Dokumentacja projektu WPFilmweb

Autorzy:
Marian Dorosz
Oskar Fojcik

26 czerwca 2022

Spis treści

1 Część I

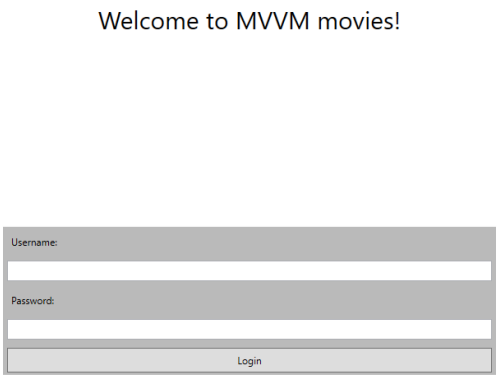
1.1 Opis programu

Aplikacja WPFilweb to program, w którym z bazy danych wyświetlane są filmy, aktorzy, reżyserzy oraz nagrody. Możemy oglądać szczegółowy opis wyżej wymienionych encji oraz je dodawać i usuwać.

1.2 Instrukcja obsługi

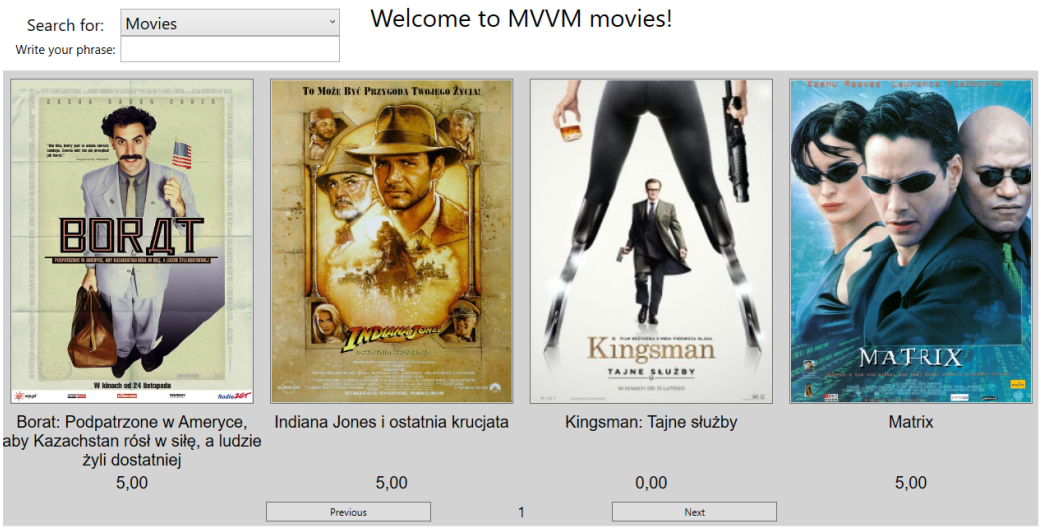
Po włączeniu aplikacji wyświetla się ekran logowania. Mamy 2 dostępne typy użytkowników:

- **Administrator** - login: admin hasło: 123
- **Użytkownik** - login: user hasło: 1234



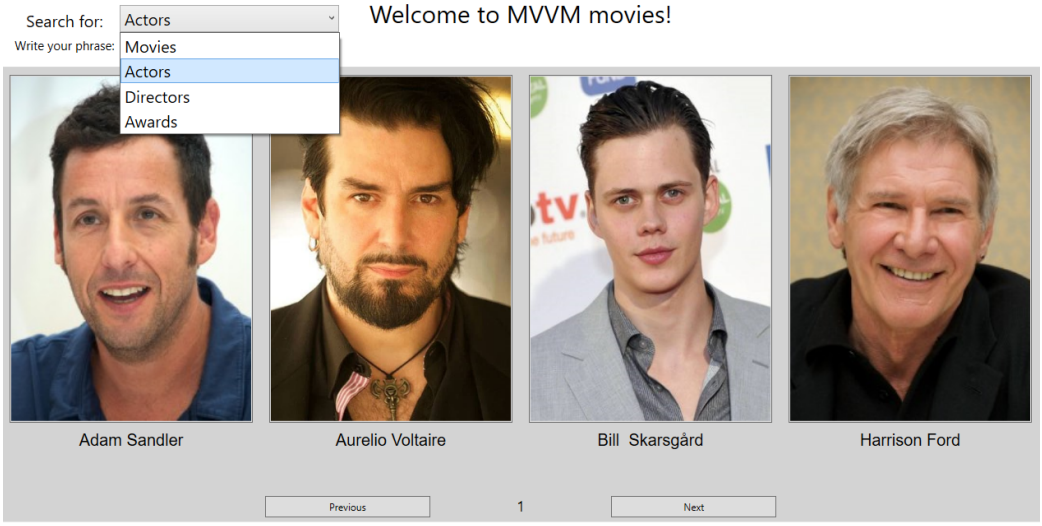
Rysunek 1: Ekran logowania

Po zalogowaniu na użytkownika możemy przeglądać filmy, aktorów, reżyserów i nagrody. Mamy dostęp do wyświetlania informacji o wcześniej wspomnianych encji.



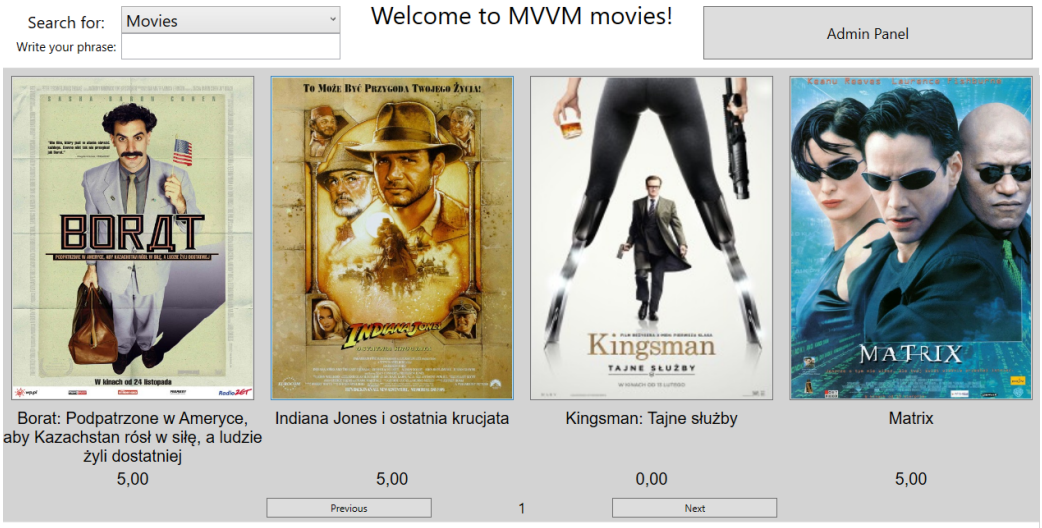
Rysunek 2: Widok po zalogowaniu na konto użytkownika

Widok interesującej nas encji zmieniamy w comboboxie w lewym górnym rogu aplikacji



Możemy też wyszukiwać encje po nazwie przy pomocy wyszukiwarki umieszczonej pod comboboxem

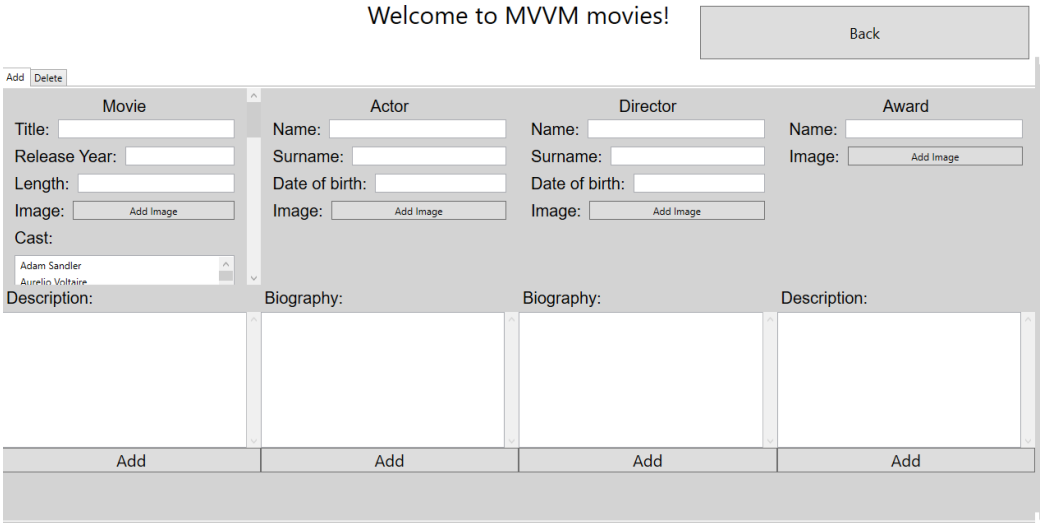
Po zalogowaniu na administratora mamy dostęp do wszystkich funkcjonalności użytkownika. Różnicą jest pojawienie się w prawym górnym rogu aplikacji przycisku Admin Panel.



Rysunek 3: Widok po zalogowaniu na konto administratora

Po wejściu w Admin Panel przechodzimy do widoku, w którym mamy dwie zakładki:

- **Add** - możemy w niej dodawać filmy, aktorów, reżyserów oraz nagrody
- **Delete** - możemy w niej usuwać filmy, aktorów, reżyserów oraz nagrody



Rysunek 4: Zakładka Add

Rysunek 5: Zakładka Delete

2 Część II

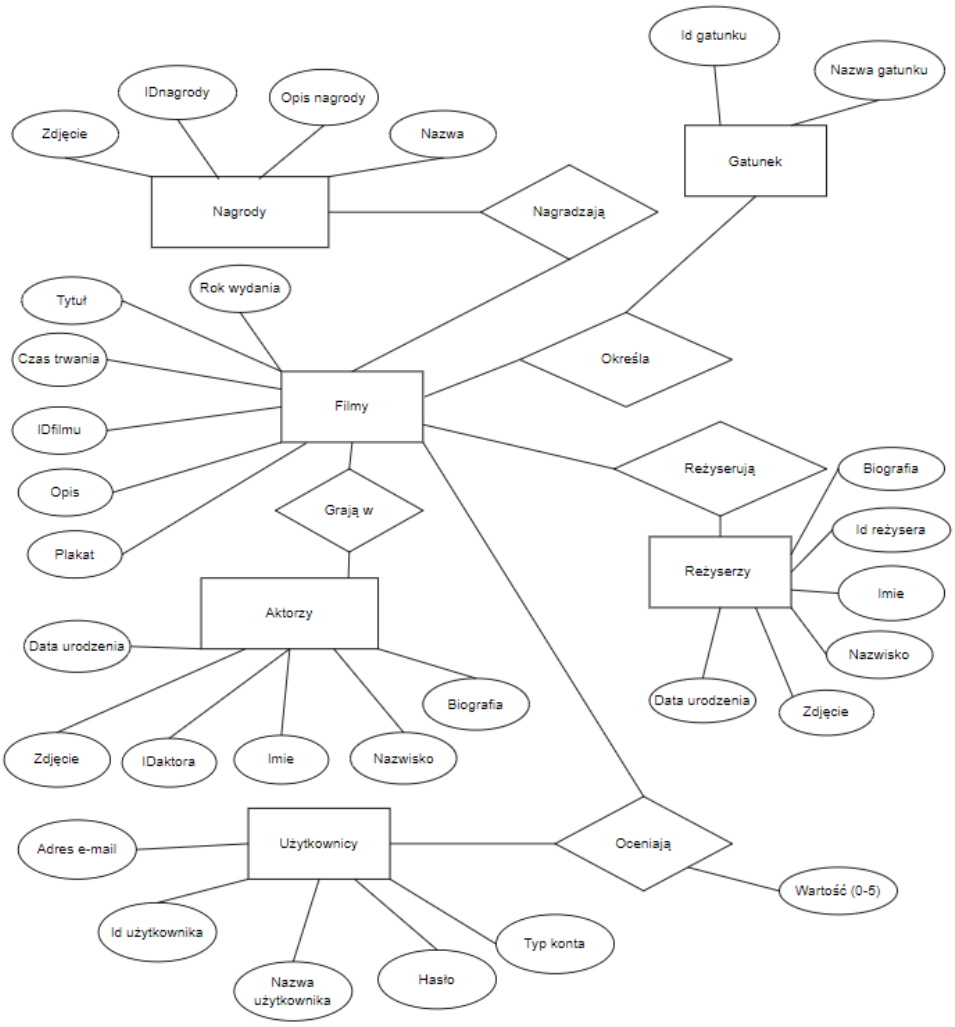
2.1 Baza danych

WPFilmweb posiada własną bazę danych, która przechowuje informacje o filmach dodanych do bazy. Aby móc opisać film maksymalnie dokładnie baza posiada następujące typy encji:

- FILMY (n:n) czas trwania, tytuł, rok wydania, opis, plakat
- REŻYSERZY imię, nazwisko, data urodzenia, biografia, zdjęcie
- AKTORZY imię, nazwisko, data urodzenia, biografia, zdjęcie
- GATUNEK nazwa gatunku
- NAGRODY opis nagrody, nazwa, zdjęcie
- UŻYTKOWNICY nazwa użytkownika, hasło, adres e-mail, typ konta

oraz następujące związki między encjami:

- Reżyserują: REŻYSERZY Reżyserują FILMY (n:n) - Wiele reżyserów może reżyserować jeden film; wiele filmów może być reżyserowanych przez jednego reżysera
- Grają w: AKTORZY Grają w FILMY (n:m) - Wiele aktorów może grać w jednym filmie; aktor może grać w wielu filmach
- Określa: GATUNEK Określa FILMY (n:m) - Wiele gatunków może określać jeden film; gatunek może określać wile filmów
- Nagradzają: NAGRODY Nagradzają FILMY (n:m) - Wiele nagród może nagrodzić jeden film; jedna nagroda może zostać przyznana jednemu filmowi
- Oceniają: UŻYTKOWNICY Oceniają FILMY (n:m) - Wielu użytkowników może ocenić jeden film; użytkownik może ocenić wiele filmów



Rysunek 6: Diagram związków encji dla bazy filmów

2.2 Aplikacja

2.2.1 Model aplikacji

Aplikacja WPFilmweb została zbudowana w architekturze trójwarstwowej zgodnie z wzorcem projektowym MVVM. Dodatkowo kod aplikacji został rozbudowany o wartswę dostępu do bazy, która jest singletonem pozwalającym na pobieranie danych z bazy serwera MySQL.

2.2.2 Model

Model składa się z dwóch klas, które są singletonami - NavigationModel oraz Model. Funkcją modelu nawigacji jest umożliwienie płynnej zmiany widoku np.: w przypadku naciśnięcia przycisku, natomiast Model odpowiada za wszystkie pozostałe funkcjonalności takie jak odświeżanie listy filmów itp.

2.3 DAL (Data Access Layer)

Warstwa odpowiedzialna za dostęp do bazy, zawiera klasę DBConnection.

```
1  class DBConnection
2  {
3  private MySqlConnectionStringBuilder stringBuilder = new
       MySqlConnectionStringBuilder();
4
5  private static DBConnection instance = null;
6  public static DBConnection Instance
7  {
8      get
9      {
10         if (instance == null)
11             instance = new DBConnection();
12         return instance;
13     }
14 }
15
16 public MySqlConnection Connection => new MySqlConnection(
       stringBuilder.ToString());
17
18
19 private DBConnection()
20 {
21     // creating connection string with data stored in
          settings.settings
22     stringBuilder.UserID = Properties.Settings.Default.
          userID;
23     stringBuilder.Server = Properties.Settings.Default.
          server;
24     stringBuilder.Database = Properties.Settings.Default.
          database;
25     stringBuilder.Port = Properties.Settings.Default.port;
26     stringBuilder.Password = Properties.Settings.Default.
          paswd;
27 }
28 }
```

2.3.1 Encje

Klasy reprezentujące encje bazy danych. Przykład encji Filmy:

```
1  public class Filmy
2  {
3      #region Properties
4
5      public int IDmovie { get; set; }
6      public string Title { get; set; }
7      public int ReleaseYear { get; set; }
8      public string Length { get; set; }
9      public string Description { get; set; }
10     public byte[] Poster { get; set; }
11
12     #endregion
13
14     #region Constructors
15     // MySqlDataReader constructor - creates object based
           on MySql data
16     public Filmy(MySqlDataReader reader)
17     {
18         IDmovie = int.Parse(reader["IDfilmu"].ToString());
19         Title = reader["tytul"].ToString();
20         ReleaseYear = int.Parse(reader["rok_wydania"].
               ToString());
21         Length = reader["czas_trwania"].ToString();
22         Description = reader["opis"].ToString();
23         if (reader["plakat"] == DBNull.Value)
24             Poster = null;
25         else
26             Poster = (byte[])reader["plakat"];
27     }
28     // New object ctcreated from scratch, to add into
           database
29     public Filmy(string title, int releaseYear, string
           length, string description, byte[] poster)
30     {
31         //IDmovie = null;
32         Title = title;
33         ReleaseYear = releaseYear;
34         Length = length;
35         Description = description;
36         Poster = poster;
37     }
38
39     public Filmy(string title, int releaseYear, string
           length, string description)
40     {
41         //IDmovie = null;
```

```
42          Title = title;
43          ReleaseYear = releaseYear;
44          Length = length;
45          Description = description;
46          Poster = null;
47      }
48
49      //Copty ctor
50      public Filmy(Filmy movie)
51      {
52          IDmovie = movie.IDmovie;
53          Title = movie.Title;
54          ReleaseYear = movie.ReleaseYear;
55          Length = movie.Length;
56          Description = movie.Description;
57          Poster = movie.Poster;
58      }
59      #endregion
60
61      #region Methods
62
63      // Override Equals method to check if actor is not
            duplicated with Contains function
64      public override bool Equals(object obj)
65      {
66          var movie = obj as Filmy;
67          if (movie == null) return false;
68          if (Title.ToLower() != movie.Title.ToLower())
                return false;
69          if(ReleaseYear != movie.ReleaseYear) return false;
70          if(Length != movie.Length) return false;
71          if(Description.ToLower() != movie.Description.
                ToLower()) return false;
72          return true;
73      }
74      public override int GetHashCode()
75      {
76          return base.GetHashCode();
77      }
78      #endregion
79  }
```

2.3.2 Repozytoria

Klasy stworzone jako repozytoria pozwalające na dostęp do danych. Przykład RepozytoriumFilmy:

```
1  class RepozytoriumFilmy
```

```
2 {
3     private const string GET_ALL_MOVIES = "SELECT * FROM
          filmy";
4     private const string ADD_MOVIE = "INSERT INTO filmy(
          tytul,rok_wydania,czas_trwania,opis,plakat) VALUES (
          @tyt, @rok, @czas, @op, @pla)";
5     public static List<Filmy> GetMovies()
6     {
7         List<Filmy> movies = new List<Filmy>();
8         using (var connection = DBConnection.Instance.
              Connection)
9         {
10            MySqlCommand command = new MySqlCommand(
                  GET_ALL_MOVIES, connection);
11            connection.Open();
12            var reader = command.ExecuteReader();
13            while (reader.Read())
14                movies.Add(new Filmy(reader));
15            connection.Close();
16        }
17        return movies;
18    }
19    public static bool AddMovie(Filmy movie)
20    {
21        bool state = false;
22        using (var connection = DBConnection.Instance.
              Connection)
23        {
24            MySqlCommand command = new MySqlCommand($"{
                  ADD_MOVIE}", connection);
25            command.Parameters.Add("@tyt", MySqlDbType.
                  VarChar);
26            command.Parameters.Add("@rok", MySqlDbType.
                  Int64);
27            command.Parameters.Add("@czas", MySqlDbType.
                  Time);
28            command.Parameters.Add("@op", MySqlDbType.
                  VarChar);
29            command.Parameters.Add("@pla", MySqlDbType.Blob
                  );
30            command.Parameters["@tyt"].Value = movie.Title;
31            command.Parameters["@rok"].Value = movie.
                  ReleaseYear;
32            command.Parameters["@czas"].Value = TimeSpan.
                  Parse(movie.Length);
33            command.Parameters["@op"].Value = movie.
                  Description;
34            command.Parameters["@pla"].Value = movie.Poster
                  ;
```

```
35              connection.Open();
36              Console.WriteLine(command.CommandText);
37              var id = command.ExecuteNonQuery();
38              state = true;
39              movie.IDmovie = (int)command.LastInsertedId;
40              connection.Close();
41          }
42          return state;
43      }
44
45      public static bool EditMovie(Filmy movie, int IDmovie)
46      {
47          bool state = false;
48          using (var connection = DBConnection.Instance.
                Connection)
49          {
50              string EDIT_MOVIE = $"UPDATE filmy SET tytul={
                    movie.Title}, rok_wydania='{movie.
                    ReleaseYear}', " +
51                  $"czas_trwania='{movie.Length}', opis = '{
                        movie.Description}', plakat='{movie.
                        Poster}' WHERE IDfilmu = {IDmovie}";
52              MySqlCommand command= new MySqlCommand(
                    EDIT_MOVIE, connection);
53              connection.Open();
54              var id = command.ExecuteNonQuery();
55              if (id == 1)
56                  state = true;
57              connection.Close();
58          }
59          return state;
60      }
61      public static bool DeleteMovie(Filmy movie)
62      {
63          bool state = false;
64          using (var connection = DBConnection.Instance.
                Connection)
65          {
66              string DELETE_MOVIE = $"DELETE FROM filmy WHERE
                     filmy.tytul = '{movie.Title}'";
67              MySqlCommand command= new MySqlCommand(
                    DELETE_MOVIE, connection);
68              connection.Open();
69              var id = command.ExecuteNonQuery();
70              if (id == 1)
71                  state = true;
72
73              connection.Close();
74          }
```

```
75          return state;
76      }
77
78  }
```

2.3.3 ViewModel

View model składa się z kilku różnych klas, które dziedziczą po klasie ViewModelBase. Dodatkowo jest w nim zawarta klasa RelayCommand, która jest interfejsem dla ICommand, przy pomocy którego zostały utworzone specjalne wydarzenia powiązane m.in. z funkcjonalnością przycisków.

Kod klasy ViewModelBase:

```
1  namespace WPFilmweb.ViewModel.BaseClasses
2  {
3      public class ViewModelBase : INotifyPropertyChanged
4      {
5          public event PropertyChangedEventHandler
               PropertyChanged;
6
7          protected void onPropertyChanged(params string[]
               namesOfProperties)
8          {
9              if (PropertyChanged != null)
10             {
11                 foreach (var prop in namesOfProperties)
12                 {
13                     PropertyChanged(this, new
                           PropertyChangedEventArgs(prop));
14                 }
15             }
16         }
17     }
18 }
```

Kod klasy RelayCommand:

```
1  namespace WPFilmweb.ViewModel.BaseClasses
2  {
3      public class RelayCommand : ICommand
4      {
5
6          readonly Action<object> _execute;
7          readonly Predicate<object> _canExecute;
8
9          public RelayCommand(Action<object> execute,
               Predicate<object> canExecute)
```

```
10          {
11              if (execute == null)
12                  throw new ArgumentNullException(nameof(
                        execute));
13              else
14                  _execute = execute;
15              _canExecute = canExecute;
16          }
17
18          public bool CanExecute(object parameter)
19          {
20              return _canExecute == null ? true : _canExecute
                    (parameter);
21          }
22
23          public event EventHandler CanExecuteChanged
24          {
25              add
26              {
27                  if (_canExecute != null) CommandManager.
                        RequerySuggested += value;
28              }
29              remove
30              {
31                  if (_canExecute != null) CommandManager.
                        RequerySuggested -= value;
32              }
33          }
34
35          public void Execute(object parameter)
36          {
37              _execute(parameter);
38          }
39      }
40  }
```

2.3.4 View

Znajduje się tu główne okno MainWIndow.xaml który korzysta z własności ViewModels:MoviesViewModel oraz kontrolki użytkownika:

- ActorDescription - korzysta z własności ViewModels:ActorDescriptionViewModel
- Actors - korzysta z własności ViewModels:ActorsViewModel
- AdminPanel - korzysta z własności ViewModels:AdminPanelViewModel
- AwardDescription - korzysta z własności ViewModels:AwardDescriptionViewModel

- Awards - korzysta z własności ViewModels:AwardsViewModel
- DirectorDescription - korzysta z własności ViewModels:DirectorDescriptionViewModel
- Directors - korzysta z własności ViewModels:DirectorsViewModel
- MovieDescription - korzysta z własności ViewModels:MovieDescriptionViewModel
- Movies - korzysta z własności ViewModels:MoviesViewModel

3 Podsumowanie

Aplikacja WPFilmweb zawiera wszystkie funkcjonalnośći, które zostały dla niej zaplanowane. Można przy jej pomocy poszukiwać informacje o aktorach, filmach, nagrodach oraz reżyserach. Dodatkowo użytkownicy otrzymali możliwość oceny filmu. Interfejs graficzny aplikacji jest na tyle prosty, że każdy może skorzystać z bazy i wyszukać film. Można więc powiedzieć, że funkcjonalności, które miała mieć aplikacja zostały zaimplementowane.

4 Repozytorium Github

`https://github.com/oskif/WPFilmwebMVVM`