WYŻSZA SZKOŁA
INFORMATYKI i ZARZĄDZANIA
z siedzibą w Rzeszowie

Temat projektu: Aplikacja serwisu bibliotecznego

Autorzy: Krystian Labak w68962, Maksymilian Lechowicz w67652

Rzeszów 2025

Rzeszów 2025

Spis treści

Rzeszów 2025

1. Opis wybranego stosu technologicznego

Aplikacja wykorzystuje następujące technologie i biblioteki:

Język programowania:

- .NET w wersji 8.0

Frameworki i biblioteki:

- ASP Net Core API
- Microsoft.EntityFrameworkCore w wersji 8.0.17
- Microsoft.EntityFrameworkCore.Design w wersji 8.0.17
- Microsoft.EntityFrameworkCore.SqlServer w wersji 8.0.17
- Microsoft.EntityFrameworkCore.Tools w wersji 8.0.17
- Swachbuckle.AspNetCore w wersji 6.4.0
- Newtonsoft.Json 13.0.3
- Xunit 2.5.3
- Xunit.runner.visualstudio 2.5.3
- Microsoft.NET.Test.Sdk 17.8.0
- coverlet.collector 6.0.0

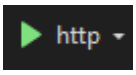
Narzędzia developerskie:

- Visual Studio 2022
- Swagger UI
- Microsoft Sql Server Management (SSMS)

2. Instrukcja uruchamiania aplikacji

Do uruchomienia aplikacji należy:

- Pobrać plik .zip przesłany w zadaniu projektowym na platformie Moodle
- W folderze BibliotekaAplikacjaProjekt uruchomić plik BibliotekaAplikacjaProjekt.sln
- Należy uruchomić migracje, w celu zaktualizowania schematu bazy danych poleceniami:
    - Add-migration Projekt
    - Update-database
- Uruchomić projekt w visual studio za pomocą http



Rys 1 Przycisk do uruchomienia projektu

Rzeszów 2025

3. Opis kluczowych elementów back-endu

3.1 Mikro serwis Wypożyczeń i Książek

Mikro serwis ten zajmuje się zarządzaniem informacji o książkach oraz zajmuje się procesem wypożyczenia książek przez użytkowników. Najważniejszymi elementami tego mikro serwisu są:

3.1.1 Encje

Encje reprezentują modele danych w systemie i są mapowane na tabele bazodanowe

```
namespace BibliotekaAplikacjaProjekt.Entities
{
    [Table("Book", Schema = "Bookstore")]
    6 references
    public class Book : BaseEntity
    {
        [Required]
        3 references
        public Guid Id { get; set; } = Guid.NewGuid();
        [Required]
        2 references
        public string BookTitle { get; set; }
        2 references
        public string Author {  get; set; }
        [Required]
        2 references
        public bool IsActive { get; set; }
    }
}
```

Rys 2 Encja Book

```
namespace BibliotekaAplikacjaProjekt.Entities
{
    [Table("Book", Schema = "Bookstore")]
    4 references
    public class Order : BaseEntity
    {
        [Required]
        3 references
        public Guid Id { get; set; }
        2 references
        public Guid BuyerId { get; set; }
        2 references
        public int Discount { get; set; }
        0 references
        public ICollection<Book>? Books { get; set; }
    }
}
```

Rys 3 Encja Order

3.1.2 Połączenie bazodanowe DbContext

Klasa BookstoreDbContext pełni funkcję połączenia z bazą danych oraz konfiguracji dla encji Book oraz Order

```
namespace BibliotekaAplikacjaProjekt
{
    9 references
    public class BookstoreDbContext : DbContext
    {


        1 reference
        private IConfiguration _configuration { get; }
        0 references
        public DbSet<Book> Books { get; set; }
        0 references
        public DbSet<Order> Orders { get; set; }

        0 references
        public BookstoreDbContext(IConfiguration configuration)
            : base()
        {
            _configuration = configuration;
        }
        0 references
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"server=(localdb)\MSSQLLocalDB;database=bibliotekproj_szk3;trusted_connection=true",
                x => x.MigrationsHistoryTable("__EFMigrationsHistory", "projektbookshop"));
        }
        0 references
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
        }
    }
}
```

Rys 4 DbContext klasy BookstoreDbContext

3.1.3 Warstwa serwisów BibliotekaAplikacjaProjekt.Services

Warstwa te pełni role logiki funkcjonalności projektu, by każda z wykonanych akcji poprawnie działała

```
4 references
public class BookService : CrudServiceBase<BookstoreDbContext, Book, BookDto>
{
    private BookstoreDbContext _bookstoreDbContext;
    0 references
    public BookService(BookstoreDbContext bookstoreDbContext) : base(bookstoreDbContext)
    {
        _bookstoreDbContext = bookstoreDbContext;
    }
    2 references
    public async Task<BookDto> GetById(Guid id)
    {
        var book = await base.GetById(id);
        return book.ToDto();
    }
    1 reference
    public async Task<IEnumerable<BookDto>> Get()
    {
        var book = await base.Get();
        return book.Select(e => e.ToDto());
    }
    1 reference
    public async Task<CrudOperationResult<BookDto>> Create(BookDto dto)
    {
        var entity = dto.ToEntity();
        var entityId = await base.Create(entity);
        var newDto = await GetById(entity.Id);
        return new CrudOperationResult<BookDto>
        {
            Result = newDto,
            Status = CrudOperationResultStatus.Success
        };
    }
    0 references
    public async Task<CrudOperationResult<BookDto>> Delete(Guid id)
    {
        return await base.Delete(id);
    }
    0 references
    public async Task<CrudOperationResult<BookDto>> Update(BookDto dto)
    {
        var entity = dto.ToEntity();
        return await base.Update(entity);
    }
}
```

Rys 5 Serwis dla BookService

```
3 references
public class OrderService : CrudServiceBase<BookstoreDbContext, Order, OrderDto>
{
    private BookstoreDbContext _bookstoreDbContext;
    0 references
    public OrderService(BookstoreDbContext bookstoreDbContext) : base(bookstoreDbContext)
    {
        _bookstoreDbContext = bookstoreDbContext;
    }
    2 references
    public async Task<OrderDto> GetById(Guid id)
    {
        var order = await base.GetById(id);
        return order.ToDto();
    }
    1 reference
    public async Task<IEnumerable<OrderDto>> Get()
    {
        var order = await base.Get();
        return order.Select(e => e.ToDto());
    }
    1 reference
    public async Task<CrudOperationResult<OrderDto>> Create(OrderDto dto)
    {
        var entity = dto.ToEntity();
        var entityId = await base.Create(entity);
        var newDto = await GetById(entity.Id);
        return new CrudOperationResult<OrderDto>
        {
            Result = newDto,
            Status = CrudOperationResultStatus.Success
        };
    }
    0 references
    public async Task<CrudOperationResult<OrderDto>> Delete(Guid id)
    {
        return await base.Delete(id);
    }
    0 references
    public async Task<CrudOperationResult<OrderDto>> Update(OrderDto dto)
    {
        var entity = dto.ToEntity();
        return await base.Update(entity);
    }
}
```

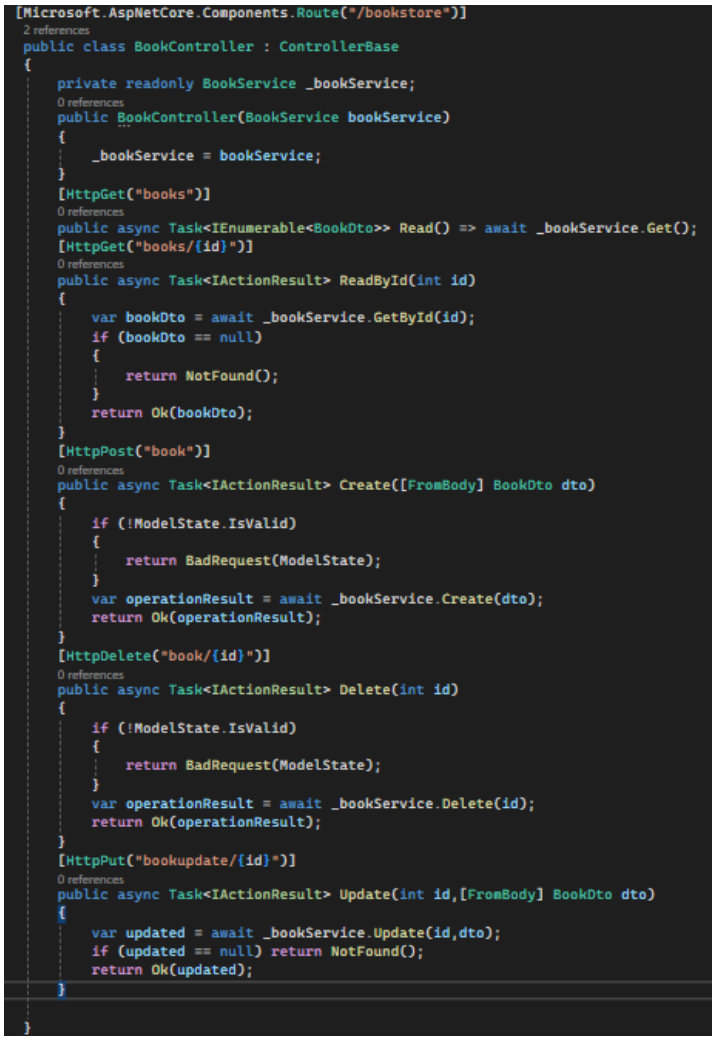
Rys 6 Serwis dla OrderService

3.1.4 Kontrolery serwisu Wypożyczeń i Książek

Kontrolery są odpowiedzialne za działalność backendu

```
[Microsoft.AspNetCore.Components.Route("/bookstore")]
2 references
public class OrderController : ControllerBase
{
    private readonly OrderService _orderService;
    0 references
    public OrderController(OrderService orderService)
    {
        _orderService = orderService;
    }
    [HttpGet("Orders")]
    0 references
    public async Task<IEnumerable<OrderDto>> Read() => await _orderService.Get();
    [HttpGet("Orders/{id}")]
    0 references
    public async Task<IActionResult> ReadById(int id)
    {
        var orderDto = await _orderService.GetById(id);
        if (orderDto == null)
        {
            return NotFound();
        }
        return Ok(orderDto);
    }
    [HttpPost("Order")]
    0 references
    public async Task<IActionResult> Create([FromBody] OrderDto dto)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }
        var operationResult = await _orderService.Create(dto);
        return Ok(operationResult);
    }
    [HttpDelete("order/{id}")]
    0 references
    public async Task<IActionResult> Delete(int id)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }
        var operationResult = await _orderService.Delete(id);
        return Ok(operationResult);
    }
}
```

Rys 7 Kontroler OrderController



```
[Microsoft.AspNetCore.Components.Route("/bookstore")]
2 references
public class BookController : ControllerBase
{
    private readonly BookService _bookService;
    0 references
    public BookController(BookService bookService)
    {
        _bookService = bookService;
    }
    [HttpGet("books")]
    0 references
    public async Task<IEnumerable<BookDto>> Read() => await _bookService.Get();
    [HttpGet("books/{id}")]
    0 references
    public async Task<IActionResult> ReadById(int id)
    {
        var bookDto = await _bookService.GetById(id);
        if (bookDto == null)
        {
            return NotFound();
        }
        return Ok(bookDto);
    }
    [HttpPost("book")]
    0 references
    public async Task<IActionResult> Create([FromBody] BookDto dto)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }
        var operationResult = await _bookService.Create(dto);
        return Ok(operationResult);
    }
    [HttpDelete("book/{id}")]
    0 references
    public async Task<IActionResult> Delete(int id)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }
        var operationResult = await _bookService.Delete(id);
        return Ok(operationResult);
    }
    [HttpPut("bookupdate/{id}")]
    0 references
    public async Task<IActionResult> Update(int id,[FromBody] BookDto dto)
    {
        var updated = await _bookService.Update(id,dto);
        if (updated == null) return NotFound();
        return Ok(updated);
    }

}
```

Rys 8 Kontroler BookController

3.2 Mikro serwis Czytelnik

Mikro serwis Czytelnik w projekcie pełni role zarządzania kontami użytkowników. Może dodawać, odczytywać oraz usuwać użytkowników. Najważniejsze elementy dla tego serwisu są:

3.2.1 Encja

Serwis Czytelnik posiada jedną encję, która pełni role użytkownika biblioteki.

```
namespace CzytelnikAplikacjaProjekt.Entities
{
    [Table("Reader", Schema = "ReaderDb")]

    public class Reader : BaseEntity
    {
        [Required]

        public Guid Id { get; set; } = Guid.NewGuid();
        [Required]
        [MaxLength(100)]
        2 references
        public string Name { get; set; }
        [Required]
        [MaxLength(100)]
        2 references
        public string Surname { get; set; }
        [Required]
        [EmailAddress]
        2 references
        public string Email { get; set; }
    }
}
```

Rys 9 Encja Reader

3.2.2 Połączenie bazodanowe DbContext

Klasa ReaderDbContext pełni role połączenia bazodanowego, z którego później serwis ma dostęp do listy użytkowników, może dodawać nowych użytkowników lub ich usuwać.

```
namespace CzytelnikAplikacjaProjekt
{
    7 references
    public class ReaderDbContext : DbContext
    {
        1 reference
        private IConfiguration _configuration { get; }
        5 references
        public DbSet<Entities.Reader> Readers { get; set; }

        0 references
        public ReaderDbContext(IConfiguration configuration)
            : base()
        {
            _configuration = configuration;
        }
        0 references
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"server=(localdb)\MSSQLLocalDB;database=bibliotekproj_szk3;trusted_connection=true",
                x => x.MigrationsHistoryTable("__EFMigrationsHistory", "projektbookshop"));
        }
        0 references
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
        }
    }
}
```

Rys 10 DbContext klasy ReaderDbContext

3.2.3 Warstwa serwisów CzytelnikAplikacjaProjekt.Services

Serwis ten kontroluje logiką mikro serwisu, dzięki niemu jest możliwość odczytywania listy użytkowników, dodania lub usunięcia użytkownika.

```
4 references
public class ReaderService
{
    private readonly ReaderDbContext _context;

    0 references
    public ReaderService(ReaderDbContext context)
    {
        _context = context;
    }

    1 reference
    public IEnumerable<Entities.Reader> GetAll() => _context.Readers.ToList();

    2 references
    public Entities.Reader GetById(Guid id)
    {
        return _context.Readers.FirstOrDefault(r => r.Id == id);
    }

    1 reference
    public void Add(Entities.Reader reader)
    {
        _context.Readers.Add(reader);
        _context.SaveChanges();
    }

    1 reference
    public void Delete(Guid id)
    {
        var reader = _context.Readers.Find(id);
        if (reader != null)
        {
            _context.Readers.Remove(reader);
            _context.SaveChanges();
        }
    }
}
```

Rys 11 Serwis dla ReaderService

3.2.4 Kontrolery serwisu Czytelnik

Są one odpowiedzialne za funkcjonowanie backendu serwisu Czytelnik

```
[Microsoft.AspNetCore.Components.Route("/bookstore")]
2 references
public class ReaderController : ControllerBase
{
    private readonly ReaderService _readerService;
    private readonly OrderResolver _orderResolver;
    private readonly CouponResolver _couponResolver;
    0 references
    public ReaderController(ReaderService readerService, OrderResolver orderResolver, CouponResolver couponResolver)
    {
        _readerService = readerService;
        _orderResolver = orderResolver;
        _couponResolver = couponResolver;
    }
    [HttpGet("readers")]
    0 references
    public async Task<IEnumerable<ReaderDto>> Read() => await _readerService.Get();
    [HttpGet("readers/{id}")]
    0 references
    public async Task<IActionResult> ReadById(int id)
    {
        var readerDto = await _readerService.GetById(id);
        if (readerDto == null)
        {
            return NotFound();
        }
        var orders = await _orderResolver.GetOrdersByReaderId(id);
        Task<List<CouponDto>> coupons = _couponResolver.GetCouponsByDiscount(1);
        foreach (OrderDto i in orders) {
            coupons = _couponResolver.GetCouponsByDiscount(i.Discount);
            break;
                }
        return Ok(new
        {
            Id = readerDto.Id,
            Name = readerDto.Name,
            Surname = readerDto.Surname,
            Email = readerDto.Email,
            Orders = orders,
            Coupons = coupons
        }) ;

    }
```

Rys 12 Kontroler ReaderController 1

```
[HttpPost("reader")]
0 references
public async Task<IActionResult> Create([FromBody] ReaderDto dto)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    var operationResult = await _readerService.Create(dto);
    return Ok(operationResult);
}
[HttpDelete("reader/{id}")]
0 references
public async Task<IActionResult> Delete(int id)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    var operationResult = await _readerService.Delete(id);
    return Ok(operationResult);
}
```

Rys 13 Kontroler ReaderController 2

3.2.5 Warstwa Resolver

Klasy w folderze Resolver pełnią funkcje komunikacji między innymi serwisami

```
4 references
public class CouponResolver
{
    private readonly HttpClient _httpClient;

    0 references
    public CouponResolver(HttpClient httpClient)
    {
        _httpClient = httpClient;
        _httpClient.BaseAddress = new Uri("http://localhost:5075/");
        _httpClient.DefaultRequestHeaders.Accept.Clear();
        _httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));
    }

    2 references
    public async Task<List<CouponDto>> GetCouponsByDiscount(int readerId)
    {
        var response = await _httpClient.GetAsync($"CouponByOrder/{readerId}");
        if (!response.IsSuccessStatusCode) return new();

        var json = await response.Content.ReadAsStringAsync();
        return JsonConvert.DeserializeObject<List<CouponDto>>(json) ?? new();
    }
}
```

Rys 14 Resolver dla serwisu z Promocjami i Programami Lojalnościowymi

```
4 references
public class OrderResolver
{

    private readonly HttpClient _httpClient;

    0 references
    public OrderResolver(HttpClient httpClient)
    {
        _httpClient = httpClient;
        _httpClient.BaseAddress = new Uri("http://localhost:5003/");
        _httpClient.DefaultRequestHeaders.Accept.Clear();
        _httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));
    }

    1 reference
    public async Task<List<OrderDto>> GetOrdersByReaderId(int readerId)
    {
        var response = await _httpClient.GetAsync($"OrdersBuy/{readerId}");
        if (!response.IsSuccessStatusCode) return new();

        var json = await response.Content.ReadAsStringAsync();
        return JsonConvert.DeserializeObject<List<OrderDto>>(json) ?? new();
    }
}
```

Rys 15 Resolver dla serwisu z Wypożeczeniami i Książkami

3.3 Mikro serwis Promocji i Programów lojalnościowych

3.3.1 Encje

Serwis promocji i programów lojalnościowych posiada 2 encje która składuje promocje oraz punkty lojalnościowe dla aktywnych użytkowników

```
[Table("Coupon", Schema = "Bookstore")]
5 references
public class Coupon : BaseEntity
{
    [Required]
    4 references
    public int Id { get; set; }
    2 references
    public int MultiplierDiscount { get; set; }
    2 references
    public DateTime ExpirationDate { get; set; }
}
```

Rys 16 Encja Coupon

```
[Table("Point", Schema = "Bookstore")]

public class Point : BaseEntity
{
    [Required]

    public int Id { get; set; }
    [Required]

    public int ReaderId { get; set; }
    [Required]
    [MaxLength(100)]

    public int Amount { get; set; }
}
```

Rys 17 Encja Point

3.3.2 Połączenie bazodanowe DbContext

Klasa CouponDbContext pełni role połączenia bazodanowego, z którego serwis jest w stanie pobierać promocje oraz punktu lojalnościowe

```
11 references
public class CouponDbContext : DbContext
{


    1 reference
    private IConfiguration _configuration { get; }
    0 references
    public DbSet<Coupon> Coupons { get; set; }
    0 references
    public DbSet<Point> Points { get; set; }
    0 references
    public CouponDbContext(IConfiguration configuration)
        : base()
    {
        _configuration = configuration;
    }
    0 references
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(@"server=(localdb)\MSSQLLocalDB;database=bibliotekproj_szk3;trusted_connection=true",
            x => x.MigrationsHistoryTable("__EFMigrationsHistory", "projektbookshop"));
    }
    0 references
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
    }
}
```

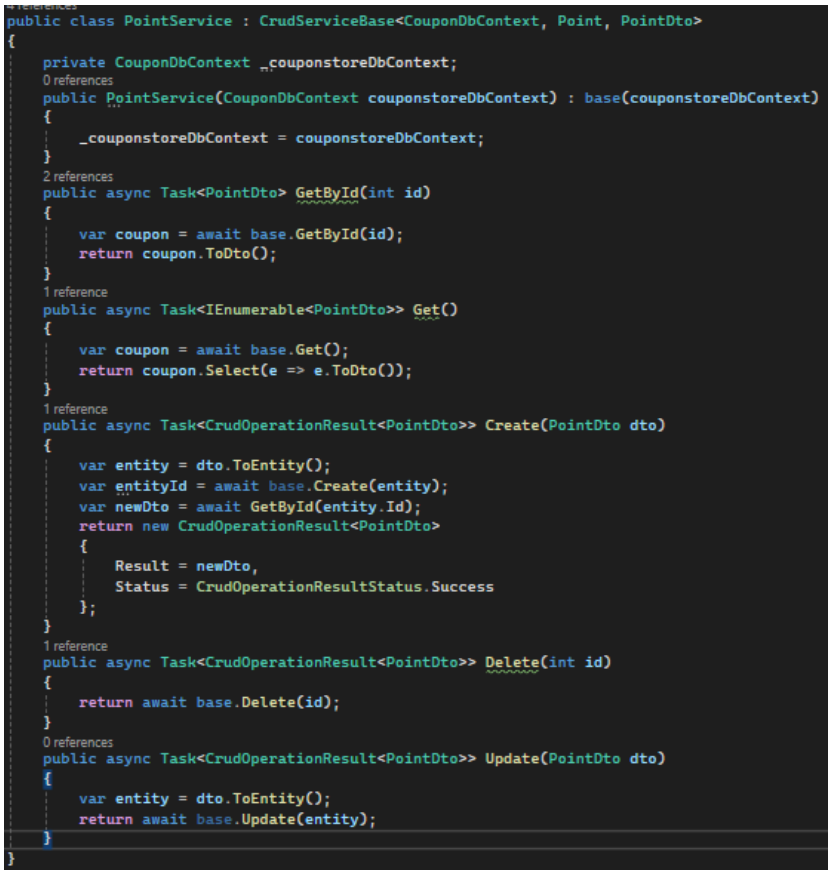
Rys 18 DbContext klasy CouponDbContext

3.3.3 Warstwa serwisów dla mikro serwisu Promocje i Programy lojalnościowe

Serwis ten posiada dwa serwisy które kontrolują logiką przypisywania promocji i punktów lojalnościowych dla aktywnych użytkowników biblioteki

```
public class CouponService : CrudServiceBase<CouponDbContext, Coupon, CouponDto>
{
    private CouponDbContext _couponstoreDbContext;
    0 references
    public CouponService(CouponDbContext couponstoreDbContext) : base(couponstoreDbContext)
    {
        _couponstoreDbContext = couponstoreDbContext;
    }
    2 references
    public async Task<CouponDto> GetById(int id)
    {
        var coupon = await base.GetById(id);
        return coupon.ToDto();
    }
    1 reference
    public async Task<IEnumerable<CouponDto>> Get()
    {
        var coupon = await base.Get();
        return coupon.Select(e => e.ToDto());
    }
    1 reference
    public async Task<IEnumerable<CouponDto>> GetDiscount(int buyerid)
    {
        var order = await base.Get();
        return order.Where(e => e.Id == buyerid).Select(x => x.ToDto());
    }
    1 reference
    public async Task<CrudOperationResult<CouponDto>> Create(CouponDto dto)
    {
        var entity = dto.ToEntity();
        var entityId = await base.Create(entity);
        var newDto = await GetById(entity.Id);
        return new CrudOperationResult<CouponDto>
        {
            Result = newDto,
            Status = CrudOperationResultStatus.Success
        };
    }
    1 reference
    public async Task<CrudOperationResult<CouponDto>> Delete(int id)
    {
        return await base.Delete(id);
    }
    0 references
    public async Task<CrudOperationResult<CouponDto>> Update(CouponDto dto)
    {
        var entity = dto.ToEntity();
        return await base.Update(entity);
    }
}
```

Rys 19 Serwis CouponService



```
public class PointService : CrudServiceBase<CouponDbContext, Point, PointDto>
{
    private CouponDbContext _couponstoreDbContext;
    0 references
    public PointService(CouponDbContext couponstoreDbContext) : base(couponstoreDbContext)
    {
        _couponstoreDbContext = couponstoreDbContext;
    }
    2 references
    public async Task<PointDto> GetById(int id)
    {
        var coupon = await base.GetById(id);
        return coupon.ToDto();
    }
    1 reference
    public async Task<IEnumerable<PointDto>> Get()
    {
        var coupon = await base.Get();
        return coupon.Select(e => e.ToDto());
    }
    1 reference
    public async Task<CrudOperationResult<PointDto>> Create(PointDto dto)
    {
        var entity = dto.ToEntity();
        var entityId = await base.Create(entity);
        var newDto = await GetById(entity.Id);
        return new CrudOperationResult<PointDto>
        {
            Result = newDto,
            Status = CrudOperationResultStatus.Success
        };
    }
    1 reference
    public async Task<CrudOperationResult<PointDto>> Delete(int id)
    {
        return await base.Delete(id);
    }
    0 references
    public async Task<CrudOperationResult<PointDto>> Update(PointDto dto)
    {
        var entity = dto.ToEntity();
        return await base.Update(entity);
    }
}
```

Rys 20 Serwis PointService

3.3.4 Warstwa kontrolerów

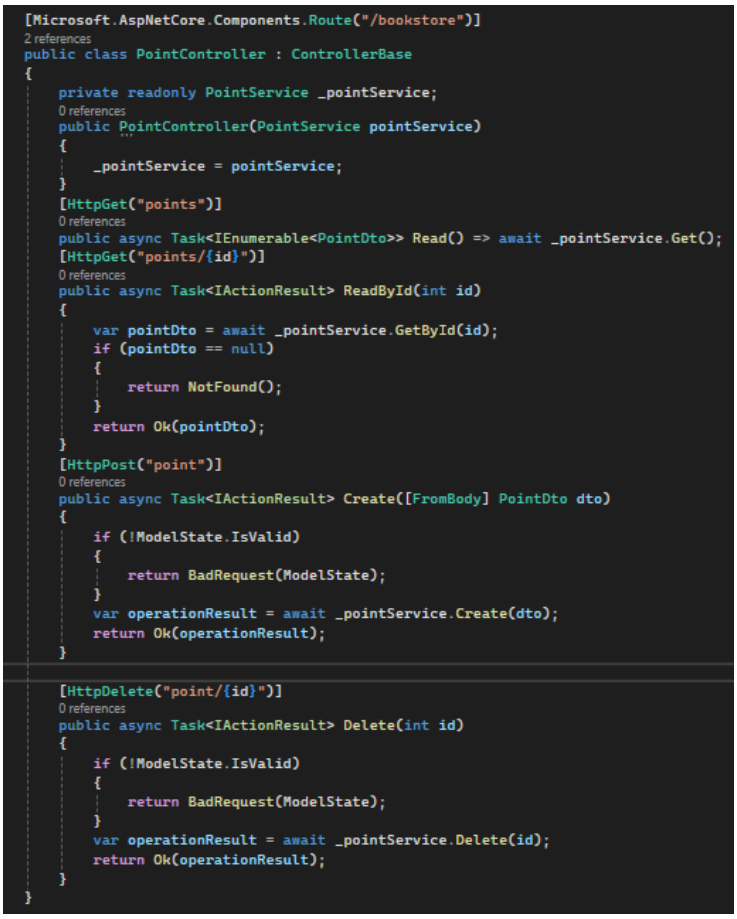
Serwis ten posiada 2 kontrolery, które zapewniają funkcjonalność backendu dla tego serwisu.

```
[Microsoft.AspNetCore.Components.Route("/bookstore")]
0 references
public class CouponController : ControllerBase
{
    private readonly CouponService _couponService;
    0 references
    public CouponController(CouponService couponService)
    {
        _couponService = couponService;
    }
    [HttpGet("coupons")]
    0 references
    public async Task<IEnumerable<CouponDto>> Read() => await _couponService.Get();
    [HttpGet("coupons/{id}")]
    0 references
    public async Task<IActionResult> ReadById(int id)
    {
        var couponDto = await _couponService.GetById(id);
        if (couponDto == null)
        {
            return NotFound();
        }
        return Ok(couponDto);
    }
    [HttpGet("CouponByOrder/{id}")]
    0 references
    public async Task<IActionResult> ReadByDiscount(int id)
    {
        var orderDto = await _couponService.GetDiscount(id);
        if (orderDto == null)
        {
            return NotFound();
        }
        return Ok(orderDto);
    }
    [HttpPost("coupon")]
    0 references
    public async Task<IActionResult> Create([FromBody] CouponDto dto)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }
        var operationResult = await _couponService.Create(dto);
        return Ok(operationResult);
    }
```

Rys 21 Kontroler CouponController 1

```
[HttpDelete("coupon/{id}")]
0 references
public async Task<IActionResult> Delete(int id)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    var operationResult = await _couponService.Delete(id);
    return Ok(operationResult);
}
```

Rys 22 Kontroler CouponController 2



```
[Microsoft.AspNetCore.Components.Route("/bookstore")]
2 references
public class PointController : ControllerBase
{
    private readonly PointService _pointService;
    0 references
    public PointController(PointService pointService)
    {
        _pointService = pointService;
    }
    [HttpGet("points")]
    0 references
    public async Task<IEnumerable<PointDto>> Read() => await _pointService.Get();
    [HttpGet("points/{id}")]
    0 references
    public async Task<IActionResult> ReadById(int id)
    {
        var pointDto = await _pointService.GetById(id);
        if (pointDto == null)
        {
            return NotFound();
        }
        return Ok(pointDto);
    }
    [HttpPost("point")]
    0 references
    public async Task<IActionResult> Create([FromBody] PointDto dto)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }
        var operationResult = await _pointService.Create(dto);
        return Ok(operationResult);
    }

    [HttpDelete("point/{id}")]
    0 references
    public async Task<IActionResult> Delete(int id)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }
        var operationResult = await _pointService.Delete(id);
        return Ok(operationResult);
    }
}
```

Rys 23 Kontroler PointController

3.4 Projekt testów jednostkowych

Projekt testów ma na celu przetestowanie funkcjonalności między serwisami.

3.4.1 Klasa ReaderServiceTest

Klasa ta trzyma w sobie logikę testów wykonywanych w celu sprawdzenia czy funkcje testowane współpracują miedzy sobą.

```
public class ReaderServiceTests
{
    2 references
    private ReaderService GetServiceWithDb(out ReaderDbContext context)
    {
        var options = new DbContextOptionsBuilder<ReaderDbContext>()
            .UseInMemoryDatabase($"ReaderDb_{Guid.NewGuid()}")
            .Options;

        var config = new ConfigurationBuilder().Build();
        context = new ReaderDbContext(config);
        typeof(ReaderDbContext).GetProperty("Options", System.Reflection.BindingFlags.NonPublic | System.Reflection.BindingFlags.Instance)
            ?.SetValue(context, options);

        context.Database.EnsureCreated();

        return new ReaderService(context);

    }

    [Fact]
    0 references
    public void AddReader_ShouldAddReader()
    {
        var service = GetServiceWithDb(out var context);
        var reader = new Reader
        {
            Name = "Jan",
            Surname = "Kowalski",
            Email = "jan.kowalski@example.com"
        };

        service.Create(reader);
        var added = context.Readers.FirstOrDefault(r => r.Email == "jan.kowalski@example.com");

        Assert.NotNull(added);
        Assert.Equal("Jan", added.Name);
    }
```

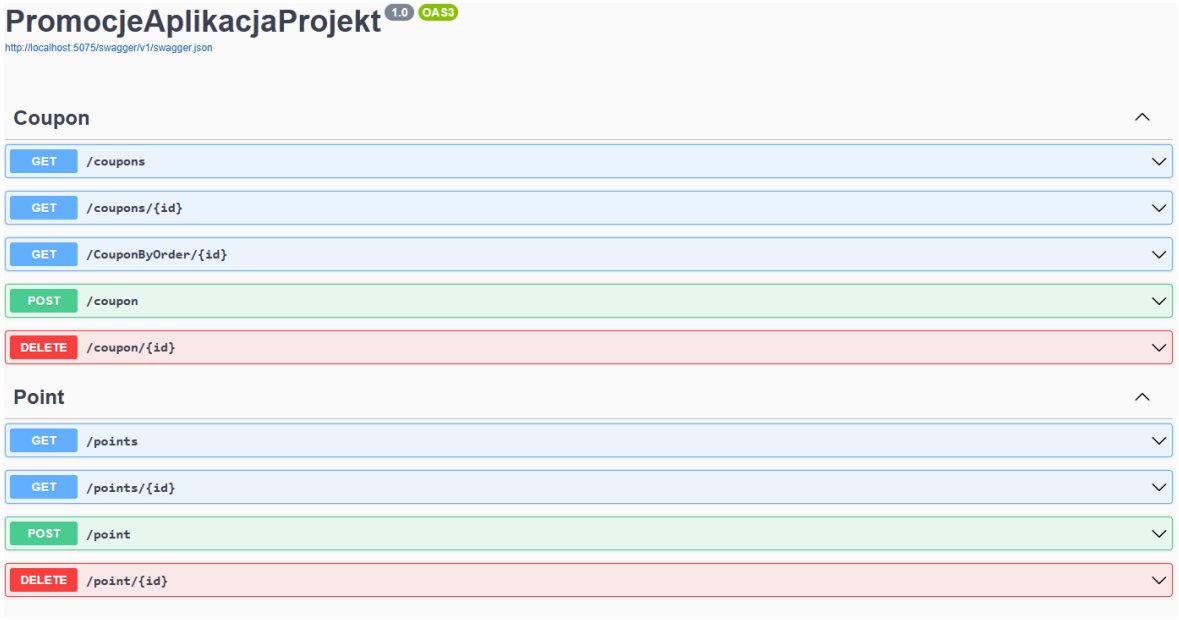
Rys 24 Klasa testów ReaderServiceTests 1

```
[Fact]
0 references
public async void GetReaders_ShouldReturnAllReaders()
{
    var service = GetServiceWithDb(out var context);
    context.Readers.Add(new Reader { Name = "Anna", Surname = "Nowak", Email = "anna.nowak@example.com" });
    context.Readers.Add(new Reader { Name = "Piotr", Surname = "Zielinski", Email = "piotr.zielinski@example.com" });
    context.SaveChanges();

    var readers = (await service.Get()).ToList();

    Assert.Equal(2, readers.Count);
    Assert.Contains(readers, r => r.Name == "Anna");
    Assert.Contains(readers, r => r.Name == "Piotr");
}
```

Rys 25 Klasa testów ReaderServiceTests 2

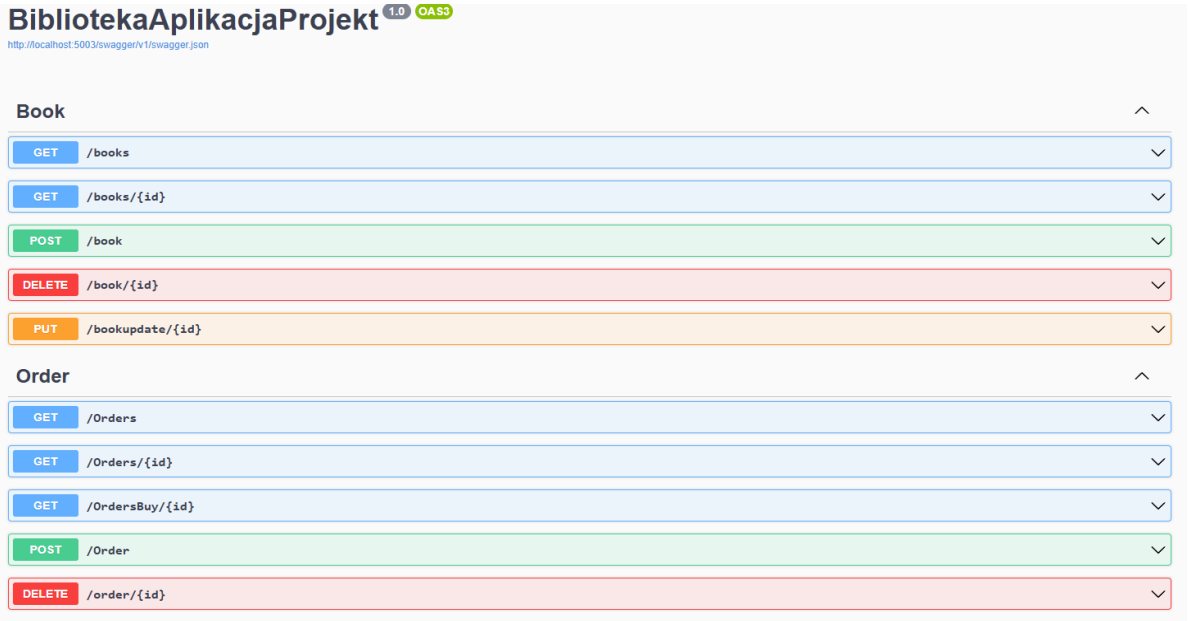
4. Widok serwisów

4.1 Serwis PromocjeAplikacjaProjekt



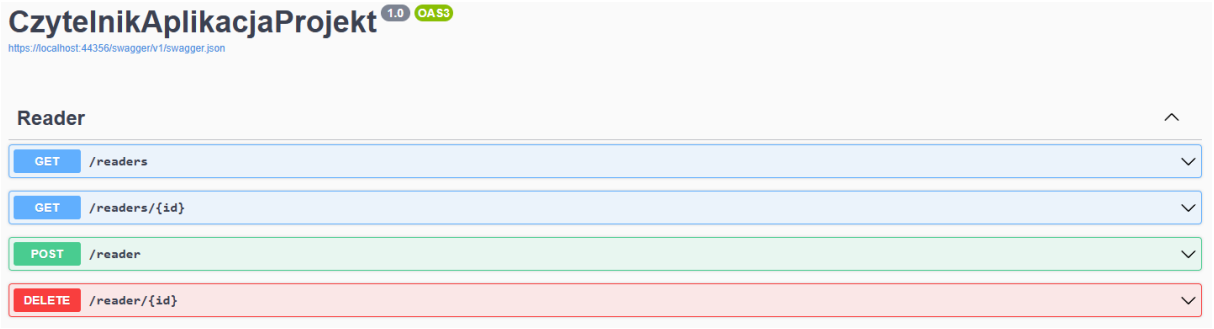
Rys 26 Serwis PromocjeAplikacjaProjekt

4.2 Serwis BibliotekaAplikacjaProjekt



Rys 27 Serwis BibliotekaAplikacjaProjekt

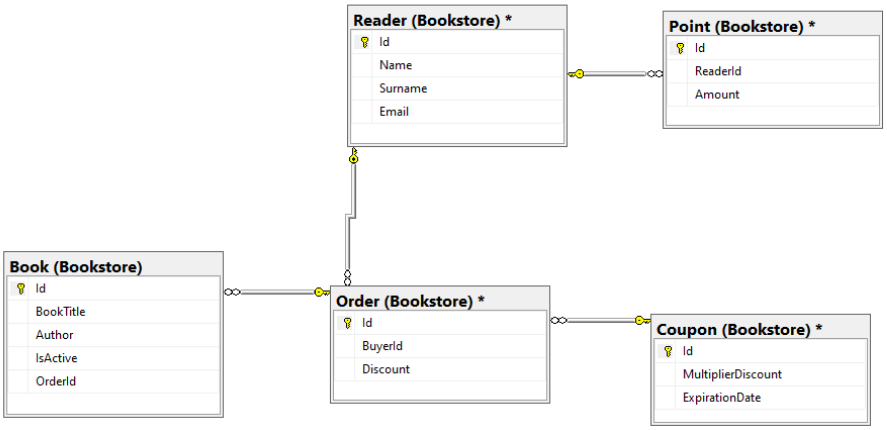
4.3 Serwis CzytelnikAplikacjaProjekt



Rys 28 Serwis CzytelnikAplikacjaProjekt

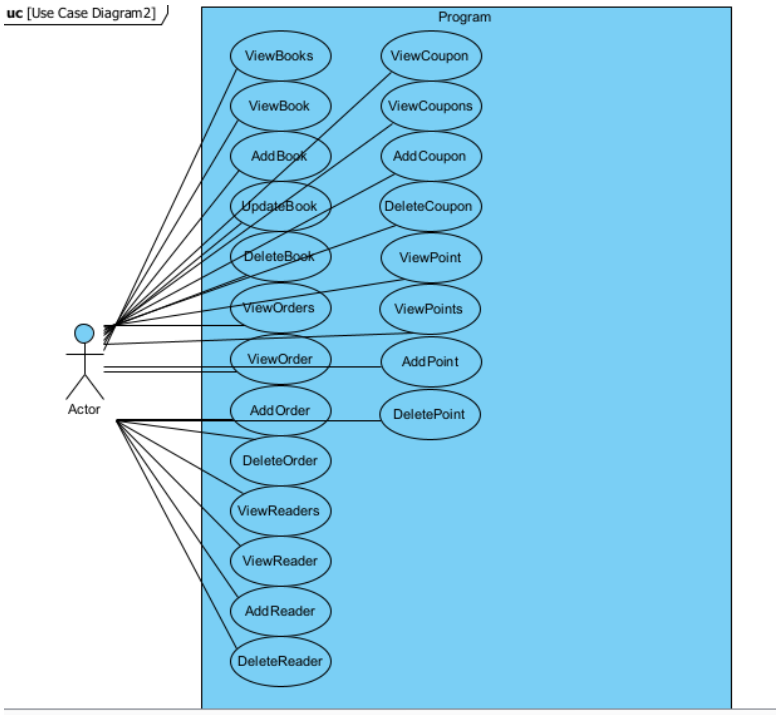
5.Diagram bazy danych i diagram UML

5.1 Diagram bazy danych



Rys 29 Diagram bazy danych

5.2 Diagram UML przypadków użycia



Rys 30 Diagram UML przypadków użycia

6.Testy Gherkin

Test 1:

Feature: Wyświetlanie zamówień dla danego czytelnika

Scenario: Czytelnik ma przypisane zamówienia

Given czytelnik o ID 1 istnieje w systemie

And system biblioteka zawiera zamówienia przypisane do czytelnika o ID 1

When użytkownik żąda listy zamówień dla czytelnika o ID 1

Then system zwraca listę zamówień zawierającą co najmniej 1 element

Test 2

Feature: Wyświetlanie zamówień dla danego czytelnika

Scenario: Czytelnik nie ma przypisanych zamówień

Given czytelnik o ID 999 istnieje w systemie

And system biblioteka nie zawiera zamówień dla czytelnika o ID 999

When użytkownik żąda listy zamówień dla czytelnika o ID 999

Then system zwraca pustą listę

7Literatura

- https://learn.microsoft.com/pl-pl/ef/
- Materiały z zajęć laboratorium Szkolenie techniczne 3
- https://learn.microsoft.com/pl-pl/aspnet/overview
- https://www.newtonsoft.com/json/help/html/deserializeobject.htm
- https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-csharp-with-xunit