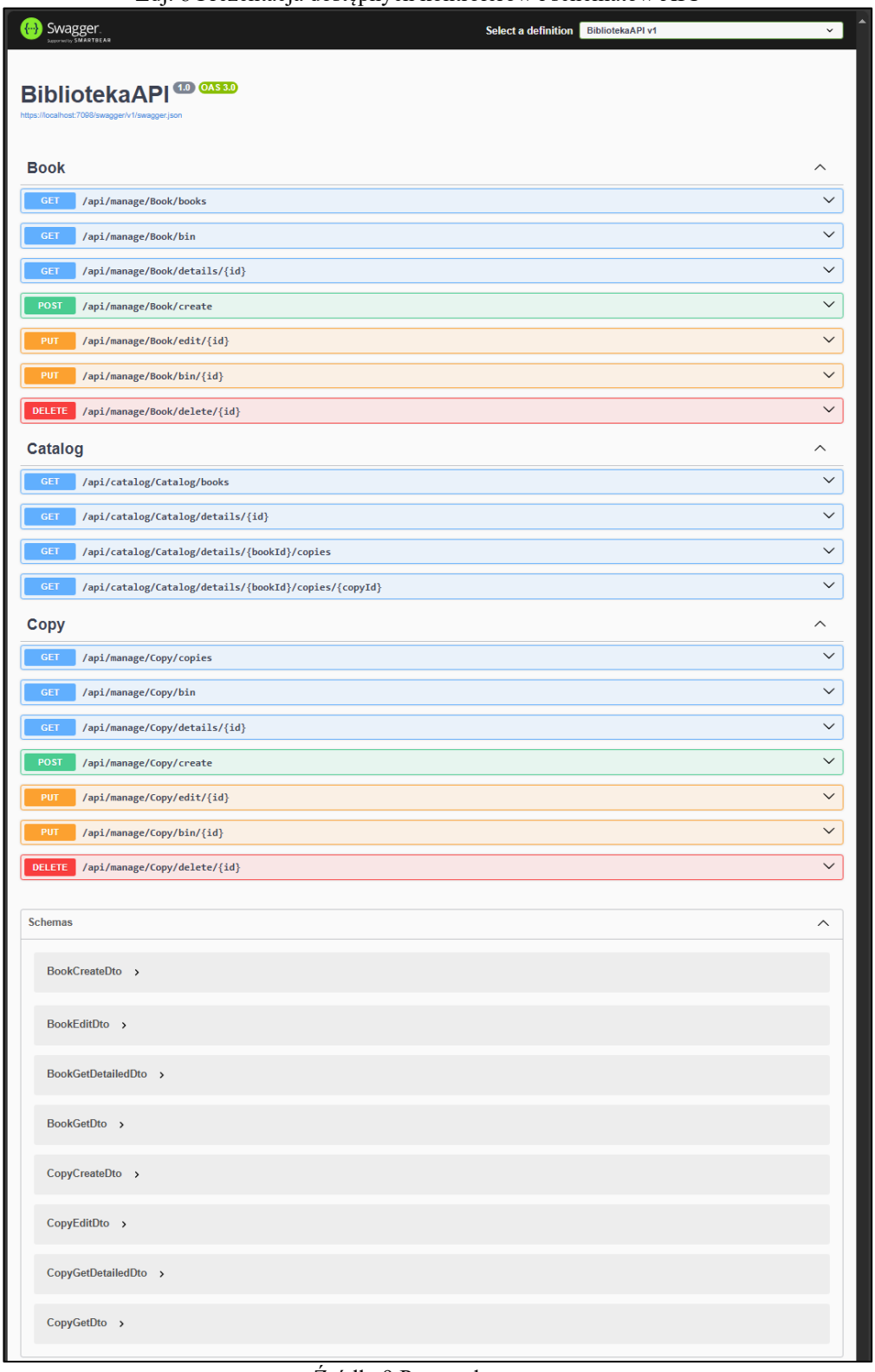
```
[Route("api/manage/[controller]")]
[ApiController]
0 references
public class CopyController(ApplicationDbContext _context) : ControllerBase
{
    #region GET
    [HttpGet("copies")]
    0 references
    public async Task<ActionResult<IEnumerable<CopyGetDetailedDto>>> GetCopiesDetailed([FromQuery] CopyQueryParams query)...

    [HttpGet("bin")]
    0 references
    public async Task<ActionResult<IEnumerable<CopyGetDetailedDto>>> GetDeletedCopies()...

    [HttpGet("details/{id}")]
    1 reference
    public async Task<ActionResult<CopyGetDetailedDto>> GetCopyByIdDetailed(int id)...
    #endregion

    #region POST
    [HttpPost("create")]
    0 references
    public async Task<ActionResult<CopyGetDetailedDto>> CreateCopy([FromBody] CopyCreateDto dto)...
    #endregion

    #region PUT
    [HttpPut("edit/{id}")]
    0 references
    public async Task<ActionResult<CopyGetDetailedDto>> EditCopy(int id, [FromBody] CopyEditDto dto)...

    // Soft delete - przeniesienie egzemplarza do "kosza"
    [HttpPut("bin/{id}")]
    0 references
    public async Task<ActionResult<CopyGetDetailedDto>> SoftDeleteCopy(int id)...
    #endregion

    #region DELETE
    //hard delete - usuwa z bazy, ale tylko jeśli jest w koszu
    [HttpDelete("delete/{id}")]
    0 references
    public async Task<IActionResult> DeleteCopy(int id)...
    #endregion
}
```

Źródło 8 Praca własna

4. Prezentacja aplikacji za pomocą Swagger

Działanie API można zaprezentować za pomocą Swaggera, który oferuje prosty interfejs do obrazowania działania i przesyłanych danych. Poniżej przykłady wybranych metod GET.

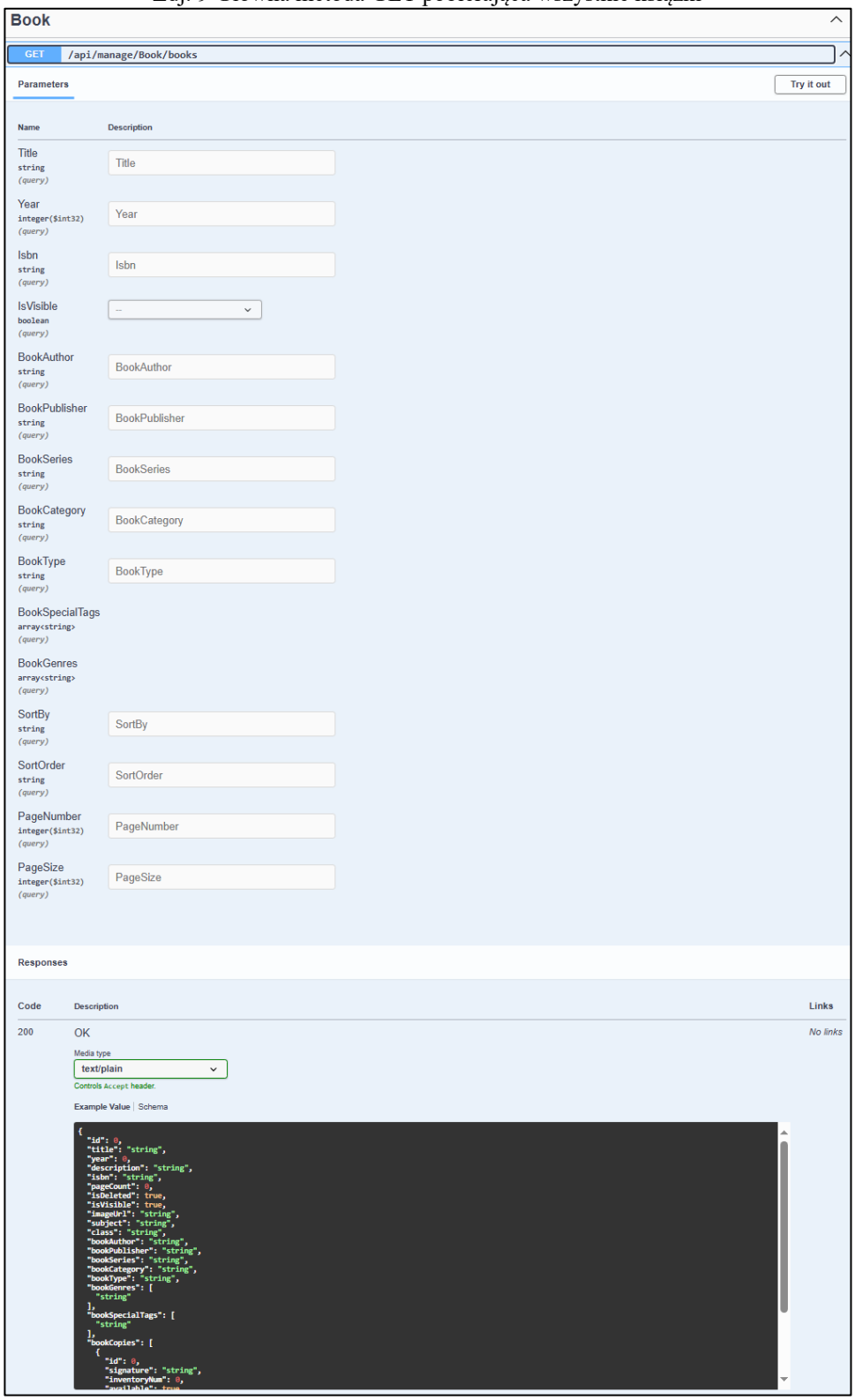
Zdj. 8 Prezentacja dostępnych kontrolerów i schematów API



Źródło 9 Praca własna

Metoda GET dla wszystkich elementów *Books* – pobiera wszystkie obiekty *Book* oraz oferuje filtry i sortowanie. Zwraca po kilka wyników na stronę (domyślnie 10). Poniżej pogląd na metodę oraz przykładowe wyniki z filtrem i bez.

Zdj. 9 Główna metoda GET pobierająca wszystkie książki



Źródło 10 Praca własna

Zdj. 10 Przykładowe wyniki bez filtrowania

Curl

```
curl -X 'GET' \
  'https://localhost:7098/api/manage/Book/books' \
  -H 'accept: text/plain'
```

Request URL

```
https://localhost:7098/api/manage/Book/books
```

Server response

Code | Details

200

Response body


```
{
  "pageNumber": 1,
  "pageSize": 10,
  "totalCount": 63,
  "totalPages": 7,
  "items": [
    {
      "id": 49,
      "title": "Akademia Pana Kleksa",
      "year": 2023,
      "description": "Powieść baśniowa o naukach chłopca Adasia Niezgódki w dziwnej, trochę śmiesznej i zwariowanej akademii prowadzonej przez sympatycznego d
ziwaka Pana Kleksa. Ilustracje Jana Marcina Szancera.",
      "isbn": "978-83-7272-432-8",
      "pageCount": 120,
      "isDeleted": false,
      "isVisible": true,
      "imageUrl": "/images/books/10013759-9026-4886-b0a7-306b82bcef12..jpg",
      "class": "IV-VI",
      "bookAuthor": "Brzechwa, Jan",
      "bookPublisher": "G&P",
      "bookSeries": "brak",
      "bookCategory": "lektura",
      "bookType": "Powieść",
      "bookGenres": [
        "Fantastyka",
        "Młodzieżowa",
        "Dziecięca",
        "Science fiction"
```


Download

Response headers

```
content-type: application/json; charset=utf-8
date: Tue,01 Jul 2025 10:32:39 GMT
server: Kestrel
```

Źródło 11 Praca własna

Zdj. 11 Przykładowe wyniki z filtrem "Rowling" w *BookAuthor*

Curl

```
curl -X 'GET' \
  'https://localhost:7098/api/manage/Book/books?BookAuthor=Rowling' \
  -H 'accept: text/plain'
```

Request URL

```
https://localhost:7098/api/manage/Book/books?BookAuthor=Rowling
```

Server response

Code | Details

200

Response body


```
{
  "pageNumber": 1,
  "pageSize": 10,
  "totalCount": 7,
  "totalPages": 1,
  "items": [
    {
      "id": 4,
      "title": "Harry Potter i Czara Ognia",
      "year": 2001,
      "description": "W Hogwarcie rozgrywa sie Turniej Trójmagiczny, na który przybywaja uczniowie szkół z Francji i Bulgarii. Zgodnie ze stara tradycja, do t
urnieju maja byc wybrani trzej przedstawiciele, kazdej ze szkół. Jednak dziwnym zbiegiem okolicznosci do turnieju zostaje wybranych czterech uczestników, w ty
m i Harry Potter.",
      "isbn": "978-83-7278-021-8",
      "pageCount": 766,
      "isDeleted": false,
      "isVisible": true,
      "imageUrl": "/images/books/94cb3604-46aa-4fa4-a979-12bddf3e87cd..jpeg",
      "bookAuthor": "Rowling, Joanne K.",
      "bookPublisher": "Media Rodzina",
      "bookSeries": "Harry Potter",
      "bookCategory": "pozostałe",
      "bookType": "Powieść",
      "bookGenres": [
        "Fantastyka",
        "Przygodowa",
        "Młodzieżowa"
      ],
```


Download

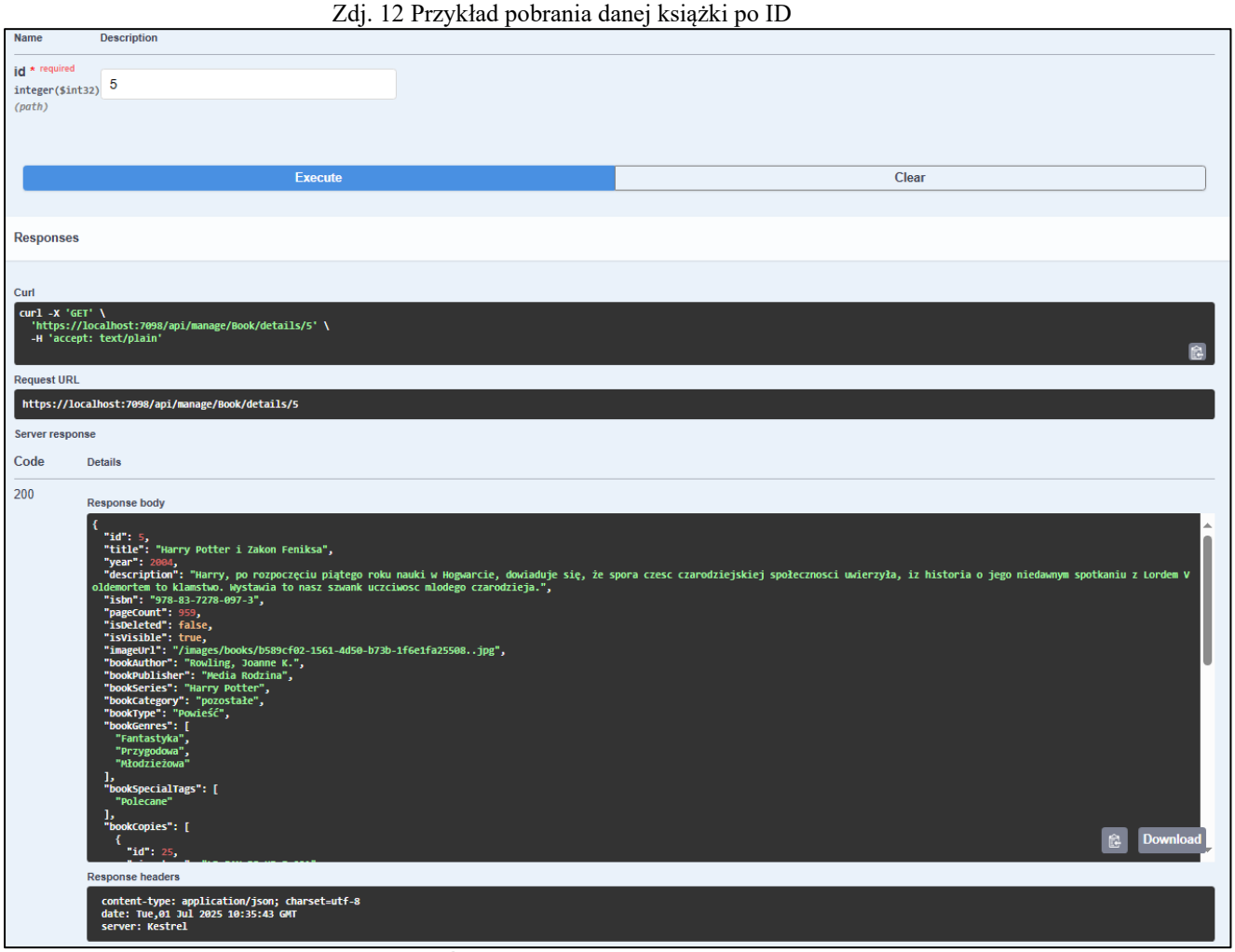
Response headers

```
content-type: application/json; charset=utf-8
date: Tue,01 Jul 2025 10:27:58 GMT
server: Kestrel
```

Źródło 12 Praca własna

Metoda GET pobierająca wybraną książkę po ID – zwraca wybraną książkę z listą egzemplarzy i innymi informacjami.

Zdj. 12 Przykład pobrania danej książki po ID



Źródło 13 Praca własna

5. Podsumowanie projektu

Repozytorium

Projekt wyeksportowano za pomocą Git. Kod aplikacji oraz baza danych znajdują się pod linkiem: https://github.com/iguanaiza/BibliotekaAPI

Podsumowanie

Udało się stworzyć podstawę pod rozbudowany system zarządzania zasobami biblioteki oraz ich udostępnianiem. Projekt został zaprojektowany w sposób umożliwiający jego dalszy rozwój.

W przyszłości aplikacja może zostać rozszerzona m.in. o sortowanie wyników i zaawansowane filtrowanie danych dla kolejnych tabel, rozbudowany system zarządzania użytkownikami, integrację z dodatkowymi usługami zewnętrznymi (np. API Biblioteki Narodowej) i warstwę frontend.

6. Netografia

1. *Entity Framework Core* (2024). Microsoft. https://learn.microsoft.com/pl-pl/ef/core
2. *What's new in C# 13* (2025). Microsoft. https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-13
3. *Tutorial: Create a controller-based web API with ASP.NET Core* (2025). Microsoft. https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-9.0&tabs=visual-studio
4. *What's new in ASP.NET Core in .NET 9* (2024). Microsoft. https://learn.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-9.0?view=aspnetcore-9.0
5. *Install SQL Server Management Studio* (2025). Microsoft. https://learn.microsoft.com/en-us/ssms/install/install
6. *Swagger* https://swagger.io/