Sprawozdanie z projektu - System Zarządzania Biblioteką

Autorzy projektu

- Szymon Mikolajek - Nr 75691
- Sebastian Krause - Nr 45647
- Stefan Reszel - Nr 75696
- Dariusz Niewdana

Wprowadzenie

Niniejsze sprawozdanie przedstawia projekt systemu do zarządzania biblioteką, który został zaimplementowany zgodnie z wymaganiami projektu zaliczeniowego. System został zbudowany w oparciu o język C# z wykorzystaniem architektury mikrousługowej, gdzie backend napisany jest w technologii .NET, frontend w React z Tailwind CSS, a baza danych to PostgreSQL.

Aplikacja realizuje kluczowe funkcjonalności biblioteki, takie jak:

- Zarządzanie książkami (dodawanie, usuwanie, wyszukiwanie)
- Autentykacja i autoryzacja użytkowników
- Obsługa wypożyczeń i zwrotów książek
- Panel administracyjny do zarządzania użytkownikami i książkami
- Responsywny interfejs użytkownika

Paradygmat programowania obiektowego

W projekcie zastosowano liczne elementy i zasady programowania obiektowego, które stanowią fundament architektury systemu:

1. Klasy

W projekcie zaimplementowano następujące kluczowe klasy, które modelują główne encje systemu:

- **Book** - reprezentuje książkę w systemie bibliotecznym
- **User** - reprezentuje użytkownika systemu
- **CheckOut** - reprezentuje wypożyczenie książki
- **ApplicationDbContext** - klasa kontekstu bazy danych odpowiedzialna za mapowanie obiektów na struktury bazodanowe
- **Serwisy implementujące interfejsy** (BookService, UserService, CheckOutService, EmailService)

2. Interfejsy

Interfejsy odgrywają kluczową rolę w zapewnieniu modułowości i łatwości testowania aplikacji:

- **IBookService** - definiuje operacje związane z zarządzaniem książkami
- **IUserService** - definiuje operacje związane z zarządzaniem użytkownikami
- **ICheckOutService** - definiuje operacje związane z wypożyczeniami

- **IEmailService** - definiuje operacje związane z wysyłaniem wiadomości email

3. Enumy

Zastosowano następujące typy wyliczeniowe (enum) do reprezentowania stałych wartości:

- **Genre** - kategorie książek (Fiction, NonFiction, Mystery, SciFi, itd.)
- **UserRole** - role użytkowników (Customer, Admin)
- **CheckOutStatus** - status wypożyczenia (Active, Returned, Overdue, Lost)

4. Dziedziczenie i implementacja interfejsów

W projekcie zastosowano dziedziczenie i implementację interfejsów:

- Klasa `ApplicationDbContext` dziedziczy po klasie `DbContext` z Entity Framework
- Klasy serwisów (np. `BookService`) implementują odpowiadające im interfejsy (np. `IBookService`)

Wykorzystane wzorce projektowe

Wzorzec Dependency Injection (Wstrzykiwanie zależności)

Jest to jeden z głównych wzorców projektowych wykorzystanych w aplikacji. Został zaimplementowany przy użyciu wbudowanego kontenera DI w ASP.NET Core.

Opis wzorca

Dependency Injection to wzorzec projektowy, który polega na dostarczaniu zależności obiektu (usług) z zewnątrz, zamiast tworzenia ich wewnątrz klasy. Dzięki temu podejściu kod staje się:

- Bardziej modułowy
- Łatwiejszy w testowaniu
- Bardziej elastyczny i konfigurowalny
- Zgodny z zasadą odwrócenia zależności (Dependency Inversion Principle) z zasad SOLID

Implementacja w projekcie

W naszym projekcie wzorzec ten jest widoczny w pliku `Program.cs`, gdzie rejestrowane są wszystkie zależności systemu:

```
// Rejestracja serwisów w kontenerze DI
builder.Services.AddScoped<IBookService, BookService>();
builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddScoped<ICheckOutService, CheckOutService>();
builder.Services.AddScoped<IEmailService, EmailService>();
builder.Services.AddSingleton<JwtService>();
```

Dzięki tej rejestracji, gdy kontroler wymaga implementacji interfejsu `IBookService`, kontener DI automatycznie dostarcza mu instancję klasy `BookService`. Przykładowo, w kontrolerze `BooksController` zależności są wstrzykiwane poprzez konstruktor:

```
public class BooksController : ControllerBase
{
    private readonly IBookService _bookService;

    public BooksController(IBookService bookService)
    {
        _bookService = bookService;
    }

    // Metody kontrolera korzystające z _bookService
}
```

Wzorzec Repository (Repozytorium)

W projekcie wykorzystano również wzorzec Repository, choć w nieco zmodyfikowanej formie, połączonej z Entity Framework.

Opis wzorca

Wzorzec Repository oddziela logikę dostępu do danych od logiki biznesowej aplikacji. Zapewnia abstrakcję warstwy dostępu do danych, dzięki czemu reszta aplikacji może pracować z danymi bez znajomości szczegółów ich przechowywania.

Implementacja w projekcie

W naszym systemie wzorzec ten jest widoczny w organizacji serwisów i interfejsów. Na przykład, interfejs `IBookService` definiuje operacje związane z książkami, a jego implementacja `BookService` zajmuje się wykonaniem tych operacji z wykorzystaniem `ApplicationDbContext`:

```
public class BookService : IBookService
{
    private readonly ApplicationDbContext _context;

    public BookService(ApplicationDbContext context)
    {
        _context = context;
    }

    public async Task<IEnumerable<Book>> GetAllBooksAsync()
    {
        return await _context.Books.ToListAsync();
    }

    // Pozostałe metody implementujące IBookService
}
```

Wzorzec MVC (Model-View-Controller)

Chociaż w API backendowym nie ma klasycznego widoku (View), to struktura projektu opiera się na zasadach wzorca MVC.

Opis wzorca

Model-View-Controller to wzorzec architektoniczny, który oddziela logikę aplikacji na trzy główne komponenty:

- Model - reprezentuje dane i reguły biznesowe
- View - wyświetla dane użytkownikowi (w przypadku API to odpowiedź JSON)
- Controller - obsługuje żądania użytkownika, komunikując się z modelem i widokiem

Implementacja w projekcie

W naszej aplikacji:

- Models (np. `Book`, `User`, `CheckOut`) reprezentują strukturę danych
- Controllers (np. `BooksController`, `AuthController`) obsługują żądania HTTP
- Services (np. `BookService`) zawierają logikę biznesową

Podsumowanie

Projekt systemu zarządzania biblioteką skutecznie implementuje wymagane elementy programowania obiektowego, w tym klasy, interfejsy, enumy oraz wzorce projektowe. Szczególną uwagę poświęcono zastosowaniu wzorca Dependency Injection, który jest kluczowy dla architektury całego systemu.

Dzięki zastosowaniu tych elementów, aplikacja jest:

- Modułowa i łatwa w rozbudowie
- Zgodna z zasadami SOLID
- Testowalna i łatwa w utrzymaniu
- Skalowalna dzięki architekturze mikrousługowej