PRACA DYPLOMOWA

INSTYTUT PODSTAW INFORMATYKI

POLSKIEJ AKADEMIA NAUK

STUDIA PODYPLOMOWE

PROGRAMOWANIE NA PLATFORMIE .NET

APLIKACJA .NET CORE
BIBLIOTEKA MULTIMEDÓW

Autor: inż. Piotr Antończak

Promotor: dr inż. Grzegorz Glonek

WARSZAWA, LUTY 2022

Oświadczenie autora pracy

Oświadczam, że złożoną pracę dyplomową pod tytułem „Aplikacja .NET CORE - Biblioteka Multimediów” napisałem samodzielnie. Jednocześnie oświadczam, że praca ta:

1. Nie narusza praw autorskich osób trzecich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (tekst jednolity Dz. U. 2006 nr 90, poz. 631 z późniejszymi zmianami).
2. Nie zawiera informacji i danych uzyskanych w sposób nielegalny, a w szczególności informacji zastrzeżonych przez inny podmiot albo stanowiących tajemnicę przedsiębiorstwa.
3. Nie była wcześniej przedmiotem innych procedur związanych z uzyskaniem dyplomów/świadectw lub tytułów zawodowych wyższych uczelni. Nadto oświadczam, że treści zawarte w pisemnym egzemplarzu pracy oraz w egzemplarzu tej pracy w formie elektronicznej, złożonych przeze mnie, są jednobrzmiące.

Przyjmuję do wiadomości, że w przypadku stwierdzenia popełnienia przeze mnie czynu polegającego na przypisaniu sobie autorstwa istotnego fragmentu lub innych elementów cudzej pracy, lub ustalenia naukowego, właściwy organ stwierdzi nieważność postępowania w sprawie nadania mi tytułu zawodowego (art. 193 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym, tekst jednolity Dz. U. z 2012 r., poz. 572 z późniejszymi zmianami).

……..

data i podpis

Streszczenie

Przedmiotem pracy dyplomowej jest aplikacja webowa umożliwiającą zarządzanie, katalogowanie i wypożyczanie dowolnych zasobów multimedialnych takich jak książki, czasopisma, płyty, kasety.

Aplikacja została wykonana przy użyciu ASP.NET Core Razor Pages w wersji 5.0 z wykorzystaniem Entity Framework Core w wersji 5.0.2.

Spis treści

Założenia i cel pracy

Celem niniejszej pracy jest stworzenie aplikacji webowej do katalogowania domowych zbiorów bibliotecznych takich jak książki, czasopisma, płyty CD.

W celu realizacji powyższego założenia wykorzystano następujące narzędzia i technologie:

- środowisko programistyczne – Visual Studio 2019 Version 16.11.10;
- projekt typu – ASP.NET Core Web App - Razor Pages Context .NET 5.0;
- framework ORM - Microsoft.EntityFrameworkCore w wersji 5.0.12
- język programowania – C# 9.0;
- baza danych – LocalDB;
- aplikacja do zarządzania bazą danych – SQL Server Management Studio v18.10;
- środowisko do konteneryzacji – Docker.
- framework testów jednostkowych – NUnit (3.13.1);

Analiza wymagań

W ramach analizy wymagań przyjęto następujące założenia dla budowanej aplikacji:

- możliwość dodawania, usuwania i edytowania kont czytelników, którzy mogą korzystać z zasobów biblioteki;
- możliwość dodawania, usuwania i edytowania zasobów biblioteki, które są elementami biblioteki;
- rejestrowanie statusu zasobu biblioteki i ewentualna możliwość zmiany statusu podczas edycji zasobu;
- możliwość dodawania, usuwania i edytowania kart bibliotecznych;
- możliwość dodawania wielu kart bibliotecznych do jednego czytelnika;
- możliwość rejestrowania aktywności czytelników jako wypożyczenie, zwrot lub zagubienie zasobu biblioteki;
- prezentacja aktywności czytelników posortowana według daty i czasu tej aktywności, tak aby najnowsze zdarzenia były pokazywane na górze;
- aplikacja powinna chronić zapis aktywności, tak aby nie było możliwości edycji, ani usuwania zdarzeń z listy aktywności.

Wprowadzenie

Aplikacje webowe (ang. web application) już od wielu lat zdominowały rynek oprogramowania na całym świecie. Główną zaletą aplikacji webowych jest to, że nie trzeba instalować ich na komputerze użytkownika. Aby korzystać z takiej aplikacji wystarczy przeglądarka z dostępem do Internetu. Dzięki temu użytkownik może korzystać z takiej aplikacji praktycznie na dowolnym komputerze.

Nie bez znaczenia jest też dostępność aplikacji webowej na różnych typach urządzeń. Część aplikacji webowych można używać zarówno na komputerze, tablecie czy telefonie.

Równie istotna jest łatwość poprawiania błędów. W aplikacjach desktopowych jest to utrudnione, ponieważ w jakiś sposób aktualizacja musi być dostarczona do użytkownika. W przypadku aplikacji webowych taka aktualizacja może być dla użytkownika niewidoczna. Dzięki temu poprawki błędów czy nowe funkcjonalności są szybciej dostępne dla użytkowników.

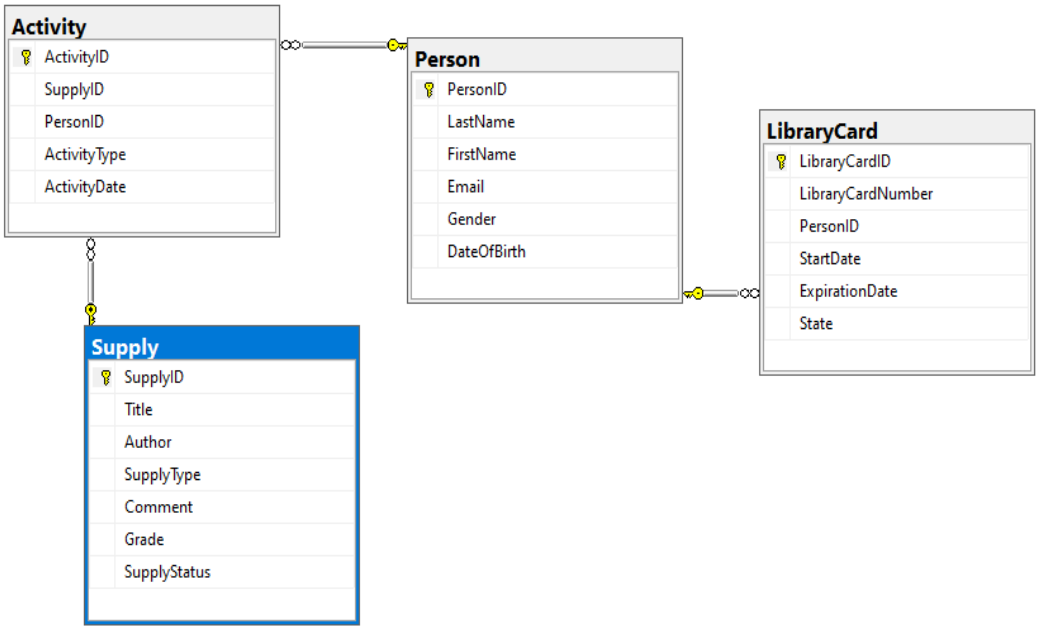
Struktura danych aplikacji

Dane aplikacji są przechowywane w bazie danych SQL. Na etapie budowy aplikacji do przechowywania danych wykorzystano silnik bazy danych LocalDB, który jest częścią składową środowiska programistycznego Visual Studio. Bazę danych LocalDB można wybrać jako opcję podczas instalacji środowiska programistycznego Visual Studio.

Sposób połączenia z bazą danych opisuje *ConnectionStrings* umieszczony w pliku konfiguracyjnym *appsettings.json*, który znajduje się w katalogu głównym aplikacji.

```
  "ConnectionStrings": {
    "LibraryContext":
"Server=(localdb)\\mssqllocaldb;Database=Library-
DB;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
```

Powyżej przedstawiono pełną treść *ConnectionStrings.* W razie potrzeby umieszczenia bazy danych na innym serwerze wystarczy odpowiednio zmodyfikować wpis *ConnectionStrings* wskazując na bazę danych w której mają być przechowywane dane.



Rysunek 1 Struktura danych aplikacji Biblioteka Multimediów

Baza danych aplikacji składa się z czterech tabel powiązanych ze sobą odpowiednimi kluczami. Na Rysunku 1 przedstawiono ogólną strukturę i powiązania pomiędzy tabelami.

Poniżej przedstawiono typy danych w poszczególnych tabelach.

Supply

	Column Name	Data Type	Allow Nulls
🔑	SupplyID	int	☐
	Title	nvarchar(MAX)	☐
	Author	nvarchar(MAX)	☐
	SupplyType	int	☐
	Comment	nvarchar(MAX)	☑
	Grade	int	☑
	SupplyStatus	int	☐
			☐

Rysunek 2 Tabela Supply przechowywująca dane zasobów biblioteki

Zasoby biblioteki czyli książki, czasopisma i inne multimedia są przechowywane w tabeli *Supply*. Typy danych w tej tabeli przedstawia rysunek 2. Dane takie jak tytuł, autor, typ zasobu, czyli informacja czy zasób jest książką, lub płytą CD są obowiązkowe. Komentarz oraz ocena zasobu są obligatoryjne i nie ma obowiązku podawania tych danych.

Kolejną tabelą jest zbiór informacji o czytelnikach.

Person

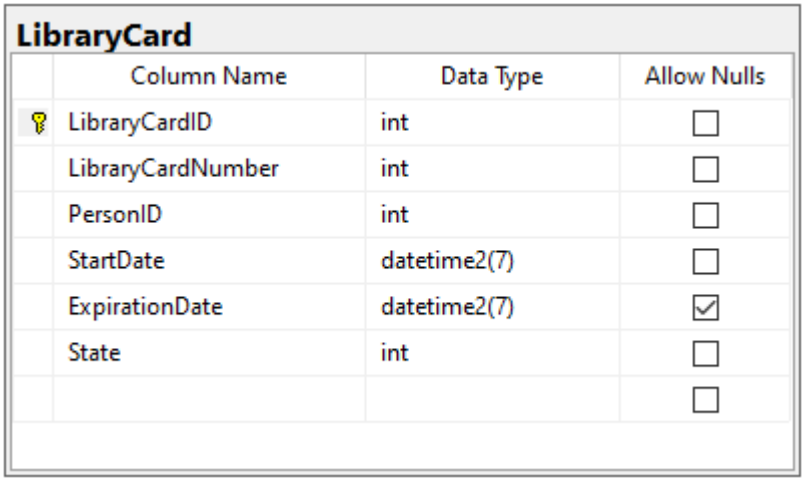
	Column Name	Data Type	Allow Nulls
🔑	PersonID	int	☐
	LastName	nvarchar(MAX)	☐
	FirstName	nvarchar(MAX)	☐
	Email	nvarchar(MAX)	☑
	Gender	int	☑
	DateOfBirth	datetime2(7)	☑
			☐

Rysunek 3 Tabela Person przechowywująca dane czytelników

Informacje o zarejestrowanych czytelnikach są przechowywane w tabeli *Person*. Typy danych w tej tabeli przedstawia rysunek 3. Dane takie jak nazwisko oraz imię są obowiązkowe.

Pozostałe dane (adres e-mail, płeć oraz data urodzenia) nie muszą być podawane, pola formularza mogą pozostać puste.

Tabelą bezpośrednio powiązaną z czytelnikiem jest zbiór informacji dotyczący kart bibliotecznych. Strukturę tej tabeli przedstawia poniższy rysunek 4.



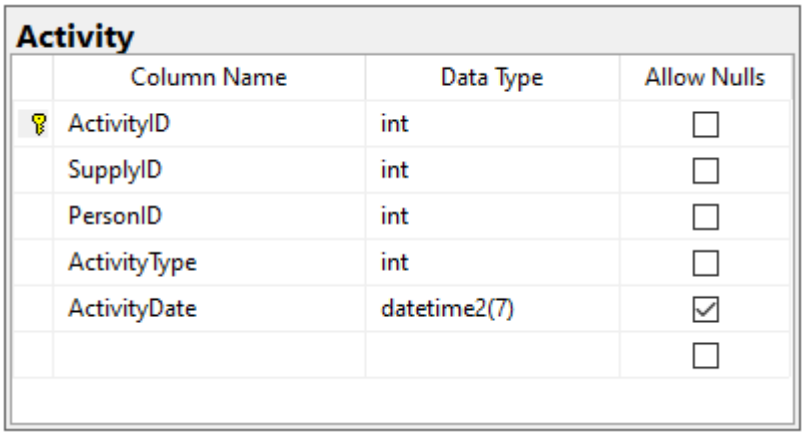
LibraryCard

Column Name	Data Type	Allow Nulls
LibraryCardID	int	☐
LibraryCardNumber	int	☐
PersonID	int	☐
StartDate	datetime2(7)	☐
ExpirationDate	datetime2(7)	☑
State	int	☐
		☐

Rysunek 4 Tabela Library przechowywująca dane kart bibliotecznych

Wszystkie dane w tej tabeli są obowiązkowe z wyjątkiem pola data ważności. Taka konfiguracja została stworzona celowo, aby umożliwić rejestrowanie kart ważnych bezterminowo. W razie potrzeby można taką kartę uzupełnić o wpis określający datę ważności luz zablokować kartę zmieniając status na nieaktywny.

Ostatnią tabelą jest zbiór informacji o wypożyczeniu zasobu biblioteki czyli tabela *Activity*. Typy danych w tej tabeli przedstawia rysunek 5.



Activity

Column Name	Data Type	Allow Nulls
ActivityID	int	☐
SupplyID	int	☐
PersonID	int	☐
ActivityType	int	☐
ActivityDate	datetime2(7)	☑
		☐

Rysunek 5 Tabela Activity łączy dane z pozostałych tabel i przechowuje informacje o wypożyczeniach

W tabeli Activity znajdują się dwa klucze obce. Taka konfiguracja agreguje w sobie dane o wypożyczeniach czyli:

- kto wypożyczył;
- co wypożyczył;
- kiedy;
- rodzaj aktywności (wypożyczenie, zwrot, zagubienie).

Istotną rzeczą jest to, że podczas rejestrowania aktywności nie ma możliwości podania daty. Jest to zabieg celowy. Data zdarzenia jest pobierana z systemu i nie ma możliwości jej zmiany. Jest to zrobione w celu uniknięcia pomyłek lub próby zafałszowania informacji.

Inicjowanie danych

Podczas uruchomienia aplikacji uruchamiana jest statyczna metoda:

```
DbInitializer.Initialize(context);
```

Metoda te sprawdza czy tabela *Persons* zawiera jakieś dane czytelników. Jeżeli w bazie danych nie ma żadnych rekordów baza danych jest wypełniana przykładowymi danymi.

W celu wypełnienia bazy danych przykładowymi danymi w metodzie `Initialize` tworzone są listy obiektów, które bez użycia zapytań SQL zostają zapisane do bazy danych. Taki zabieg jest możliwy dzięki frameworkowi ORM *Entity Framework Core*. Narzędzie to przekształca klasy na tabele w bazie danych, a obiekty tych klas na dane, którymi są wypełniane tabele.

Konfiguracja tego mechanizmu znajduje się w katalogu Data w klasie `LibraryContex` która dziedziczy z klasy `DbContext`. Zawartość tej klasy została przedstawiona na rysunku 6.

Korzystanie z takiego rozwiązania jest bardzo pomocne z kilku względów:

- nie jest wymagana głęboka wiedza dotycząca baz danych i języka SQL;
- pozwala skupić się programiście na pisaniu kodu aplikacji;
- umożliwia tworzenie bazy danych podczas uruchomienia aplikacji;

```
15 references | Piotr Antonczak, 1 day ago | 1 author, 1 change
public DbSet<Person> Persons { get; set; }
12 references | Piotr Antonczak, 1 day ago | 1 author, 1 change
public DbSet<Supply> Supplies { get; set; }
9 references | Piotr Antonczak, 1 day ago | 1 author, 1 change
public DbSet<LibraryCard> LibraryCards { get; set; }
3 references | Piotr Antonczak, 1 day ago | 1 author, 1 change
public DbSet<Activity> Activities { get; set; }

0 references | Piotr Antonczak, 1 day ago | 1 author, 1 change
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Person>().ToTable("Person");
    modelBuilder.Entity<Supply>().ToTable("Supply");
    modelBuilder.Entity<LibraryCard>().ToTable("LibraryCard");
    modelBuilder.Entity<Activity>().ToTable("Activity");
}
```

Rysunek 6 Konfiguracja DbSet czyli klasy zamapowane do bazy danych oraz metoda OnModelCreating

Konteneryzacja Docker

Aplikacja została tak skonfigurowana, aby można było umieścić ja w kontenerze Docker. Zabieg taki jest możliwy dzięki skryptowi, który został umieszczony w katalogu głównym aplikacji. Skrypt ten nazywa się: `Dockerfile` Jego treść została pokazana na poniższym rysunku 7.

Dockerfile

```
1
2  FROM mcr.microsoft.com/dotnet/aspnet:5.0 AS base
3  WORKDIR /app
4  EXPOSE 80
5  EXPOSE 443
6
7  FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build
8  WORKDIR /src
9  COPY ["MultimediaLibrary/MultimediaLibrary.csproj", "MultimediaLibrary/"]
10 RUN dotnet restore "MultimediaLibrary/MultimediaLibrary.csproj"
11 COPY . .
12 WORKDIR "/src/MultimediaLibrary"
13 RUN dotnet build "MultimediaLibrary.csproj" -c Release -o /app/build
14
15 FROM build AS publish
16 RUN dotnet publish "MultimediaLibrary.csproj" -c Release -o /app/publish
17
18 FROM base AS final
19 WORKDIR /app
20 COPY --from=publish /app/publish .
21 ENTRYPOINT ["dotnet", "MultimediaLibrary.dll"]
22
```

Rysunek 7 Listing skryptu Dockerfile do umieszczenia aplikacji w kontenerze Docker

Z uwagi na to, że system konteneryzacji nie obsługuje danych umieszczonych w bazie danych *LocalDB* do poprawnego działania aplikacji jest wymagane aby bazę danych umieścić na serwerze *Microsoft SQL Server* i odpowiednio zmodyfikować *ConnectionStrings.*

```
"Docker": {
  "commandName": "Docker",
  "launchBrowser": true,
  "launchUrl": "{Scheme}://{ServiceHost}:{ServicePort}",
  "publishAllPorts": true,
  "useSSL": true
}
```

Rysunek 8 Konfiguracja Docker w pliku launchSettings.json

W projekcie umieszczono konfigurację (rysunek 8), która umożliwia uruchomienie kontenera bezpośrednio z Visual Studio. Plik *launchSettings.json* w katalogu *Properties.*

Repozytorium GIT

Cały proces budowy aplikacji został zarejestrowany w repozytorium GIT. Projekt jest dostępny jako publiczne repozytorium na stronie *github.com*. Link do repozytorium to:

```
https://github.com/AntonczakPiotr/MultimediaLibrary.git
```

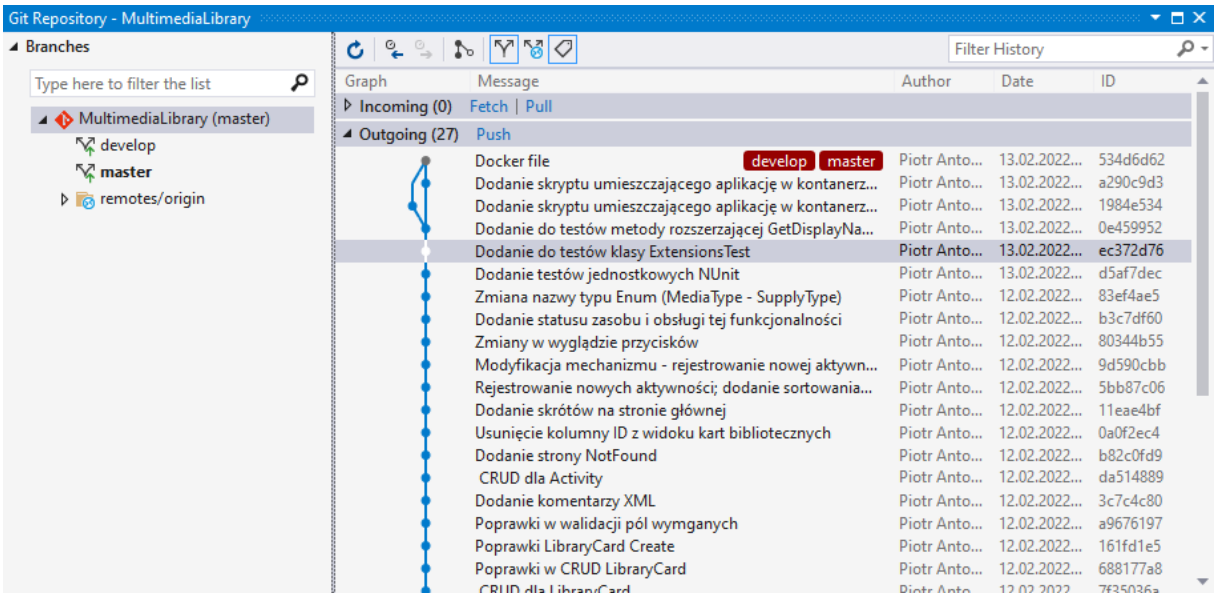
W celu pobrania repozytorium należy w dowolnym, nowo utworzonym i pustym katalogu wywołać polecenie:

```
git clone
https://github.com/AntonczakPiotr/MultimediaLibrary.gitAnt
onczakPiotr/MultimediaLibrary
```

Oczywiście aby można było skorzystać z poleceń GIT należy pobrać i zainstalować oprogramowanie GIT. Można to zrobić za darmo np. ze strony:

```
https://git-scm.com/downloads
```

Bardzo wygodnym rozwiązaniem jest korzystanie z obsługi GIT jaką daje Visual Studio 2019. Zmiany i ich wizualizacja w ramach niniejszego projektu, jakie zostały wprowadzone w aplikacji zostały przedstawione na rysunku 9.



Rysunek 9 Zmiany w aplikacji zarejestrowane w repozytorium GIT

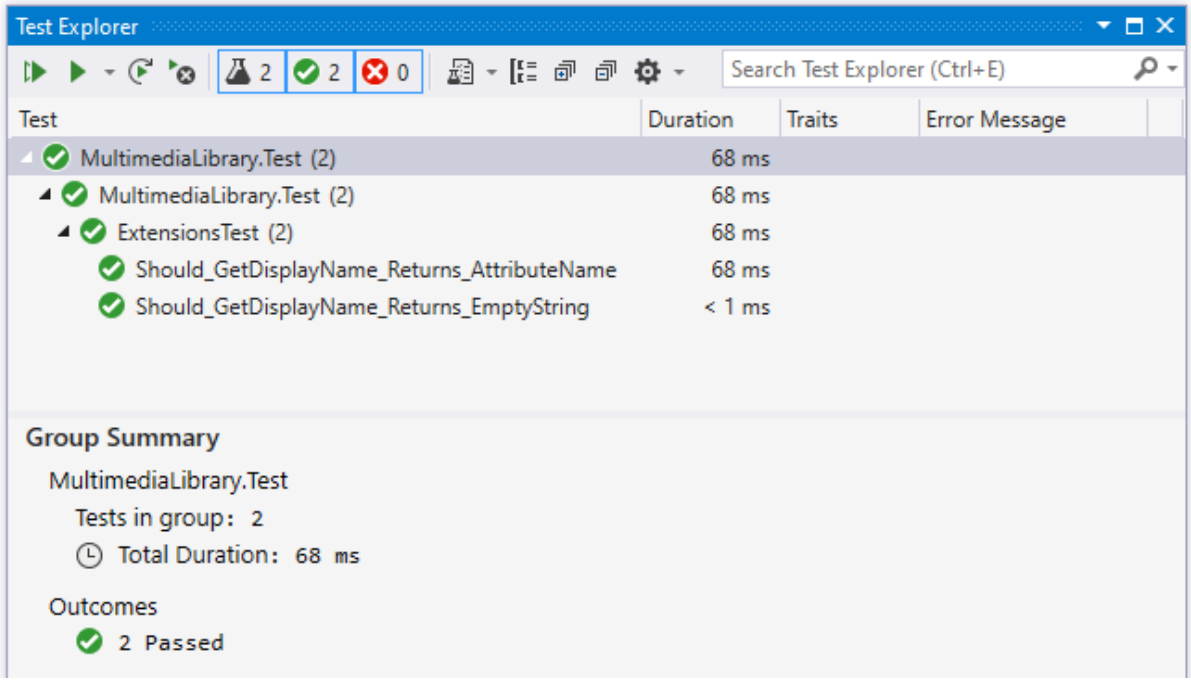
Testy jednostkowe

Dobrą praktyką programisty jest pokrywanie pisanego kodu testami jednostkowymi. Przy budowie aplikacji „Biblioteki Multimediów” do testów jednostkowych została wykorzystana biblioteka NUnit w wersji 3.13.1.

Testy jednostkowe zostały umieszczone w osobnym projekcie o nazwie:

```
MultimediaLibrary.Test
```

Wyniki testów jednostkowych przedstawia rysunek10.



Rysunek 10 Testy jednostkowe aplikacji z wykorzystaniem NUnit

NUnit jest frameworkiem open-source dla aplikacji pisanych w technologii .NET. Oferuje podobne mechanizmy jakie zostały zastosowane w JUnit - frameworku stworzonym do przeprowadzania testów jednostkowych dla programistów Java.

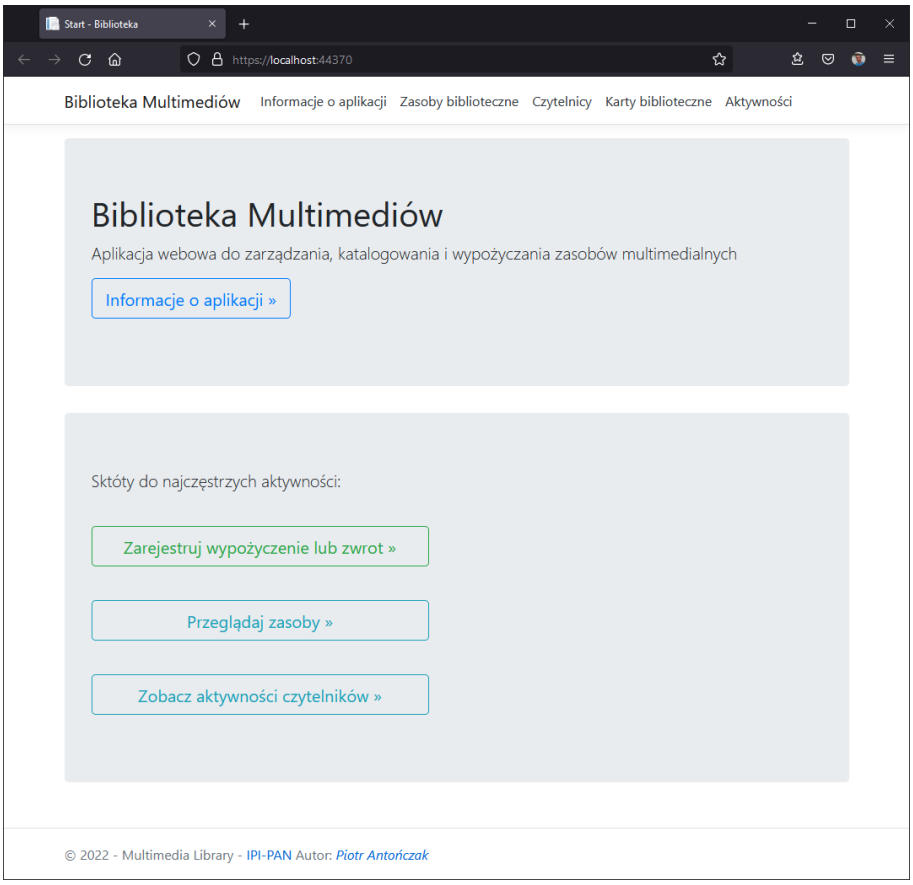
Główne cechy testów jednostkowych NUnit to:

- mogą być uruchamiane zarówno z konsoli jak i środowiska Visual Studio.
- testy można przeprowadzać równolegle.
- NUnit obsługuje wiele platform (np.: .NET Core, Xamarin, Silverlight).

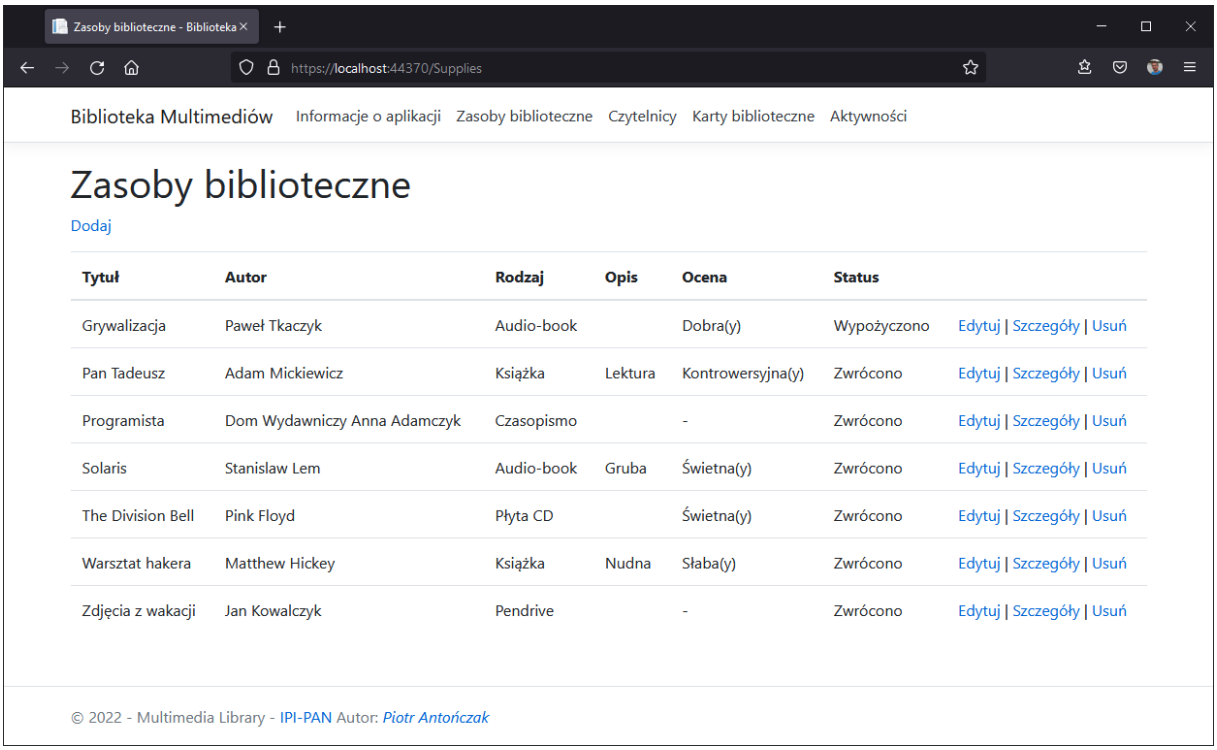
Prezentacja aplikacji

Aplikacja „Biblioteka Multimediów” składa się z czterech stron na których zostało zorganizowane zarządzanie zasobami biblioteki oraz dwóch stron informacyjnych. Każda z stron głównych zawiera kilka podstron służących do dodawania, edycji, usuwania i prezentacji obiektów których dotyczy. Poniżej przedstawiono listę stron aplikacji:

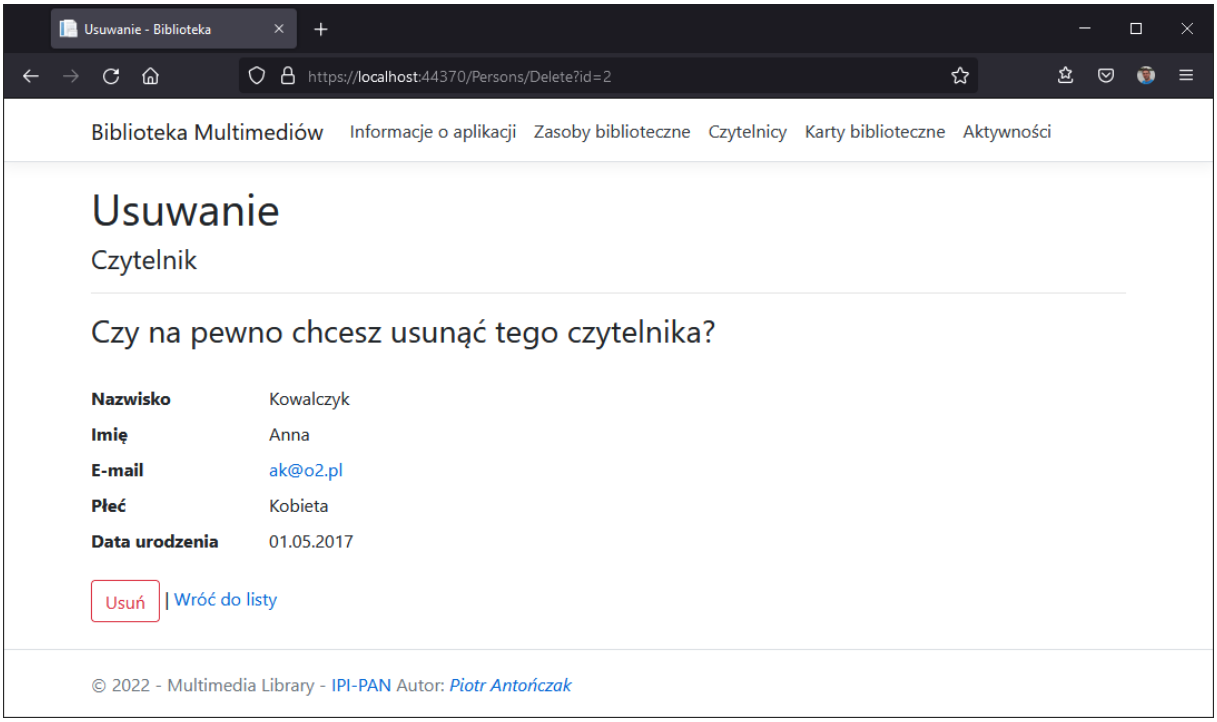
- **strona główna** – zawiera skróty do najważniejszych stron aplikacji – rysunek 11;
- **zasoby biblioteczne** – zawiera listę zasobów oraz podstrony do dodawania edycji, wyświetlania szczegółów i usuwania zasobów – rysunek 12 i 14;
- **czytelnicy** – zawiera listę czytelników oraz podstrony do dodawania edycji, wyświetlania szczegółów i usuwania czytelników – rysunek 13;
- **karty biblioteczne** – zawiera listę kart oraz podstrony do dodawania edycji, wyświetlania szczegółów i usuwania kart;
- **aktywności** – zawiera listę aktywności oraz przycisk do rejestrowania nowych aktywności – rysunek 15.



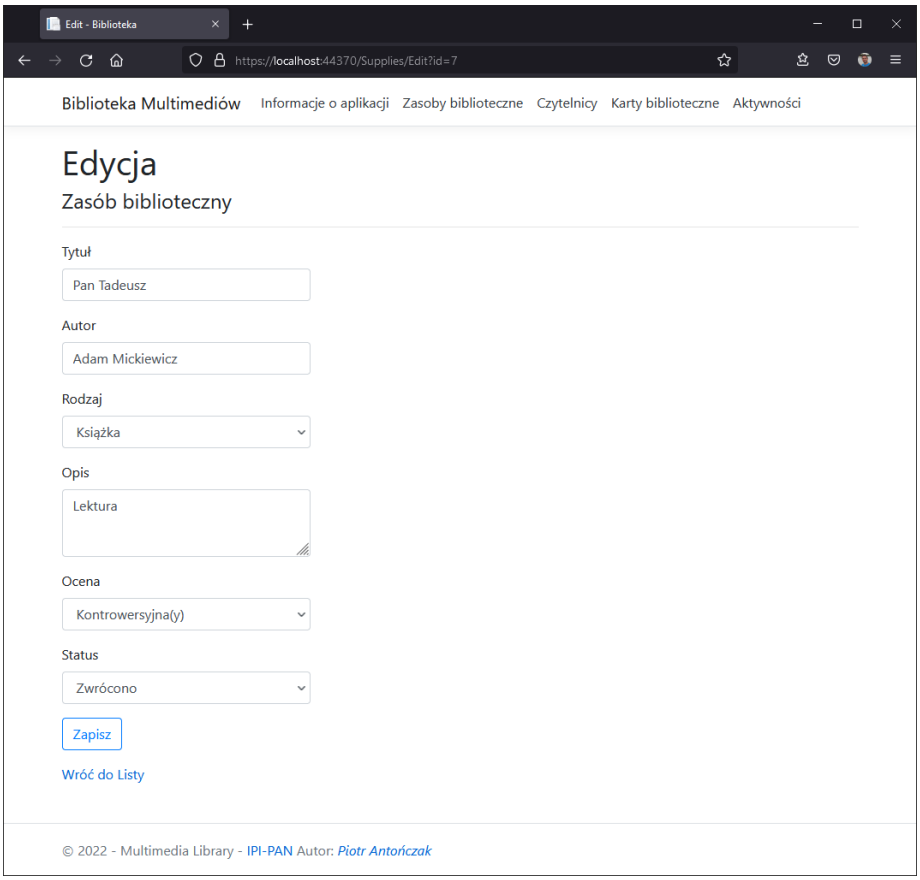
Rysunek 11 Strona główna aplikacji



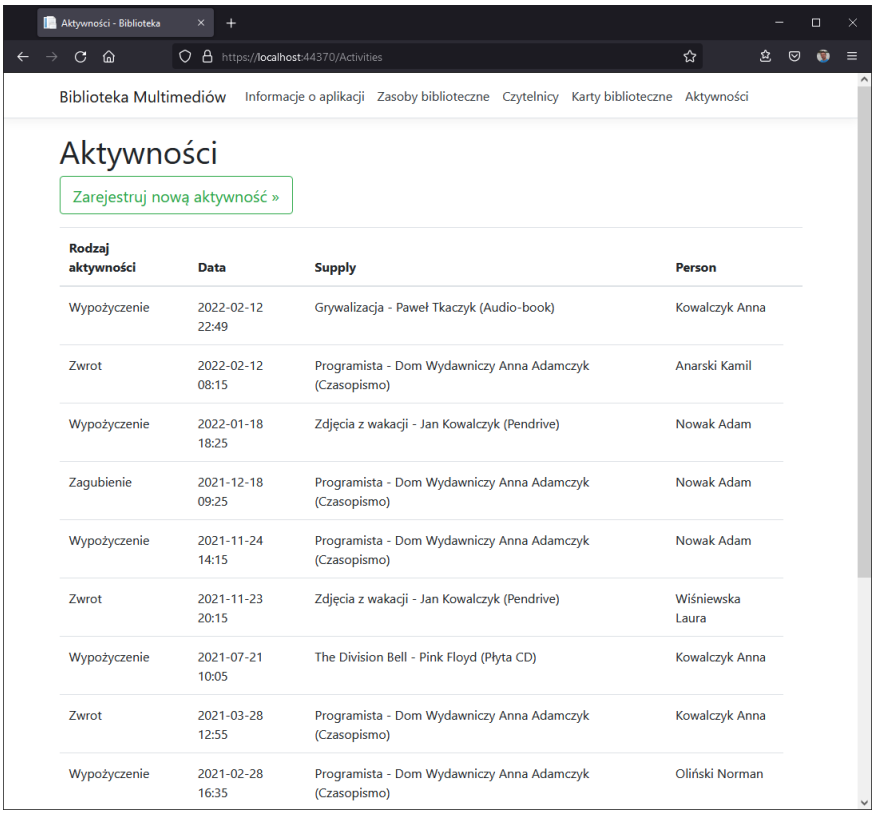
Rysunek 12 Lista zasobów biblioteki



Rysunek 13 Usuwanie czytelnika



Rysunek 14 Edycja zasobów bibliotecznych



Rysunek 15 Lista aktywności biblioteki

Podsumowanie

Aplikacja prezentowana w niniejszej pracy jest zbudowana z myślą o indywidualnym użytkowniku. Prezentuje możliwości narzędzi programistycznych. Została przygotowana zgodnie z aktualnymi możliwościami autora w szczególności ograniczonym czasem na realizację zadania. Bardzo użyteczną funkcjonalnością zastosowaną do budowy aplikacji jest Scaffolding. Jest on bardzo dynamicznie rozwijany i rozszerzany w kolejnych wersjach Visual Studio przez firmę Microsoft. Umożliwia on tworzenie elementów aplikacji, które można następnie poddać obróbce i w łatwy, a przede wszystkim bardzo szybki sposób rozwijać budowaną aplikację.

Kolejną rzeczą, w którą warto by wyposażyć aplikację „Biblioteka multimediów", a na co brakło czasu, jest mechanizm uwierzytelniania i organizacji uprawnień w aplikacji. Wspomniana wcześniej funkcjonalność Visual Studio – Scaffolding umożliwia budowanie mechanizmu autoryzacji co na pewno zostanie wykorzystane przy rozwijaniu budowanego oprogramowania.