WYŻSZA SZKOŁA
INFORMATYKI i ZARZĄDZANIA
z siedzibą w Rzeszowie

KOLEGIUM INFORMATYKI STOSOWANEJ

Kierunek: INFORMATYKA

Grupa: 3IIZ/2024-L2

Jakub Kluk
Nr albumu studenta 71327

Aplikacja do zarządzania biblioteką

Pod kierunkiem: mgr inż. Ewa Żesławska

PROJEKT

Rzeszów 2026

Spis treści

Rozdział 1

Wprowadzenie do projektu

1.1 Opis założeń projektu

Współczesne biblioteki coraz częściej sięgają po rozwiązania informatyczne, które usprawniają proces zarządzania zbiorami oraz obsługę czytelników. Tradycyjne metody oparte na ewidencji papierowej są nieefektywne, czasochłonne i podatne na błędy ludzkie. W odpowiedzi na te problemy powstała aplikacja „Biblioteka" – system mający na celu wspomaganie pracy bibliotekarzy poprzez automatyzację najważniejszych czynności związanych z wypożyczaniem i zwracaniem książek, a także prowadzeniem ewidencji czytelników i księgozbioru.

Projekt „Biblioteka" został zaprojektowany jako aplikacja konsolowa, co pozwala na jego elastyczne wykorzystanie w różnych środowiskach. Głównym założeniem projektu jest stworzenie intuicyjnego i przejrzystego systemu, który umożliwi bibliotekarzowi skuteczne zarządzanie zasobami bibliotecznymi. Program umożliwia między innymi wypożyczanie książek, rejestrowanie ich zwrotów, a także sprawdzanie dostępności konkretnych pozycji. Dodatkowo program będze pokazywał wszystkie aktualnie wypożyczone pozycje przez czytelnika, co zwiększa przejrzystość i uporządkowanie całego procesu.

Istotnym elementem aplikacji jest wbudowany mechanizm obliczania kar za opóźnienia w zwrocie książek.Automatyzuje to proces, który w tradycyjnych bibliotekach wymaga ręcznych obliczeń i kontroli terminów. Dzięki temu program nie tylko usprawnia obsługę czytelników, lecz także minimalizuje ryzyko pomyłek oraz wspiera bibliotekarzy w egzekwowaniu zasad wypożyczania.

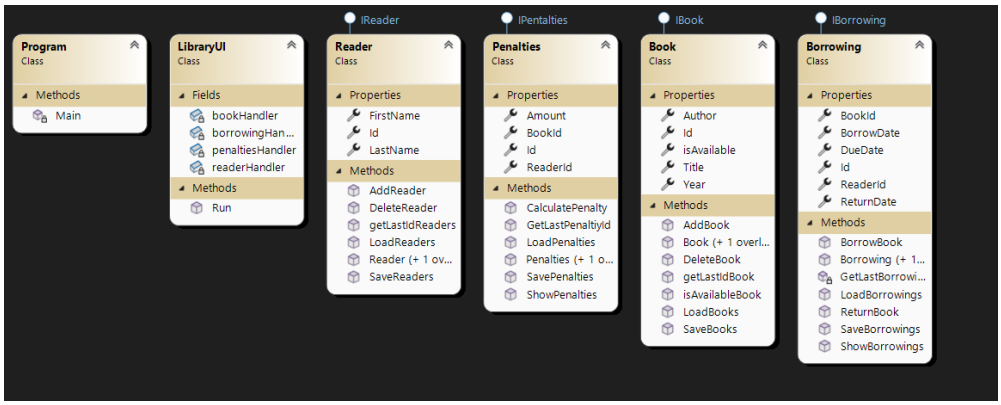
Celem realizacji niniejszego projektu jest zaprojektowanie i zaimplementowanie prostego, a zarazem funkcjonalnego systemu informatycznego, który mógłby znaleźć zastosowanie zarówno w małych bibliotekach szkolnych, jak i w większych placówkach o lokalnym zasięgu. Projekt ten stanowi także praktyczne ćwiczenie w zakresie programowania obiektowego oraz stosowania zasad inżynierii oprogramowania. Aplikacja „Biblioteka" jest przykładem narzędzia, które w przystępny sposób pokazuje, jak technologia może wspierać procesy organizacyjne i usprawniać codzienną pracę instytucji zajmujących się udostępnianiem zbiorów książkowych.

1.2 Opis struktury projektu

1.2.1 Opis techniczny projektu

Poniżej znajduje się wygenerowany diagram klas w programie Visual Studio. Jak można zobaczyć aplikacja składa się z kilku klas, gdzie każda z nich pełni zupełnie inną funkcję. Do każdej klasy mamy zdefiniowane interfejsy, które to właśnie zakładają jakie dokładnie elementy będą zawierać się w danej klasie. Szczegółowo każda z nich została opisana w dalszej części dokumentacji projektowej. Aplikacja ta została napisana w języku C# oraz obsługa jej odbywa się poprzez interfejs konsolowy. Takie podejście jest bardzo efektywne w swoim użyciu, gdyż aplikacje konsolowe ważą znacznie mniej, niż aplikacje okienkowe z GUI. Do uruchomienia aplikacji w środowisku programistycznych wymagany jest system Windows 10 lub Windows 11 oraz program Visual Studio 2022 lub nowszy.

Rysunek 1.1: Diagram klas wygenerowany w programie Visual Studio



Źródło: opracowanie własne

Do zbudowania aplikacji zdecydowano się na użycie nierelacyjnego mechanizmu przechowywania danych, w którym dane są zapisywane do pliku w formacie JSON. Lokalizacja tych plików jest taka sama jak plik Biblioteka.exe. W każdej z klas zaimplementowane są metody z Load i Save w nazwie. Poniżej zaprezentowano implementację odczytu oraz zapisu danych dla klasy Reader.

1.2.2 Odczyt danych z pliku

Metoda ta zwraca pustą listę obiektów w przypadku, gdy nie zostanie znaleziony plik readers.json. Gdy jednak plik zostanie znaleziony, dane zostają zapisane do zmiennej typu string o nazwie json. Następnie podejmowana jest próba deserializacji danych, czyli na tym etapie podejmowana jest próba konwersji danych z formatu JSON na listę obiektów klasy Reader. Gdyby to się nie udało metoda zwróci nam pustą listę ww. klasy.

Rysunek 1.2: Metoda odczytująca czytelników z pliku readers.json

```
public List<Reader> LoadReaders()
{
    if (!File.Exists("readers.json"))
        return new List<Reader>();

    string json = File.ReadAllText("readers.json");
    return JsonSerializer.Deserialize<List<Reader>>(json)
          ?? new List<Reader>();
}
```

Źródło: opracowanie własne

1.2.3 Zapis danych do pliku

Metoda zapisująca dane do pliku jest równie prosta, jak ta powyższa. W pierwszym kroku dokonywana jest serializacja danych wejściowych (proces odwrotny do deserializacji). Na tym etapie budowania funkcji można pokusić się o użycie parametru **WriteIndented = true**, który sprawi, że dane w pliku będą zapisane w postaci czytelnej dla człowieka. Następnie za pomocą File.WriteAllText() zapisywane są dane do pliku. Ta metoda przyjmuje dwa parametry: nazwa pliku (w tym przypadku readers.json) oraz dane wejściowe, czyli string o nazwie json. Gdyby plik nie istniał, to automatycznie utworzy się plik o nazwie wskazanej w pierwszym parametrze.

Rysunek 1.3: Metoda zapisująca czytelników do pliku readers.json

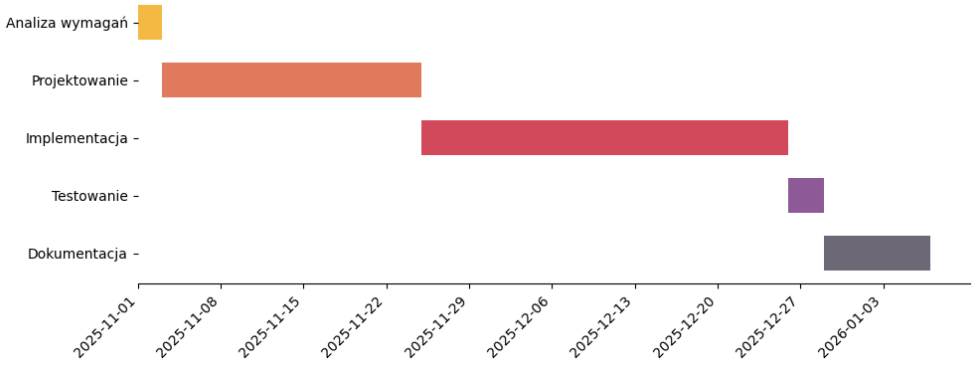
```
public void SaveReaders(List<Reader> readers)
{
    string json = JsonSerializer.Serialize(
        readers,
        new JsonSerializerOptions { WriteIndented = true }
    );

    File.WriteAllText("readers.json", json);
}
```

Źródło: opracowanie własne

1.3 Harmonogram realizacji projektu

Kluczowym elementem skutecznej realizacji projektu jest odpowiednie i przemyślane zaplanowanie poszczególnych prac. Podział projektu na mniejsze etapy umożliwia lepszą kontrolę postępów, efektywne zarządzanie zasobami oraz szybsze reagowanie na ewentualne problemy. Harmonogram realizacji projektu pozwala na określenie kolejności działań oraz terminów ich wykonania, co znacząco zwiększa szanse na terminowe i poprawne ukończenie projektu.

Rysunek 1.4: Diagram Ganta prezentujący etapy prowadzonych prac



Źródło: opracowanie własne

Analiza wymagań jest pierwszym i jednym z najważniejszych etapów projektu. Jej celem jest dokładne określenie potrzeb i oczekiwań przyszłych użytkowników oraz sprecyzowanie głównych celów, jakie projekt ma spełniać. W trakcie tego etapu identyfikuje się funkcjonalności systemu, określa ograniczenia techniczne oraz wymogi niefunkcjonalne, takie jak wydajność, niezawodność czy bezpieczeństwo. Wyniki analizy stanowią solidną podstawę do kolejnych faz projektu, w tym projektowania i implementacji, pozwalając zminimalizować ryzyko błędów i nieporozumień w dalszych etapach prac. Dla tego projektu, będącego aplikacją konsolową, przewidziano na realizację tego etapu 2 dni.

Projektowanie polega na opracowaniu szczegółowej koncepcji oraz struktury systemu, w oparciu o zebrane wcześniej wymagania. Tworzy się tutaj schematy, diagramy oraz modele funkcjonalne, które obrazują działanie poszczególnych elementów projektu i ich wzajemne powiązania. Dobrze przygotowany etap projektowania znacząco ułatwia późniejszą implementację oraz minimalizuje ryzyko pojawienia się błędów w kodzie. Ze względu na swoją złożoność, na ukończenie tego etapu przewidziano 21 dni.

Implementacja to etap, w którym przygotowany projekt przekształca się w działający system. Efektywna implementacja opiera się na starannie przeprowadzonej analizie wymagań i projektowaniu, co pozwala sprawnie wdrożyć wszystkie funkcjonalności i uniknąć błędów. Jest to najbardziej czasochłonny etap całego projektu, stanowiący właściwą pracę nad aplikacją, i przewidziano na niego 30 dni.

Testowanie ma na celu weryfikację, czy system działa zgodnie z założeniami oraz spełnia wszystkie wymagania funkcjonalne i niefunkcjonalne. W jego trakcie wykrywa się potencjalne błędy i wprowadza niezbędne poprawki, aby zapewnić stabilność, niezawodność i poprawne działanie systemu. Na wykonanie testów zaplanowano 2 dni.

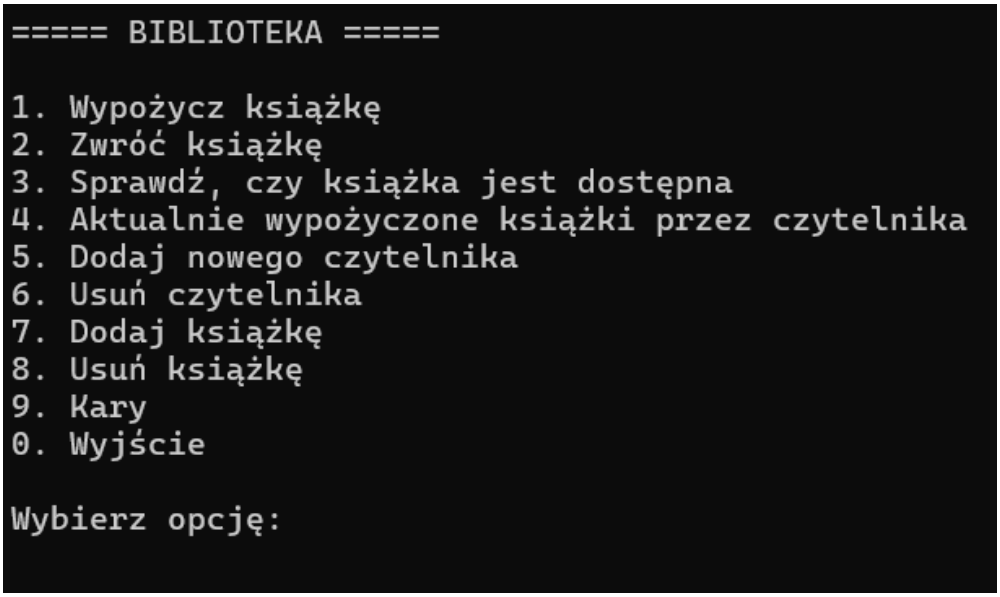
Dokumentacja obejmuje opis działania systemu, instrukcje dla użytkowników oraz informacje techniczne niezbędne do utrzymania i dalszego rozwoju projektu. Starannie przygotowana dokumentacja ułatwia korzystanie z systemu, wspiera przyszłe modyfikacje oraz umożliwia sprawne wdrażanie nowych funkcjonalności. Na przygotowanie dokumentacji przewidziano 8 dni.

Rozdział 2

Prezentacja warstwy użytkowej projektu

Po uruchomieniu aplikacji zaprezentowany zostaje poniższy widok:

Rysunek 2.1: Widok główny aplikacji



Źródło: opracowanie własne

2.1 Prezentacja klasy Program

W pierwszej kolejności po przekompilowaniu aplikacji zostaje uruchomiona klasa Program. Jest tu jedynie utworzenie obiektu klasy LibraryUI o nazwie ui, który to obiekt w uruchomionej metodzie Run() zawiera cały interfejs użytkownika. Aby zachować porządek cała logika aplikacji została przeniesiona do innej klasy. Zabieg ten ma na celu zachowania porządku w kodzie.

Rysunek 2.2: Klasa Program

```
using Biblioteka;

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        LibraryUI ui = new LibraryUI();
        ui.Run();
    }
}
```

Źródło: opracowanie własne

2.2 Omówienie działania klasy Library UI

Nieco bardziej rozbudowaną strukturą jest właśnie klasa LibraryUI. Na wstępie zdefiniowany zostaje sam widok aplikacji, poprzez wyświetlenie go na konsolę użytkownika.

Rysunek 2.3: Wygenerowanie menu

```
Console.Clear();
Console.WriteLine("===== BIBLIOTEKA =====");
Console.WriteLine();
Console.WriteLine("1. Wypożycz książkę ");
Console.WriteLine("2. Zwróć książkę ");
Console.WriteLine("3. Sprawdź, czy książka jest dostępna");
Console.WriteLine("4. Aktualnie wypożyczone książki przez czytelnika");
Console.WriteLine("5. Dodaj nowego czytelnika ");
Console.WriteLine("6. Usuń czytelnika ");
Console.WriteLine("7. Dodaj książkę ");
Console.WriteLine("8. Usuń książkę ");
Console.WriteLine("9. Kary ");
Console.WriteLine("0. Wyjście");
Console.WriteLine();
Console.Write("Wybierz opcję: ");
```

Źródło: opracowanie własne

Po wyświetleniu całego menu na konsolę użytkownik otrzymuje pytanie z jakiej funkcjonalności chce obecnie skorzystać. Następnie w bloku try próbujemy za pomocą instrukcji wielokrotnego wyboru (switch) dopasować wprowadzoną wartość przez użytkownika do właściwej instrukcji. Gdy ta operacja zakończy się sukcesem zostanie wywołana żądana metoda. Gdy wprowadzona zostanie nieprawidłowa wartość, aplikacja zwróci komunikat o treści: **Nieznana opcja!**

Rysunek 2.4: Blok try

```
try
{
    switch (option)
    {

        case "1": borrowingHandler.BorrowBook(); break;
        case "2": borrowingHandler.ReturnBook(); break;
        case "3": bookHandler.isAvailableBook(); break;
        case "4": borrowingHandler.ShowBorrowings(); break;
        case "5": readerHandler.AddReader(); break;
        case "6": readerHandler.DeleteReader(); break;
        case "7": bookHandler.AddBook(); break;
        case "8": bookHandler.DeleteBook(); break;
        case "9": penaltiesHandler.ShowPenalties(); break;

        case "0": return;
        default: Console.WriteLine("Nieznana opcja!"); break;
    }
}
```

Źródło: opracowanie własne

Blok catch z kolei służy do tego, aby nie doszło do wysypania się aplikacji poprzez wyrzucenie jakiegoś wyjątku. Gdy program wyrzuci jakiś Exception to użytkownik otrzyma jedynie stosowny monit. Aplikacja jest chroniona w pełni przed wystąpieniem niepożądanych czynności. Każdy komunikat został precyzyjnie zdefiniowany, po to aby użytkownik końcowy bez większych problemów poradził sobie z nieprzewidzianymi trudnościami.

Rysunek 2.5: Blok catch

```
catch (Exception ex)
{
    Console.WriteLine($"Błąd: {ex.Message}");
}
```

Źródło: opracowanie własne

Zastanawiające mogą być nazwy zawarte w bloku case z dopiskiem Handler, np. bookHandler, readerHandler itp. Jak można zauważyć to na nich wywołujemy metody z określonych klas. Działa to w taki sposób, że tworzymy bezargumentowy obiekt bookHandler, który nie przekazuje żadnych parametrów do konstruktora tej klasy. Opcja taka pozwala nam na korzystanie jedynie z metod zawartych w danej klasie.

Rysunek 2.6: Handlery

```
Book bookHandler = new Book();
Reader readerHandler = new Reader();
Borrowing borrowingHandler = new Borrowing();
Penalties penaltiesHandler = new Penalties();
```

Źródło: opracowanie własne

2.3 Dodanie nowego czytelnika

Aby korzystać z programu, należy uprzednio wprowadzić do niego jakieś dane o czytelnikach, książkach, wypożyczeniach, zwrotach itp. Dobrym pomysłem na rozpoczęcie pracy z aplikacją jest początkowo dodanie nowych czytelników do bazy. Aby tego dokonać w głownym menu wybrać należy opcję numer 5. Poniżej zaprezentowano działanie metody dodającej czytelnika do bazy.

Rysunek 2.7: Dodanie czytelnika

```
===== BIBLIOTEKA =====

1. Wypożycz książkę
2. Zwróć książkę
3. Sprawdź, czy książka jest dostępna
4. Aktualnie wypożyczone książki przez czytelnika
5. Dodaj nowego czytelnika
6. Usuń czytelnika
7. Dodaj książkę
8. Usuń książkę
9. Kary
0. Wyjście

Wybierz opcję: 5
Id tego czytelnika to: 1
Podaj Imię: Jakub
Podaj Nazwisko: Kluk

Dodano nowego czytelnika!

Wciśnij ENTER aby kontynuować...
```

Źródło: opracowanie własne

W klasie Reader (bo to o niej mowa) przede wszystkim pracę należy zacząć od wykonania dziedziczenia po właściwym dla klasy interfejsie (w tym przypadku IReader). W tym właśnie interfejsie zadeklarowane są elementy, które muszą być zaimplementowane w klasie Reader. W przeciwnym wypadku, gdy tego nie zrobimy otrzymamy błąd o niezadeklarowanych składnikach. Dodatkowo błąd który otrzymamy będzie zawierał w sobie dokładną nazwę elementu, który nie został zadeklarowany. Sama klasa zawiera w sobie 3 właściwości: **Id, FirstName, LastName** oraz 4 metody: **AddReader, DeleteReader, SaveReaders oraz LoadReaders**, które na etapie projektowania założono, że się tam pojawią. Bez tych składników program nie będzie działała prawidłowo.

Z ważniejszych rzeczy warto również wspomnieć o tym, że praca na plikach json wymaga nieco wiedzy. W momencie, gdy dodajemy czytelnika do bazy pojawia się informacja o tym jakie id otrzyma ta osoba. Identyfikator ten jest generowany w sposób taki, że ściągamy sobie bazę do zmiennej (lokalnej) i szukamy max id. Aby ściągnąć wszystkich czytelników do programu, należy skorzystać z listy tej klasy (czyli tutaj List<Reader>). Do niej właśnie za pomocą metody LoadReaders() ładowane są niezbędne dane. Po znalezieniu max id wartość ta przed wyświetleniem jest inkrementowana, czyli jej wartość jest zwiększana o 1.

W momencie, gdy użytkownik poda wszystkie niezbędne dane takie jak imię i nazwisko wówczas w programie generowany zostaje obiekt tej klasy, a następnie całość zostaje dodana do listy o nazwie readers. W końcowej fazie działania programu dane zostają wypchnięte do pliku, za pomocą metody SaveReaders() oraz finalnie zostaje wyświetlony stosowny komunikat.

Rysunek 2.8: Pozostałe instrukcje w metodzie AddReader()

```
Reader reader = new Reader(id, firstName, lastName);
readers.Add(reader);

SaveReaders(readers);

Console.WriteLine();
Console.WriteLine("Dodano nowego czytelnika!");
```

Źródło: opracowanie własne

2.4 Usunięcie czytelnika

Usunięcie czytelnika z bazy zaimplementowano w następujący sposób. W pierwszej kolejności zostaje zadane pytanie użytkownikowi jakie id chce on usunąć. Funkcja ta jest zbudowana w taki sposób, że w przypadku nie podania wartości integer wyświetlony zostaje odpowiedni komunikat ostrzegający. Natomiast gdy użytkownika nie znajdziemy w bazie to zwrócony zostaje następujący komunikat: **Nie znaleziono czytelnika o podanym ID**. Gdy czytelnik zostanie znaleziony wówczas w następnym kroku zostaje zadane pytanie, czy aby na pewno tego czytelnika chcemy wyrzucić z bazy. Pytanie to dodatkowo zawiera w sobie imię i nazwisko osoby, która za chwile zostanie usunięta z tej bazy. Zabieg ten ma na celu wyeliminowanie wystąpienia błędu polegającego na usunięciu innej osoby z listy czytelników.

Rysunek 2.9: Mechanizm usuwania czytelnika z bazy

```
Wybierz opcję: 6
Podaj id użytkownika, którego chcesz usunąć: 2
Czy na pewno chcesz usunąć użytkownika Marek Kowal (T/N): t
Czytelnik został prawidłowo usunięty z bazy danych

Wciśnij ENTER aby kontynuować...
```

Źródło: opracowanie własne

2.5 Klasa Book

Tak jak poprzednio na tym etapie pracę właściwą zaczynamy od zdefiniowania interfejsu dla wspomnianej klasy. Ten interfejs deklaruje wszystkie elementy niezbędne do obsługi książek w bazie.

Rysunek 2.10: Składowe interfejsu IBook

```
//properties
8 references
int Id { get; set; }
3 references
string? Title { get; set; }
2 references
string? Author { get; set; }
2 references
int Year { get; set; }
6 references
bool isAvailable { get; set; }

//methods
2 references
void AddBook();
2 references
void DeleteBook();
2 references
void isAvailableBook();
5 references
void SaveBooks(List<Book> books);
8 references
List<Book> LoadBooks();
```

Źródło: opracowanie własne

Właściwości tutaj użyte to parametry książki, czyli jej tytuł, autor, rok wydania itp. Pozycje książkowe w naszej bazie posiadają jeszcze dodatkowo identyfikator oraz właściwość typu bool, która będzie przechowywała informację, czy książka jest aktualna dostępna. Jeżeli program napotka wartość false, będzie to oznaczać, że książka jest wypożyczona przez innego czytelnika. Metody AddBook() i DeleteBook() technicznie rzecz biorąc nie różnią się od siebie zbytnio od swoich braci z klasy Reader, podobnie jak metoda SaveBooks() oraz LoadBooks(). Metoda isAvailableBook() podaje informacje, czy książka jest dostępna, czy jej nie ma. Aby skorzystać z tej funkcjonalności w głównym menu wybrać należy opcje numer 3. Cała jego funkcjonalność opiera się właśnie na właściwości isAvailable. Poniżej zaprezentowano jak wyglądają zapisane dane już w pliku docelowym.

Rysunek 2.11: Zapisana książka w pliku books.json

```
{
  "Id": 2,
  "Title": "Lalka",
  "Author": "Boles\u0142aw Prus",
  "Year": 1887,
  "isAvailable": true
}
```

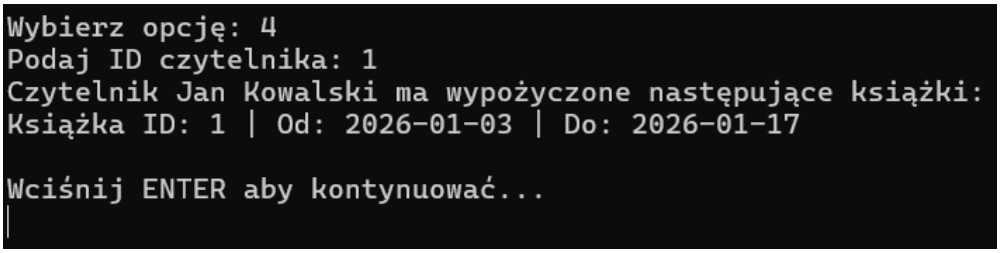
Źródło: opracowanie własne

2.6 Pozostałe klasy

2.6.1 Wypożyczenie książki

Biblioteka w swoim założeniu powinna wzorowo obsługiwać swoich czytelników. Przez to pojęcie rozumie się możliwość wypożyczania oraz zwrotu wybranych pozycji. Wypożyczyenie książki działa w sposób prosty i intuicyjny. Użytkownik aplikacji wprowadza dane identyfikacyjne czytelnika oraz pozycję jaką on wypożycza. W momencie, gdy taka operacja zakończy się sukcesem możemy poprzez funkcjonalność numer 4 podejrzeć wszystkie wypożyczone pozycje przez tego człowieka.

Rysunek 2.12: Lista wypożyczonych książek



Źródło: opracowanie własne

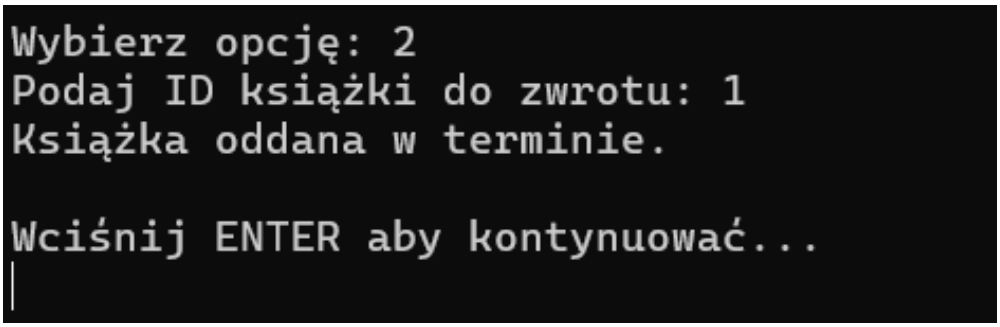
2.6.2 Zwrot książki

Opcję zwrotu książki zaimplementowano pod pozycją numer 2. Bazowo książka wypożyczona jest na 14 dni, a opóźnienia związane ze zwłoką za oddanie książki wynoszą 1zł za dzień. Do obsługi zwrotu potrzebne są następujące daty:

- Data wypożyczenia książki
- Planowa data zwrotu książki
- Rzeczywista Data zwrotu książki

W przypadku, gdy czytelnik odda książkę w terminie wyświetlony zostanie następujący komunikat:

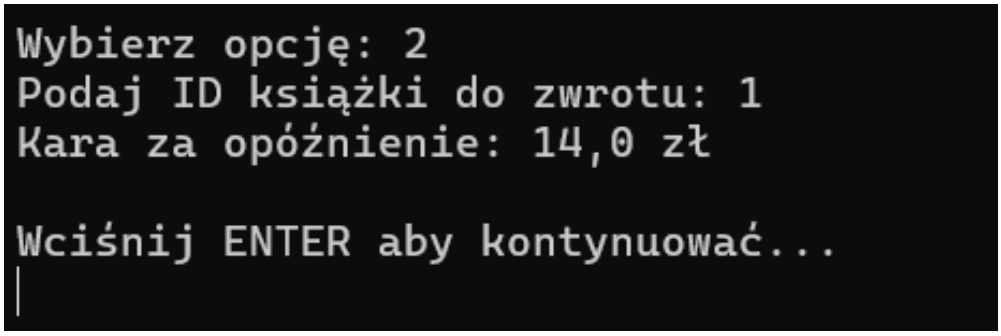
Rysunek 2.13: Zwrot książki w terminie



Źródło: opracowanie własne

Gdyby się jednak okazało, że zwłoka za książkę wynosi np. 2 tygodnie (czyli 14 dni) zostanie naliczona opłata dodatkowa. W tym przypadku do konta czytelnika zostanie doliczone 14 złotych ww. opłaty.

Rysunek 2.14: Zwrot książki po terminie

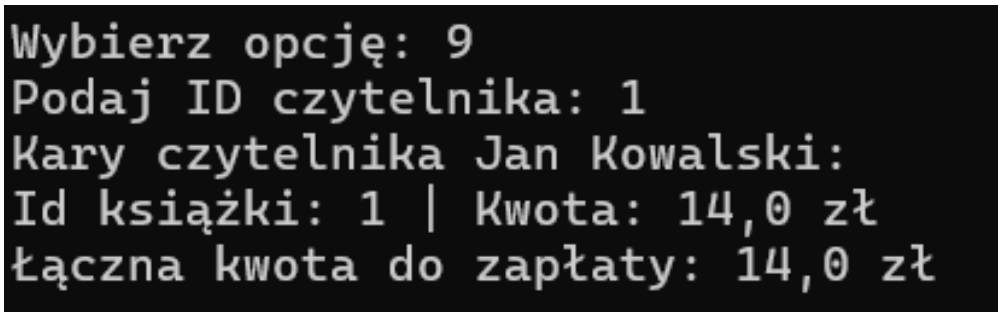


Źródło: opracowanie własne

2.6.3 Kary za zwrot książki po terminie

Czytelnik, który oddaje książki poza planowanym terminem zwrotu obciążany zostaje wówczas opłatą dodatkową za zwłokę. Przy standardowej procedurze zwrotu książki w podsumowaniu prezentowana jest na samym końcu informacja o opóźnieniu. Zaległe opłaty można podejrzeć w opcji numer 9. To tam po wpisanu identyfikatora czytelnika wyświetli się lista, gdzie wyszczególnione są pozycje książkowe wraz z należną opłatą dodatkową. W niżej prezentowanym rysunku w podsumowaniu mamy również całościową kwotę jaką czytelnik musi wnieść za zaległe wypożyczenia.

Rysunek 2.15: Lista książek oddanych po terminie wraz z należną opłatą dodatkową



Źródło: opracowanie własne

Rozdział 3

Podsumowanie

Podsumowując całokształt wykonanych prac, należy stwierdzić, że założone cele projektu zostały zrealizowane, a uzyskane efekty były zgodne z oczekiwanymi rezultatami. Realizacja poszczególnych etapów pozwoliła na pogłębienie wiedzy oraz zdobycie praktycznych umiejętności, które mogą zostać wykorzystane w przyszłych działaniach.

Projekt ten ze względu na jednoosobowy charakter oraz ograniczenie czasowe jest całkiem podatny na przyszłe modyfikacje oraz usprawnienia. W przyszłości planowane jest również wdrożenie relacyjnej bazy danych. Spowodowane jest to faktem, iż bazy tego typu mają wiele ciekawych funkcjonalności, które nie dostarczają mechanizmy zapisu danych do pliku. Jedną z nich jest obsługa transakcji. Jest to mechanizm, który gwarantuje poprawność danych w momencie gdy coś po drodze pójdzie nie tak. Również skalowalność takiego rozwiązania brzmi kusząco gdybyśmy chcieli aby program działał na większą skalę. Przykładów można by było mnożyć. Warto w tym momencie sięgnąć po dobrą literaturę i zapoznać się z wieloma zaletami, jakie oferuje to rozwiązanie. Zapis danych do pliku zagościł w tym projekcie, z uwagi na fakt, iż do prostych projektów (aplikacja konsolowa) stosujemy równie proste rozwiązania jak zapis danych do pliku .json.

Kolejną również interesującą modyfikacją całego projektu jest utworzenie środowiska graficznego. W tak zaprojektowanej aplikacji użytkownik szybciej się odnajdzie ze względu na dodatkowe wrażenia wizualne w postaci kolorów, okienek, pól, tabelek itp. W obecnym czasie aplikacje konsolowe na dobre zagościły u profesjonalistów, szczególne u sysadminów oraz specjalistów ds. cyberbezpieczeństwa. Zaawansowane aplikacje, z jakich korzystają pracują pod systemem operacyjnym linux i działają właśnie w trybie tekstowym. Takie rozwiązanie jest korzystniejsze, ponieważ zapewnia dostęp do pełnej funkcjonalności aplikacji, jaką przewidział jej twórca. Czasami zdarza się, że aplikacja konsolowa ma również wersję okienkową, jednak nie oferuje ona pełnego zakresu funkcji. Zazwyczaj dostępnych jest w niej około 95% funkcjonalności, natomiast pozostałe 5% pozostaje jedynie w wersji konsolowej.

Bibliografia

[1] Fryc Barbara, Wyklad2.pdf (KLASY), materiały dydaktyczne (PDF), WSIiZ 2025

[2] Fryc Barbara, Wyklad3_4_dziedziczenie_polimorfizm.pdf, materiały dydaktyczne (PDF), WSIiZ 2025

[3] Fryc Barbara, Wyklad5_delegaty_wyjatki.pdf, materiały dydaktyczne (PDF), WSIiZ 2025

[4] Żesławska Ewa, Lab04.pdf (POLIMORFIZM, DEFINIOWANIE KLAS ABSTRAKCYJNYCH I INTERFEJSÓW), materiały dydaktyczne (PDF), WSIiZ 2025

[5] Żesławska Ewa, Lab06.pdf (OPERACJE WEJŚCIA-WYJŚCIA, PRACA Z PLIKAMI I SYSTEMAMI BAZODANOWYMI), materiały dydaktyczne (PDF), WSIiZ 2026

[6] Piątek Łukasz, LPiatek_BD_LEC_03_cz01_2025.pdf (TRANSAKCJE), materiały dydaktyczne (PDF), WSIiZ 2025