Національний університет “Львівська політехніка”

Кафедра програмного забезпечення

КУРСОВА РОБОТА

з дисципліни «Об’єктно-орієнтоване програмування»

На тему:
«*Додаток менеджменту списку фільмів*»

Студента групи *ПЗ-25*

спеціальності 6.121

“Інженерія програмного забезпечення”

Сьомко П.Я.

Керівник: доцент кафедри ПЗ,

к.т.н., доцент Коротєєва Т. О.

Національна шкала _____

Кількість балів___Оцінка ECTS___

Члени комісії _____ _____

_____ _____

_____ _____

Львів 2021

Зміст

Календар

№ з/п	Зміст завдання	Дата
1	Здійснити аналітичний огляд літератури за заданою темою та обгрунтувати вибір інструментальних засобів реалізації.	12.09.2021
2	Побудова UML діаграм	15.09.2021
3	Розробка алгоритмів реалізації	17.09.2021
4	Реалізація завдання (кодування)	21.09.2021
5	Формування інструкції користувача	01.10.2021
6	Оформлення звіту до курсової роботи згідно з вимогами Міжнародних стандартів, дотримуючись такої структури: · зміст; · алгоритм розв‘язку задачі у покроковому представленні; · діаграми UML класів, прецедентів, послідовності виконання; · код розробленої програми з коментарями; · протокол роботи програми для кожного пункту завдання · інструкція користувача та системні вимоги; · опис виняткових ситуацій; · структура файлу вхідних даних; · висновки; · список використаних джерел.	05.10.2021

1. Технічне завдання

1. Розробити програму засобами ООП на мові C++/C#/Java згідно вказаного варіанту (Варіант №16).
2. Передбачити віконний режим роботи програми та інтерфейс користувача.
3. Передбачити ввід даних у двох режимах:
   - З клавіатури;
   - З файлу;
4. Передбачити у програмі виняткові ситуації.
5. Продемонструвати викладачу роботу розробленої програми.
6. Сформувати звіт курсової роботи обсягом не менше 20 сторінок.

Завдання варіанту №16

Створити таблицю

Назва фільму | Режисер | Рік випуску | Актори | Бюджет | Країна виробництва | Тривалість

1) Алгоритмом простої вибірки відсортувати записи за Країною виробництва.

2) Визначити Назви фільмів, в яких однакові режисери та найменші бюджети одночасно.

3) За заданою країною виробництва знайти всі фільми, в яких найбільші бюджети і найменша тривалість одночасно.

4) Для кожного Режисера визначити фільм з найбільшою тривалістю.

5) Знайти найдорожчий та найстарший фільм одночасно.

6) За заданим актором визначити всі фільми, в яких він (вона) знімались.

7) Встановити найбільш популярного актора.

Для класу створити: 1) Конструктор за замовчуванням; 2) Конструктор з параметрами; 3) конструктор копій; 4) перевизначити операції >>, << для зчитування та запису у файл. Для демонстрації роботи програми використати засоби візуального середовища програмування.

2. Алгоритм розв’язку задачі

2.1. Задача зчитування даних з файлу.

Алгоритм RF.

RF1. Виклик функції зчитування даних.

RF2. Користувач обирає шлях до файлу в файловому провіднику.

RF3. Перевірка чи шлях до файлу існує і файл не порожній. Якщо результат false, виклик відповідного повідомлення, переходимо до пункту RF7.

RF4. Відкриваємо файл.

RF5. Порядкове зчитування даних.

RF6. Закриття файлу.

RF7. Кінець.

2.2. Задача додавання елемента.

Алгоритм AN.

AN1. Виклик функції для додавання нового елемента.

AN2. Ввести дані.

AN3. Перевірка даних. Якщо дані не коректні, перейти до пункту AN5.

AN4. Додавання елемента в кінець списку.

AN5. Кінець.

2.3. Задача сортування списку.

Алгоритм SE.

SE1. Виклик функції сортування.

SE2. Вибір критерія сортування.

SE3. Відсортувати список відомим алгоритмом.

SE4. Кінець.

2.4. Задача знаходження акторів за певними критеріями.

Алгоритм FA.

FA1. Виклик функції для знаходження акторів.

FA2. Задати критерії пошуку.

FA3. Виділити список акторів зі списку фільмів.

FA4. Знаходження елементів, які відповідають заданим критеріям.

FA5. Повернути знайдені дані.

FA6. Кінець.

2.5. Задача запису у файл.

Алгоритм WF.

WF1. Виклик функції запису і вибір користувачем файлу.

WF2. Перевірка шляху до файлу. Якщо не існує, виклик відповідного повідомлення, перехід до пункту WF5.

WF3. Перевірка чи файл відкритий. Якщо ні, виклик відповідного повідомлення, перехід до пункту WF5.

WF4. Запис даних.

WF5. Закриття файлу.

WF6. Кінець.

3. Діаграми

3.1 Діаграма класів.

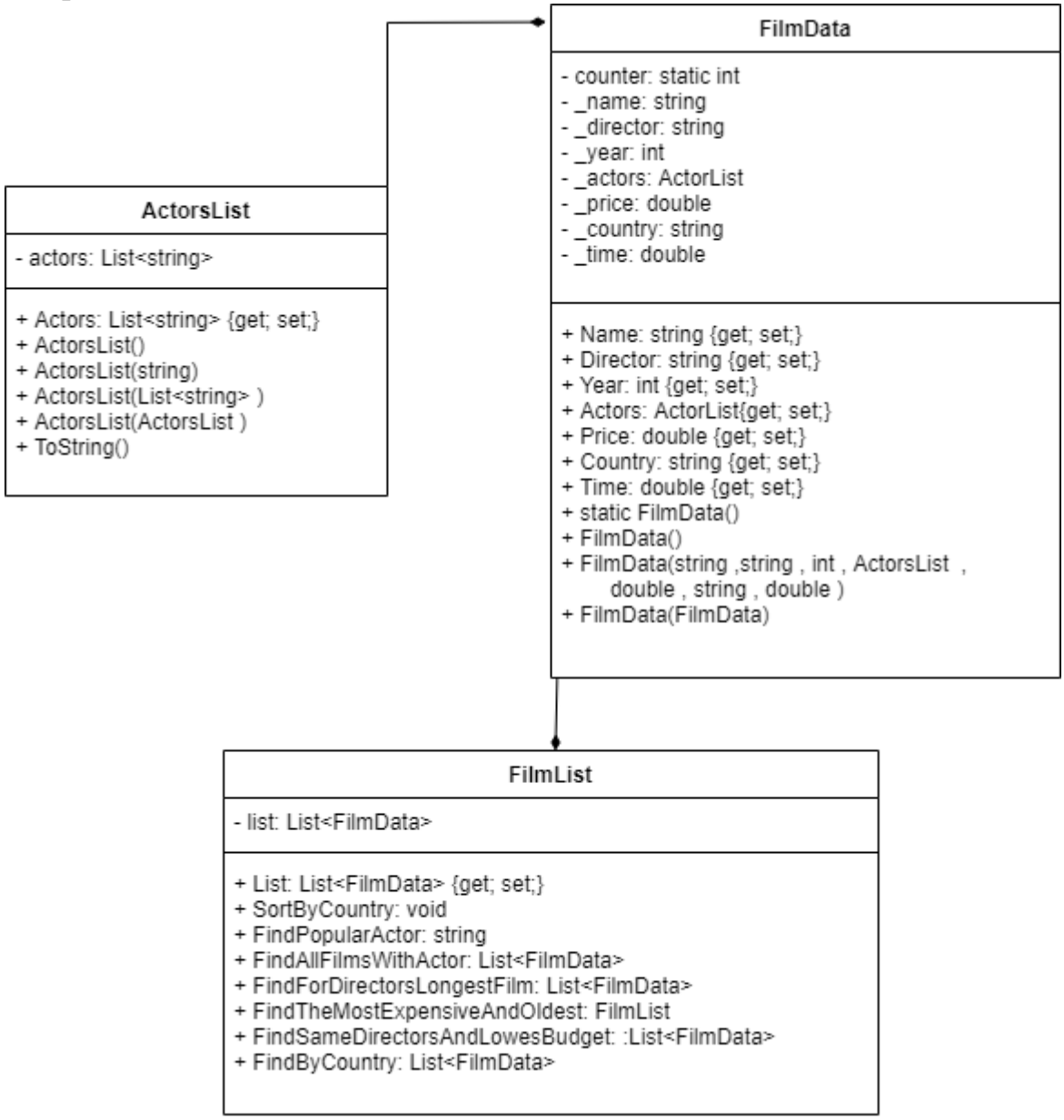

Рисунок 3.1 діаграма основних класів

3.2 Діаграма прецедентів

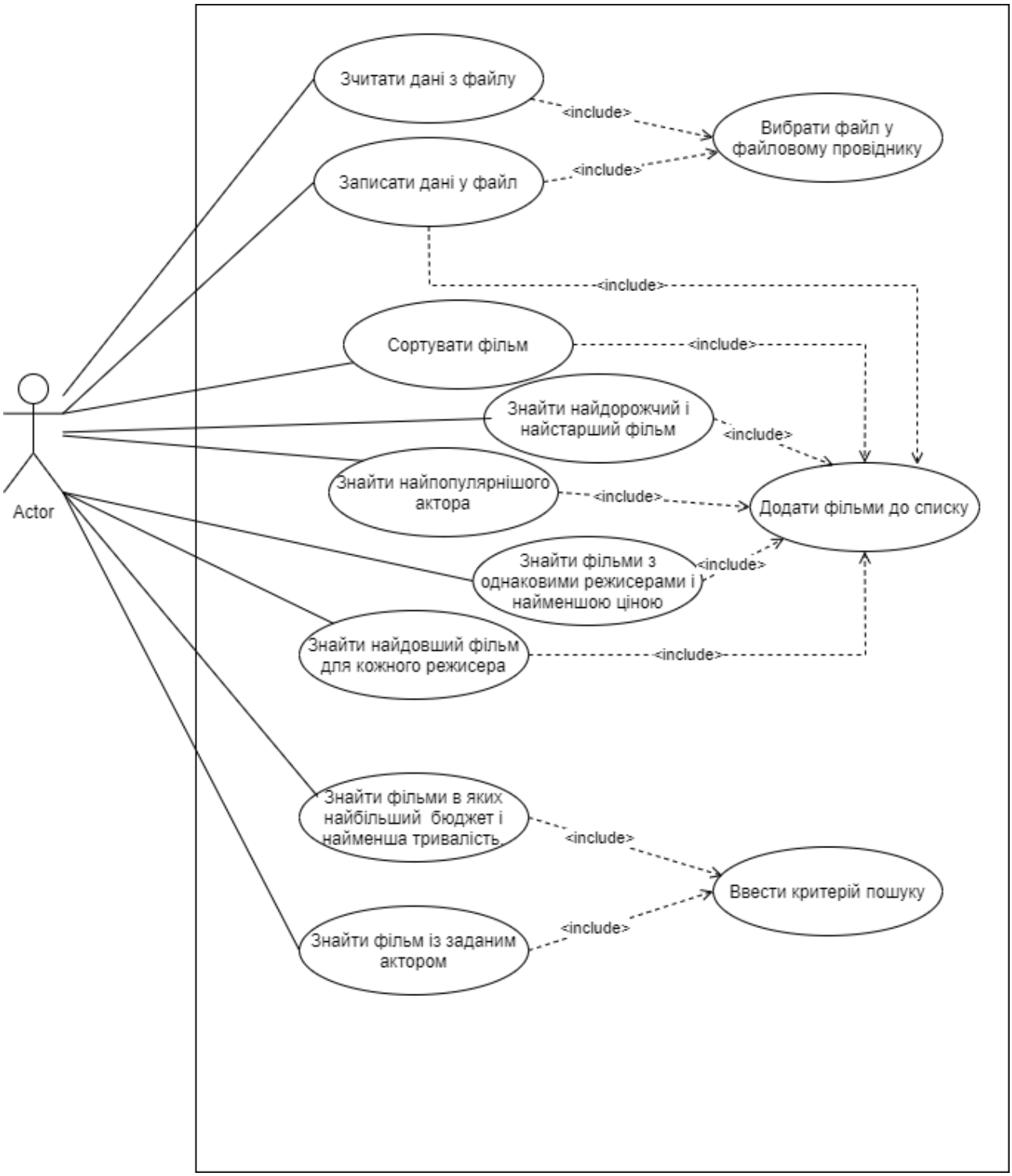

Рисунок 3.2 діаграма прецедентів

3.3 Діаграма послідовності

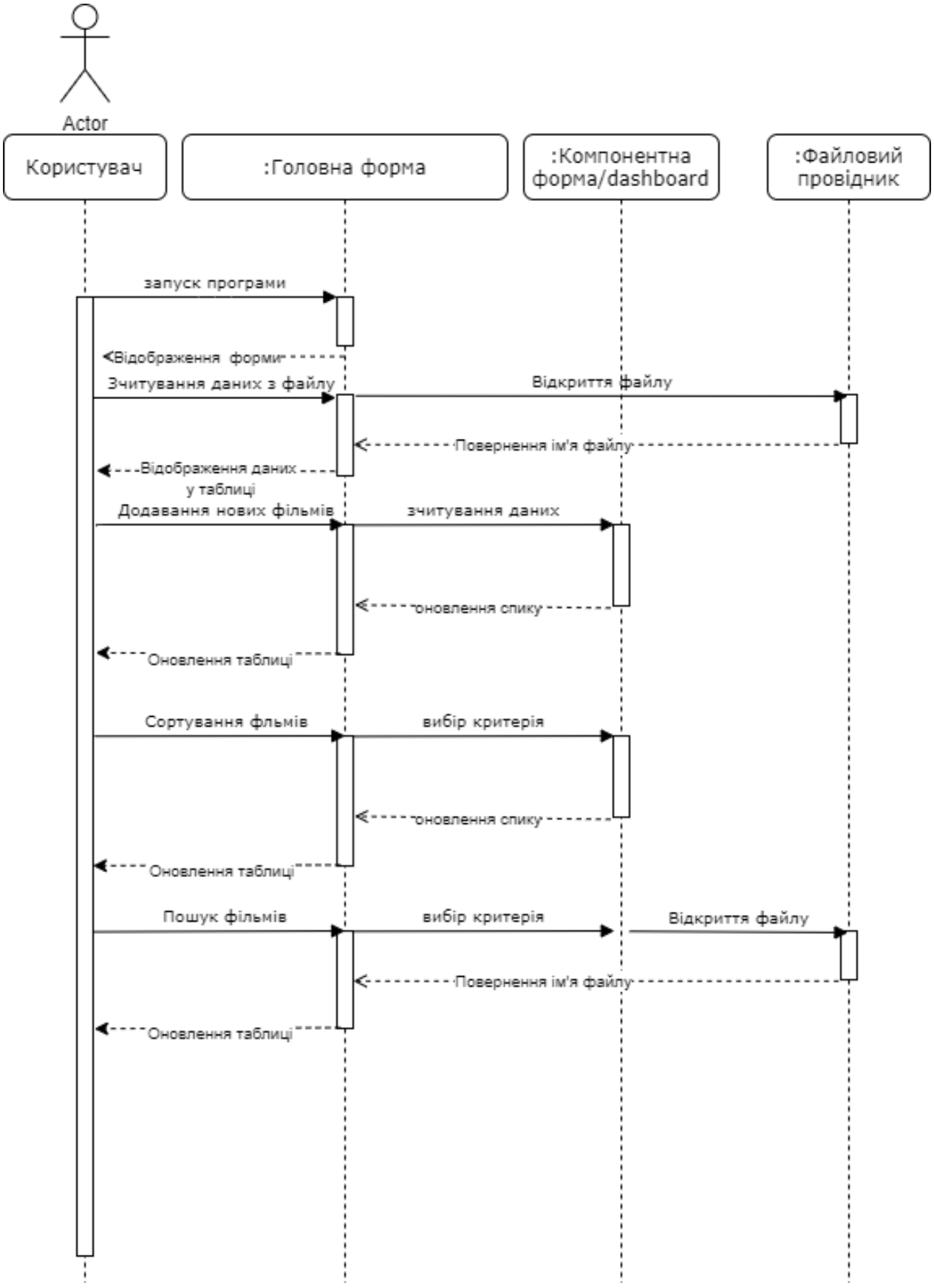

Рисунок 3.3 діаграма послідовності

4. Код програми.

4.1 FillmData.cs

```
using FilmApp.Model;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FilmApp
{
    public class FilmData
    {
        public static int counter;
        protected string _name;
        protected string _director;
        protected int _year;
        protected ActorsList _actors;
        protected double _price;
        protected string _country;
        protected double _time;

        public string Name { get { return _name; } set { _name = value; } }
        public string Director { get { return _director; } set { _director = value; } }
        public int Year { get { return _year; } set { _year = value; } }

        public ActorsList Actors { get { return _actors; } set { _actors = value; } }

        public double Price { get { return _price; } set { _price = value; } }
        public string Country { get { return _country; } set { _country = value; } }
        public double Time { get { return _time; } set { _time = value; } }

        static FilmData() { counter = 0; }
        public FilmData() { counter++;  }
        public FilmData(string name,string director, int year, ActorsList  actors,
        double price, string country, double time)
        {
            _name = name;
            _director = director;
            _year = year;
            _actors = actors;
            _price = price;
            _country = country;
            _time = time;
            counter++;
        }
        public FilmData(FilmData val)
        {
            _name = val._name;
            _director = val._director;
            _year = val._year;
            _actors = val._actors;
            _price = val._price;
            _country = val._country;
            _time = val._time;
            counter++;
        }
    }
}
```

4.2 FilmList.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;

namespace FilmApp.Model
{
    public class FilmList
    {
        List<FilmData> list = null;
        public FilmList()
        {
            list = new List<FilmData>();
        }
        public FilmList(List<FilmData> list)
        {
            this.list = list;
        }

        public List<FilmData> List {  get { return list; } set { list = value; } }

        public void  SortByCountry()
        {
            //Алгоритмом простої вибірки відсортувати записи за Країною виробництва
            if (list == null || list.Count == 0)
                throw new NullReferenceException("Your list is empty");
            FilmData fix, min;
            int min_index;
            for (int j = 0; j < list.Count - 1; j++)
            {
                fix = list[j];
                min = list[j + 1];
                min_index = j + 1;
                for (int i = j + 1; i < list.Count; i++)
                {
                    if (String.Compare(min.Country, list[i].Country, true) > 0)
                    {
                        min = list[i];
                        min_index = i;
                    }
                }
                //міняємо місцями фіксований елемент і мінімальний
                if (String.Compare(fix.Country, min.Country, true) > 0)
                {
                    list[j] = min;
                    list[min_index] = fix;
                }
            }
        }
        public string  FindPopularActor()
        {
            // Встановити найбільш популярного актора
            Dictionary<string, int> actors = new Dictionary<string, int>();
            //формуємо словник акторів і кількістю їхніх зйомок у фільмах
            for (int i = 0; i < list.Count; i++)
            {
                for (int j = 0; j < list[j].Actors.Actors.Count; j++)
                {
                    if(!actors.TryAdd(list[i].Actors.Actors[j], 1))
                    {
```

```
                actors[list[i].Actors.Actors[j]]++;
            }
        }
    }
    int max = 0;
    //знаходимо максимальну кількість зйомок
    foreach (var item in actors)
    {
        if(max < item.Value)
        {
            max = item.Value;
        }
    }
    //повертаємо актора з максмальною кількістю зйомок
    foreach (var item in actors)
    {
        if (max == item.Value)
            return item.Key;
    }
    return string.Empty;
}

public List<FilmData> FindAllFilmsWithActor(string actor_name)
{
    // За заданим актором визначити всі фільми, в яких він (вона) знімались
    if (list == null || list.Count == 0)
        throw new NullReferenceException("Your list is empty!");
    List<FilmData> films = new List<FilmData>();
    //знаходимо фільми з потрібним актором
    foreach (var film in list)
    {
        foreach (var actor in film.Actors.Actors)
        {
            if(actor == actor_name)
            {
                films.Add(film);
            }
        }
    }
    return films;
}
public List<FilmData> FindForDirectorsLongestFilm()
{
    //Для кожного Режисера визначити фільм з найбільшою тривалістю.
    if (list == null || list.Count == 0)
        throw new NullReferenceException("Your list is empty");
    List<FilmData> films = new List<FilmData>();
    Dictionary<string, FilmData> longest_films = new Dictionary<string, FilmData>();
    double max_time = 0;

    foreach (var film in list)
    {
        // find max time of director films
        foreach (var max_time_film in list)
        {
            if(max_time_film.Director == film.Director)
            {
                if(max_time < max_time_film.Time)
                {
                    max_time = max_time_film.Time;
                }
            }
        }
        // add film with max time to new list
        foreach (var max_time_film in list)
        {
```

```
                if (max_time_film.Director == film.Director)
                {
                    if (max_time == max_time_film.Time)
                    {
                        longest_films.TryAdd(max_time_film.Director, max_time_film);
                        break;
                    }
                }
            }
            max_time = 0;
        }
        films = longest_films.Select(val => val.Value).ToList();
        return films;
    }

    public FilmList FindTheMostExpensiveAndOldest()
    {
        ///Знайти найдорожчий та найстарший фільм одночасно
        if (list.Count == 0)
        {
            throw new NullReferenceException("Your list is empty");
        }
        else if (list.Count == 1)
        {
            return this;
        }
        List<FilmData> films = list.ToList() ;
        //шукаємо найдорожчі фільми
        List<FilmData> max_price = list.Where(val => val.Price == list.Max(film =>
film.Price)).ToList<FilmData>();
        //шукаємо найстарші фільми
        films = max_price.Where(val => val.Time == max_price.Min(film =>
film.Time)).ToList<FilmData>();

        return new FilmList(films);
    }

    public List<FilmData> FindSameDirectorsAndLowesBudget()
    {
        ///Визначити Назви фільмів, в яких однакові
        ///режисери та найменші бюджети одночасно.

        if (list == null || list.Count == 0)
            throw new NullReferenceException("Your list is empty!");
        //групуємо фільми по режисерах
        var group_films = from filmdata in list group filmdata by filmdata.Director;
        List<double> film_price = new List<double>(group_films.Count());
        int i = 0;
        //рахуємо суми їхніх бюджетів
        foreach (IGrouping<string, FilmData> g in group_films)
        {
            film_price.Add(new double());
            foreach (var film in g)
                film_price[i] += film.Price;
            i++;
        }
        //знаходимо індекс мінімального
        int index = film_price.IndexOf(film_price.Min());
        List<FilmData> films = group_films.ElementAt(index).Select(g => g).ToList();

        return films;
    }
    public List<FilmData> FindByCountry(string country)
    {
        ///За заданою країною виробництва знайти всі фільми,
        ///в яких найбільші бюджети і найменша тривалість одночасно.
```

```
            if (list == null || list.Count == 0)
                throw new NullReferenceException("Your list is empty!");
            //вибираємо фільми із заданої країни
            List<FilmData> films = list.Where(val => val.Country ==
country).ToList<FilmData>();

            if (films.Count == 0)
                throw new ArgumentException("There is no films from this country");
            if (films.Count == 1)
                return films;
            // вибираємо фільми з максимальною ціною
            List<FilmData> max_price = films.Where(
                film => film.Price ==
                films.Max(val => val.Price)).ToList<FilmData>();
            // вибираємо фільми з найменшою тривалістю
            List<FilmData> min_time = max_price.Where(
                val => val.Time ==
                max_price.Min(film => film.Time)).ToList<FilmData>();

            return min_time;
        }
    }
}
```

4.3 FilmsData

```
using FilmApp.Model;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;

namespace FilmApp.AppInteraction
{
    public enum ToChange { Yes, No}
    public class FilmsData
    {
        public delegate void FilmListEvent(FilmList list, ToChange value, string text);
        public static event FilmListEvent FilmsFill;
        public void UpDateFilms(FilmList filmList, ToChange value, string text)
        {
            FilmsFill?.Invoke(filmList, value, text);
        }
    }
}
```

4.4 ActorList.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FilmApp.Model
{
    public class ActorsList
    {
```

```
        List<string> actors = null;
        public List<string> Actors { get { return actors; } set { actors = value; } }

        public ActorsList() { actors = new List<string>(); }
        public ActorsList(string name)
        {
            if(actors==null)
                actors = new List<string>();
            actors.Add(name);
        }
        public ActorsList(List<string> actors_list)
        {
            actors = actors_list;
        }
        public ActorsList(ActorsList _list)
        {
            actors = _list.actors;
        }
        public override string ToString()
        {
            string str = string.Empty;
            foreach (var actor in actors)
            {
                str += actor + " ; ";
            }
            return str;
        }
    }
}
```

4.5 MainWindow.xaml.cs

```
using FilmApp.Model;
using System;
using System.Windows;
using System.IO;
using Newtonsoft.Json;
using FilmApp.AppInteraction;
using System.Diagnostics;
using Microsoft.Win32;

namespace FilmApp
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        FilmList list =null;
        FilmsData filmsData;
        public MainWindow()
        {
            InitializeComponent();
            list = new FilmList();
            filmsData = new FilmsData();
        }

        private void BSort_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                list.SortByCountry();
                filmsData.UpDateFilms(list,ToChange.Yes, "Your films sorted by country:");
```

```
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK,
MessageBoxImage.Error);
            }

        }

        private void BWriteFilmsToFile_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                string films = System.Text.Json.JsonSerializer.Serialize(list);
                File.WriteAllText($"{Directory.GetCurrentDirectory()}/films.json", films);
                MessageBox.Show("Your data saved to file.", "Successe", MessageBoxButton.OK,
MessageBoxImage.Asterisk);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK,
MessageBoxImage.Error);
            }

        }

        private void BReadFilmsFromFile_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                if(list!=null && list.List.Count > 0)
                {
                    MessageBoxResult result = MessageBox.Show("Do you want to read new data?
Your old data will be clear.", "Question",
                        MessageBoxButton.YesNo, MessageBoxImage.Question);
                    if (result == MessageBoxResult.No)
                    {
                        return;
                    }
                }
                OpenFileDialog fileDialog = new OpenFileDialog();
                fileDialog.Multiselect = false;
                fileDialog.Filter = "Text files|*.json*.*";
                fileDialog.DefaultExt = ".txt";
                Nullable<bool> dialogOk = fileDialog.ShowDialog();
                string filePath=string.Empty;
                if (dialogOk == true)
                {
                     filePath = fileDialog.FileNames[0];
                }
                if (!filePath.Contains(".json"))
                {
                    throw new Exception("We don't work with this type of files!");
                }
                FilmData.counter = 0;
                //$"{Directory.GetCurrentDirectory()}/films.json")
                using (StreamReader read = new StreamReader(filePath))
                {
                    string json = read.ReadToEnd();
                    if (json == string.Empty)
                        throw new Exception("Your file is empty! Try to open another
file.");
                    list = JsonConvert.DeserializeObject<FilmList>(json);
                }
                if (list.List.Count == 0)
```

```
                    throw new Exception("We can't convert your file data to film. Fix your
data to json format and try again.");
                filmsData.UpDateFilms(list, ToChange.Yes,"Your films:");
                MessageBox.Show("You successfully readed films from file", "Success",
MessageBoxButton.OK, MessageBoxImage.Information);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private void BFindPopularActor_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                MessageBox.Show(list.FindPopularActor(), "The most popular
actor:",MessageBoxButton.OK ,MessageBoxImage.Information);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private void BShowAllList_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                if (list.List.Count == 0)
                    throw new Exception("Your list is empty!");
                filmsData.UpDateFilms(list, ToChange.No, "Your films:");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }
        private void BFindOldestExpesive_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                filmsData.UpDateFilms(list.FindTheMostExpensiveAndOldest(),
ToChange.No,"Your the most expensive and the oldest films:");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private void BOpenMenu_Click(object sender, RoutedEventArgs e)
        {
            BOpenMenu.Visibility = Visibility.Collapsed;
            BCloseMenu.Visibility = Visibility.Visible;
            MainGrid.Margin = new Thickness(200, 0, 0, 0);
        }

        private void BCloseMenu_Click(object sender, RoutedEventArgs e)
        {
            BOpenMenu.Visibility = Visibility.Visible;
            BCloseMenu.Visibility = Visibility.Collapsed;
```

```
            MainGrid.Margin = new Thickness(70, 0, 0, 0);
        }

        private void BInfo_Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show("This app created by Somko Pavlo.\nStudent of NULP, PZ-25.",
"Info", MessageBoxButton.OK, MessageBoxImage.Information);
        }
    }
}
```

4.6 Dashboard.xaml.cs

```
using FilmApp.AppInteraction;
using FilmApp.Model;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace FilmApp.MVVM.ViewModel
{
    /// <summary>
    /// Interaction logic for DashboardViewModel.xaml
    /// </summary>
    public partial class DashboardViewModel : UserControl
    {
        private readonly FilmsData _filmdata;
        private FilmList list;
        public DashboardViewModel()
        {
            InitializeComponent();
            FilmTable.ItemsSource = new List<FilmData> { };
            _filmdata = new FilmsData();
            FilmsData.FilmsFill += OnInteract;
        }

        private void OnInteract(FilmList obj, ToChange value, string text)
        {
            try
            {
                if (value == ToChange.Yes)
                {
                    amount_fo_films.Text = "Amount of films in app: " +
FilmData.counter.ToString();
                    list = obj;
                }
                FilmTable.ItemsSource = obj.List;
                FilmTable.Items.Refresh();
                info_text.Text = text;
            }
            catch (Exception ex)
```

```
            {
                MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK,
MessageBoxImage.Error);
            }

        }

        private void BFindByCountry_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                if (parametr_to_find.Text == string.Empty || parametr_to_find.Text.Any(c =>
!char.IsLetter(c)))
                    throw new Exception("Your parameter is not valid.");
                if (list.FindByCountry(parametr_to_find.Text).Count == 0)
                    throw new Exception("There is not films from input country!");
                FilmTable.ItemsSource = list.FindByCountry(parametr_to_find.Text);
                FilmTable.Items.Refresh();
                info_text.Text = "Your films from " + parametr_to_find.Text + " with the
bigest budget and lowest time:";
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private void FindFilmsWithActor_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                if (parametr_to_find.Text == string.Empty || parametr_to_find.Text.Any(c =>
char.IsDigit(c)))
                    throw new Exception("Your parameter is not valid.");
                if (list.FindAllFilmsWithActor(parametr_to_find.Text).Count == 0)
                    throw new Exception("There is not films with input actor!");
                FilmTable.ItemsSource = list.FindAllFilmsWithActor(parametr_to_find.Text);
                FilmTable.Items.Refresh();
                info_text.Text = "Your films with " + parametr_to_find.Text ;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private void BFindWithIdentDirAndLowesPRice_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                FilmTable.ItemsSource = list.FindSameDirectorsAndLowesBudget();
                FilmTable.Items.Refresh();
                info_text.Text = "Your films with identical directors and lowest price:";
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private void FindLongesstForEachDirector_Click(object sender, RoutedEventArgs e)
        {
            try
```

```
            {
                FilmTable.ItemsSource = list.FindForDirectorsLongestFilm();
                FilmTable.Items.Refresh();
                info_text.Text = "Your the longest films for each director :";
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }

        private void BAddNewFilm_Click(object sender, RoutedEventArgs e)
        {
            try
            {
              if (film_name.Text.Any(c => !char.IsLetter(c)) ||
              film_director.Text.Any(c => !char.IsLetter(c)) ||
              film_country.Text.Any(c => !char.IsLetter(c)) ||
              film_actors.Text.Any(c => char.IsDigit(c)) ||
              film_budget.Text.Any(c => !char.IsDigit(c)) ||
              film_time.Text.Any(c => !char.IsDigit(c)) ||
              film_year.Text.Any(c => !char.IsDigit(c)))
                {
                   throw new Exception("There is an error in your data, so we can't add this
to films list. Fix your data and try again.");
                }
                else
                {
                    var film = new FilmData
                    {
                        Name = film_name.Text,
                        Director = film_director.Text,
                        Actors = new
ActorsList(film_actors.Text.Split(',').ToList<string>()),
                        Country = film_country.Text,
                        Price = Convert.ToDouble(film_budget.Text),
                        Time = Convert.ToDouble(film_time.Text),
                        Year = Convert.ToInt32(film_year.Text),
                    };
                    list.List.Add(film);
                    FilmTable.ItemsSource = list.List;
                    FilmTable.Items.Refresh();
                }
                info_text.Text = "Your films:";
                MessageBox.Show("You successfully added film", "Success",
MessageBoxButton.OK, MessageBoxImage.Information);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK,
MessageBoxImage.Error);
            }

        }
    }
}
```

5. Протокол роботи

Програма виконує такі завдання:

- Алгоритмом простої вибірки сортує записи за Країною виробництва

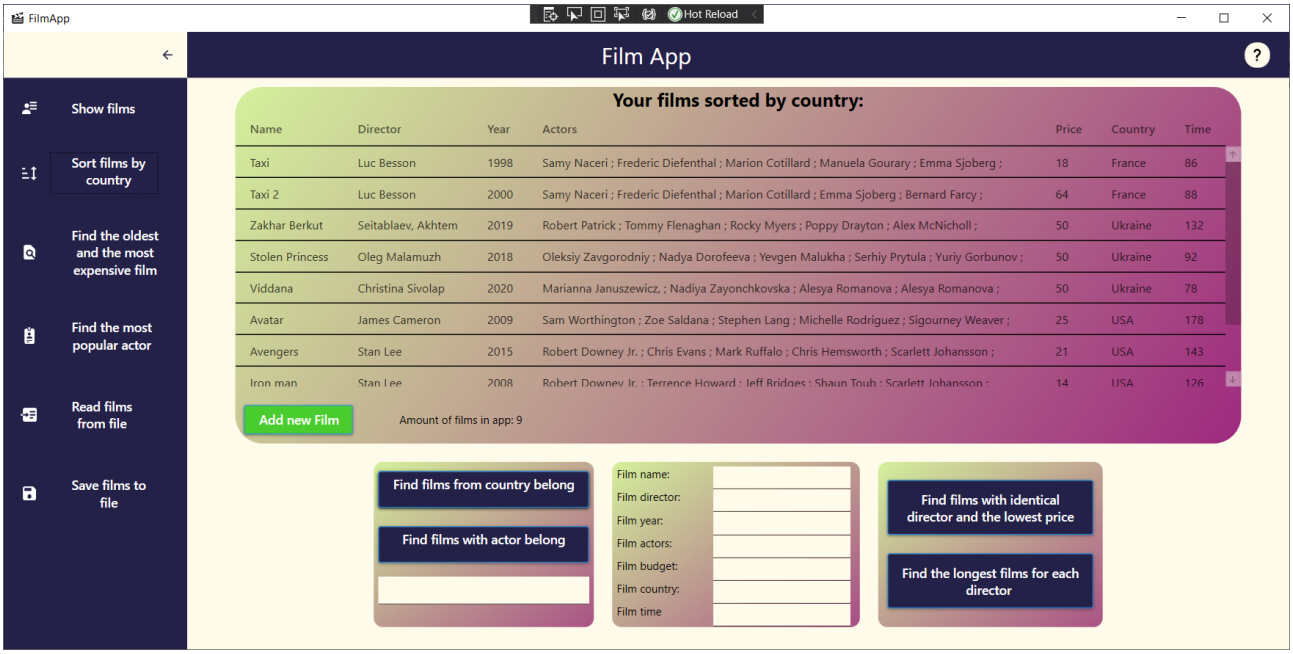

Рис. 5.1 Посортовані фільми за країною

- Визначає Назви фільмів, в яких однакові режисери та найменші бюджети одночасно.

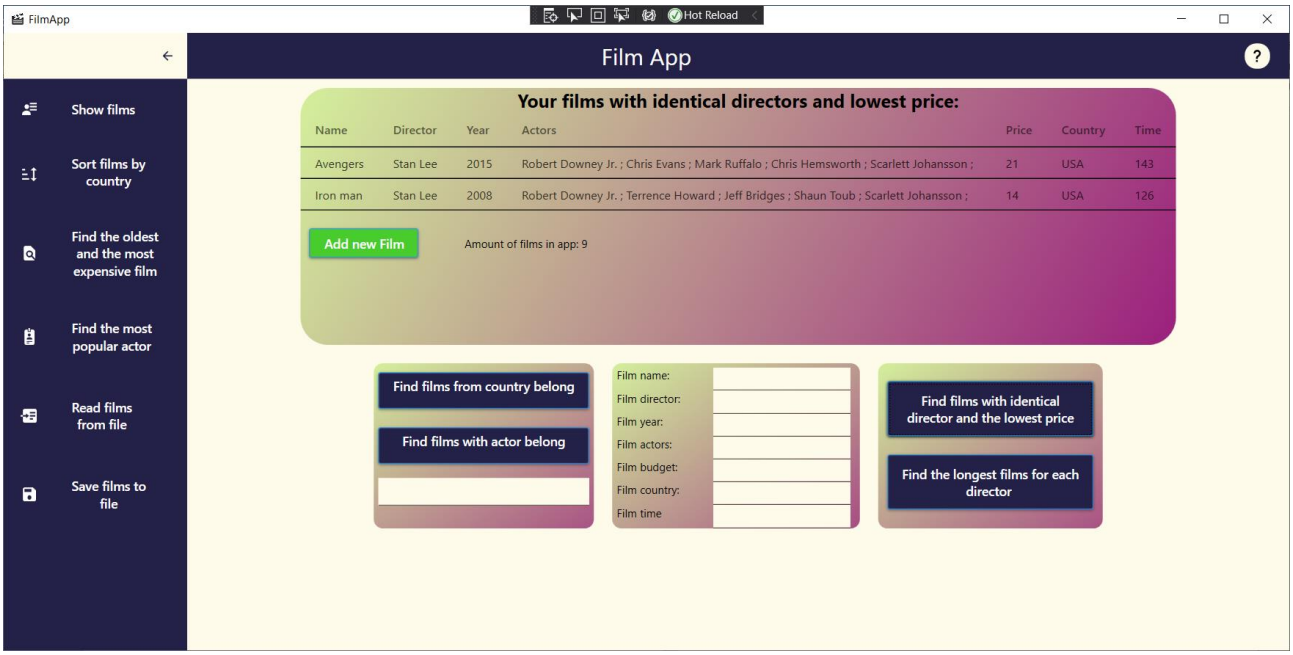

Рисунок 5.2 Назви фільмів, в яких однакові режисери та найменші бюджети одночасно.

- За заданою країною виробництва знаходить всі фільми, в яких найбільші бюджети і найменша тривалість одночасно.

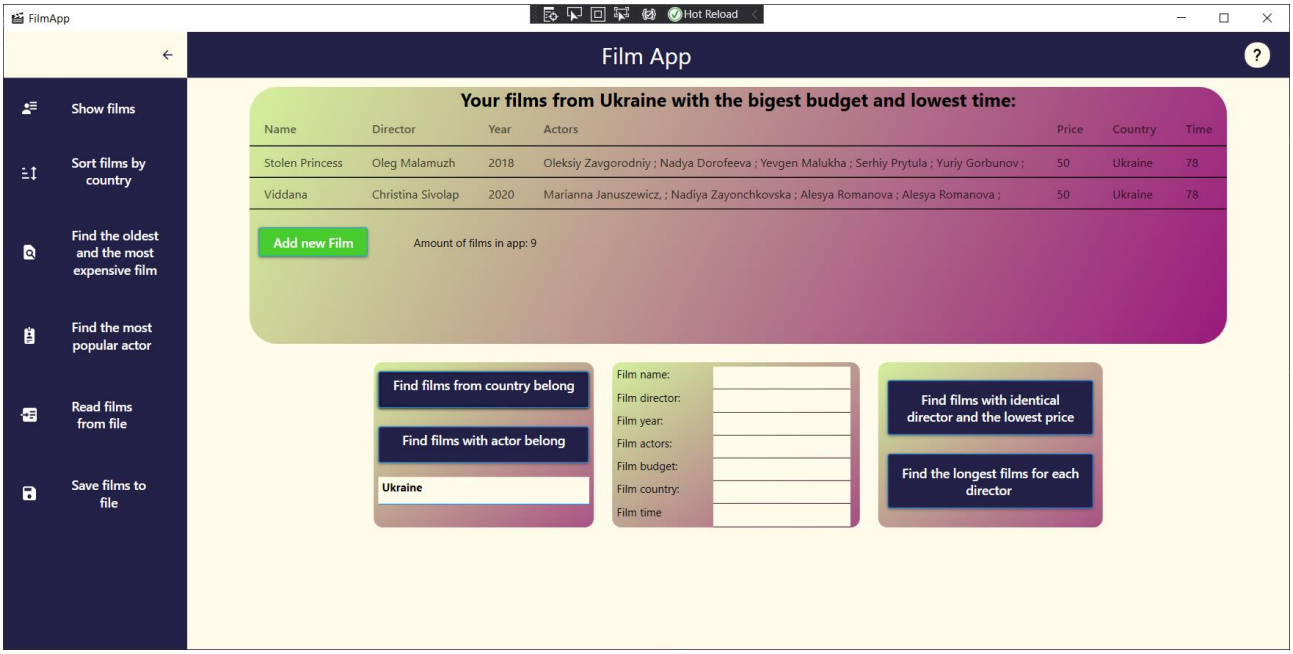

Рисунок 5.3 За заданою країною виробництва всі фільми, в яких найбільші бюджети і найменша тривалість одночасно.

- Для кожного Режисера визначає фільм з найбільшою тривалістю

Рисунок 5.4 Для кожного Режисера фільм з найбільшою тривалістю.

- Знаходить найдорожчий та найстарший фільм одночасно.

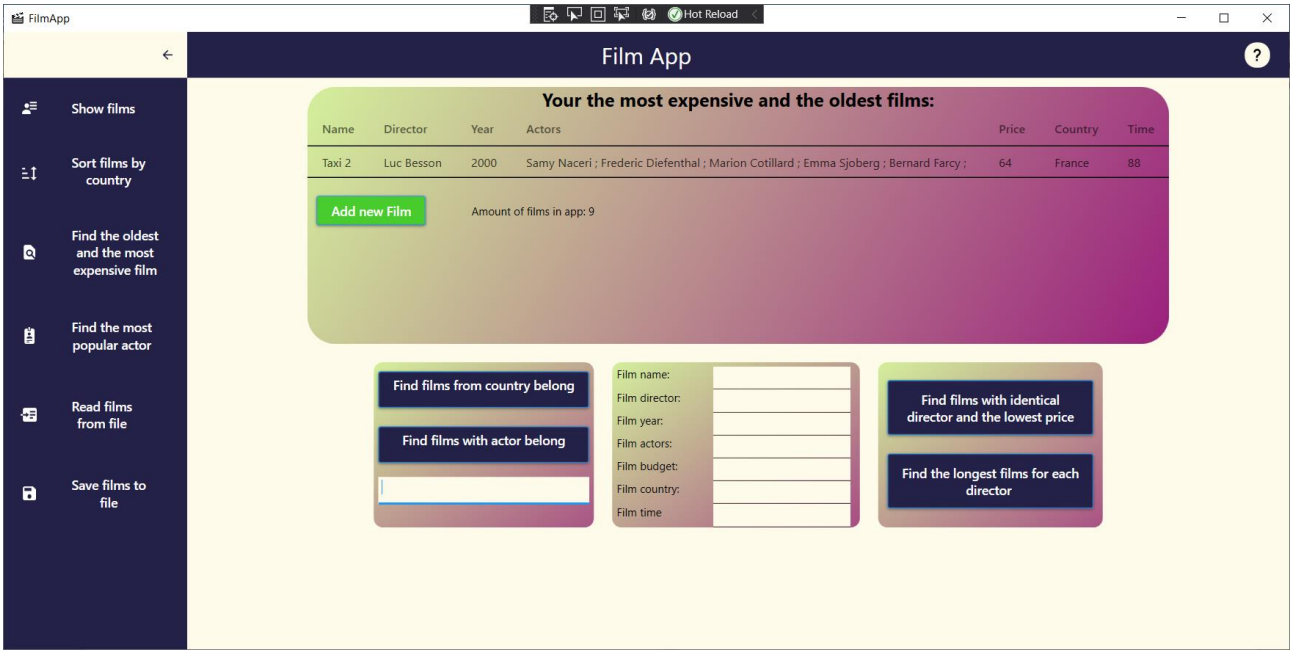

Рисунок 5.5 найдорожчий та найстарший фільм одночасно.

- За заданим актором визначає всі фільми, в яких він (вона) знімались.

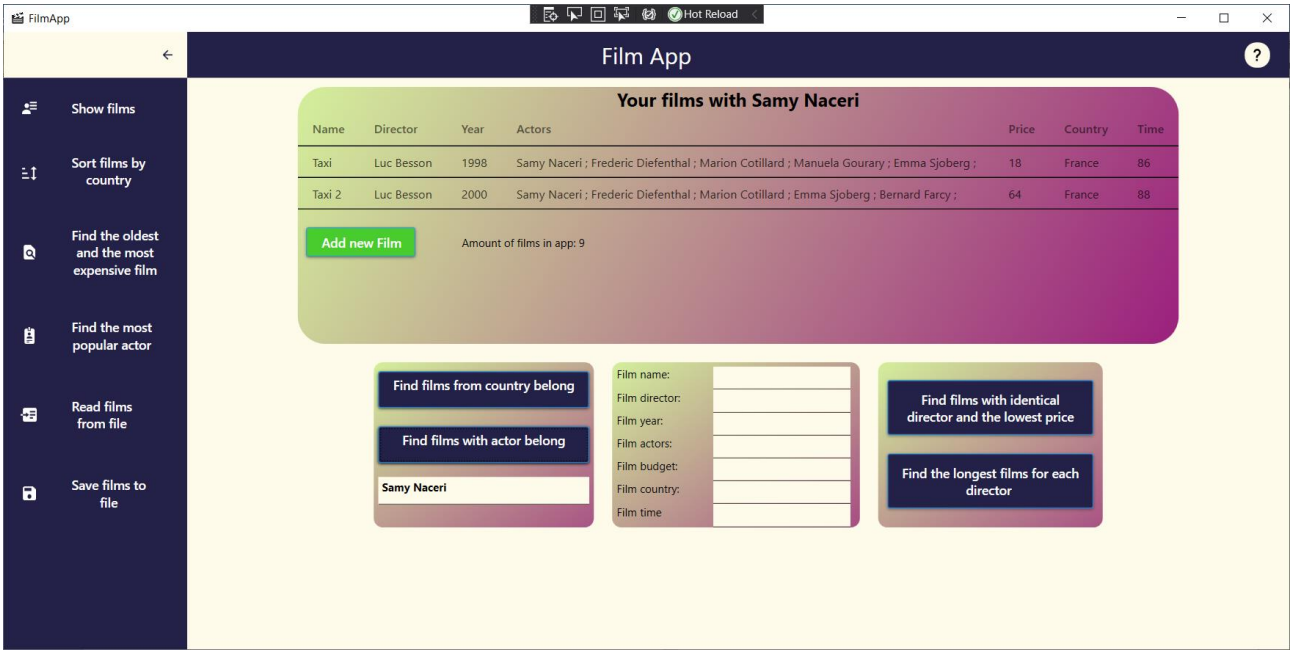

Рисунок 5.6 За заданим актором визначити всі фільми, в яких він (вона) знімались.

- Встановить найбільш популярного актора.

Рисунок 5.7 Встановити найбільш популярного актора.

6. Інструкція користувача.

1) **Компоненти ПЗ**

Програму розроблено на мові програмування C# у середовищі розробки Visual Studio 2019 з використанням технології WPF і може експлуатуватися під управлінням сімейства операційних систем Linux та MacOS. Під час проектування підсистем застосовувався об’єктно-орієнтований підхід до програмування. Всі структури та методи документувались.

2) **Встановлення ПЗ**

Для роботи програми необхідно запустити на виконання файл FilmApp.exe.

3) **Системні вимоги**

Програма потребує оперативної пам’яті не менше 1024 МБ та об’єм вільної пам’яті не менше 1024МБ.

Для коректної роботи пакету необхідна користувацька машина з процесором не менше 200 MHz.

4) **Базові функції ПЗ**

Рисунок 6.1 при запуску програми відкривається вікно загрузки

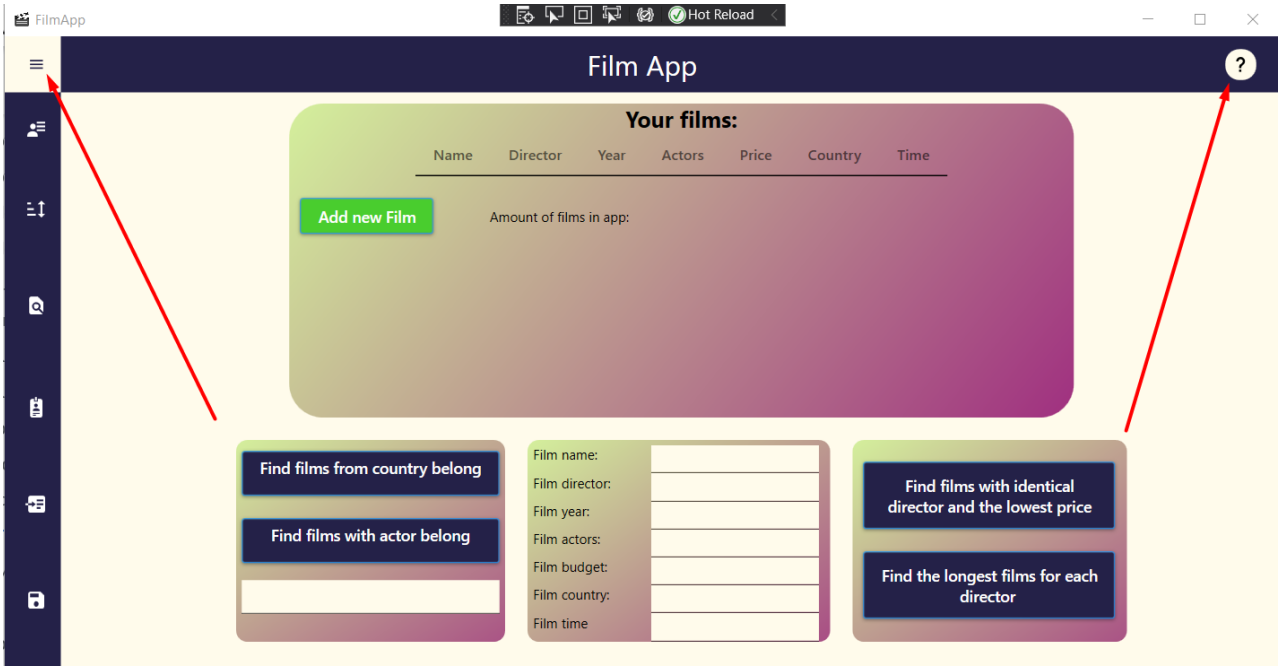

Рисунок 6.2 стартове вікно програми

Зліва вкладка меню відкривається при натиску на 3 риски, справа відкривається інформація про програми при натисканні на знак питання.

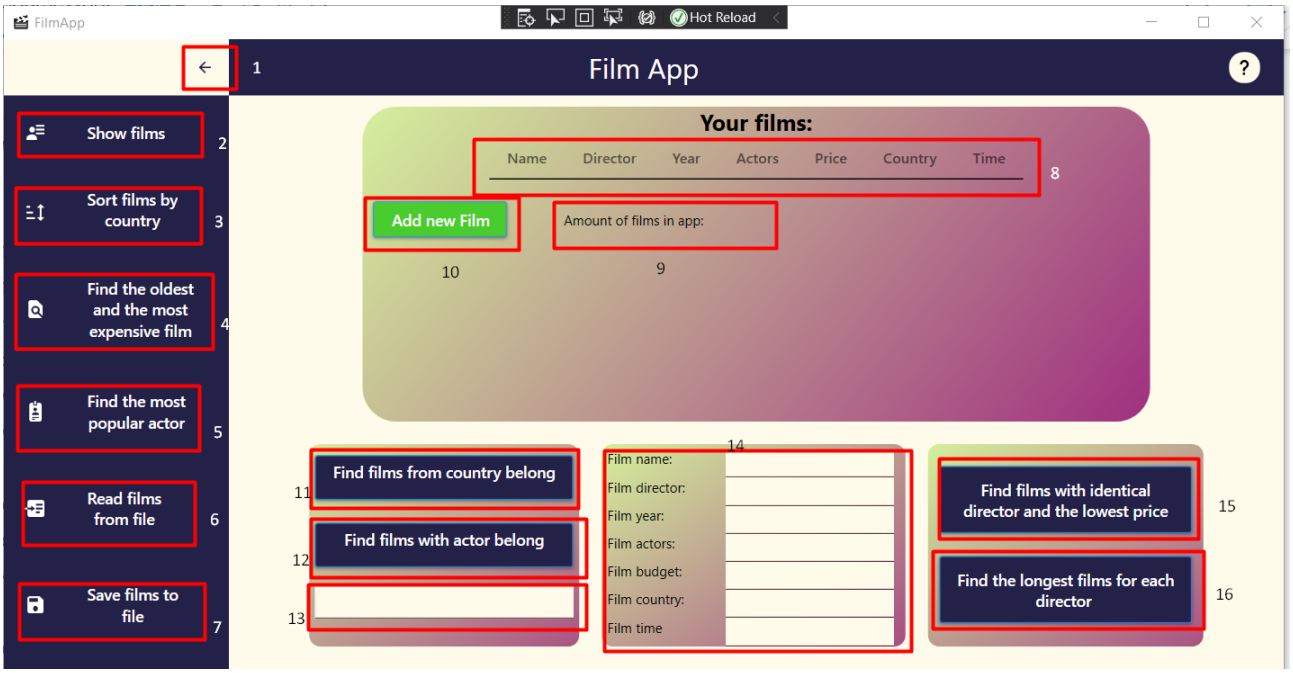

Рисунок 6.3 розгорнутий вигляд програми

1. Стрілка повернення меню до закритого стану.
2. Кнопку відображення фільмів у таблицю.
3. Кнопка сортування фільмів.
4. Кнопка пошуку найстаршого і найдорожчого фільму.
5. Кнопку пошуку найпопулярнішого актора.
6. Кнопка читання даних з файлу.
7. Кнопка збереження даних у файл.
8. Пуста таблиця.
9. Поле відображення кількості фільмів.
10. Кнопка додавання нових фільмів.
11. Кнопка пошуку фільмів за країною.
12. Кнопка пошуку фільмів із заданим актором
13. Поле вводу параметрів пошуку.
14. Поле вводу параметрів нового фільму
15. Кнопка пошуку фільмів з однаковими режисерами і найнижчою ціною.
16. Кнопка пошуку найдовших фільмів для кожного режисера.

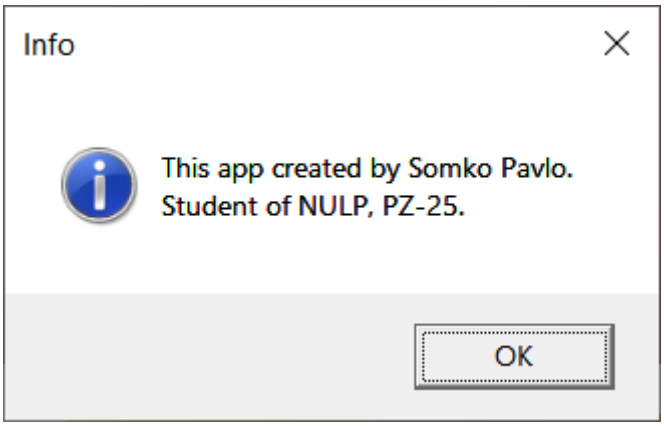

Рисунок 6.4 вікно інформації при натисканні на знак питання.

- Читання даних з файлу
  Для того, щоб прочитати дані про фільми з файлу необхідно: відкрити головне меню (нажати на 3 риски справа вгорі) → нажати на кнопку “Read films from file” і вибрати у файловому провіднику необхідний файл.
- Зберігання даних у файл
  Для того, щоб зберегти дані у файл необхідно: відкрити головне меню (нажати на 3 риски справа вгорі) → нажати на кнопку “Save films to file” і вибрати у файловому провіднику необхідний файл.
- Переглянути список фільмів.
  Для того, щоб переглянути список фільмів необхідно: відкрити головне меню (нажати на 3 риски справа вгорі) → нажати на кнопку “Show films”.
- Посортувати список фільмів.
  Для того, щоб посортувати список фільмів необхідно: відкрити головне меню (нажати на 3 риски справа вгорі) → нажати на кнопку “Sort films by country”.
- Знайти найдорожчий і найстарший фільм
  Для того, щоб знайти найдорожчий і найстарший фільм необхідно: відкрити головне меню (нажати на 3 риски справа вгорі) → нажати на кнопку “Find the oldest and the most expensive film”.
- Знайти найпопулярнішого актора
  Для того, щоб знайти найпопулярнішого актора необхідно: відкрити головне меню (нажати на 3 риски справа вгорі) → нажати на кнопку “Find the most popular actor”.
- Додати новий фільм
  Для того, щоб додати новий фільм необхідно: заповніть всі поля на формі вводу (Film name, director, year, actors, budget, country, time)→ нажати на кнопку “Add new Film”.
- Знайти фільми із заданої країни в яких найбільші бюджети і найменша тривалість одночасно

Для того, щоб додати новий фільм необхідно: заповнити поле в компоненті з кнопкою→ нажати на кнопку “Find films from country belong”.

- Знайти фільми із зданим актором
  Для того, щоб знайти фільми із зданим актором необхідно: заповнити поле в компоненті з кнопкою→ нажати на кнопку “Find films with actor belong”.
- Знайти фільми з однаковими режисерами і найменшою ціною
  Для того, щоб знайти фільми з однаковими режисерами і найменшою ціною необхідно: нажати на кнопку “Find films with identical director and the lowest price”.
- Знайти для кожного режисера найдовший фільм
  Для того, щоб знайти фільми з однаковими режисерами і найменшою ціною необхідно: нажати на кнопку “Find the longest film for each director”.

Приклад вхідних даних:


```
{
  "List": [
    {
      "Name": "Taxi",
      "Director": "Luc Besson",
      "Year": 1998,
      "Actors": { "Actors": [ "Samy Naceri", "Frederic Diefenthal", "Marion Cotillard",
"Manuela Gourary", "Emma Sjoberg" ] },
      "Price": 18,
      "Country": "France",
      "Time": 86
    }

]}
```

7. Виняткові ситуації.

У програмі передбачено такі виняткові ситуації:

- Читаємо дані з пустого файлу

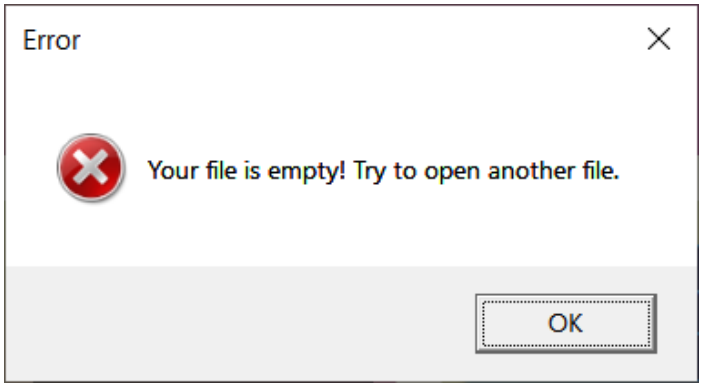

Рисунок 7.1 помилка, пустий файл

- Дані у файлі некоректного формату

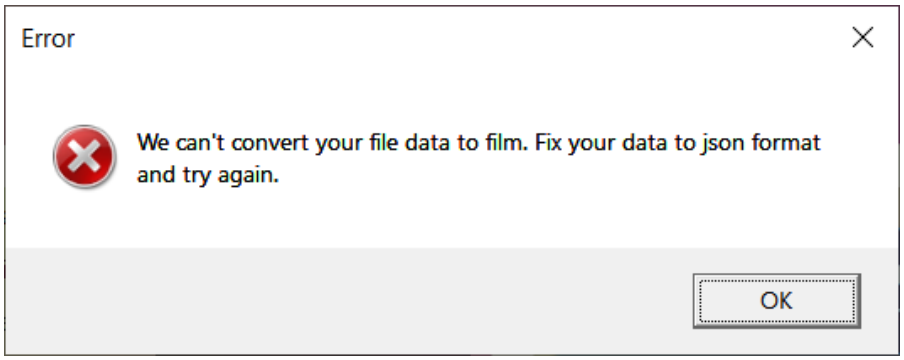

Рисунок 7.2 помилка, некоректні дані у файлі

- Помилка заповнення певного поля у файлі

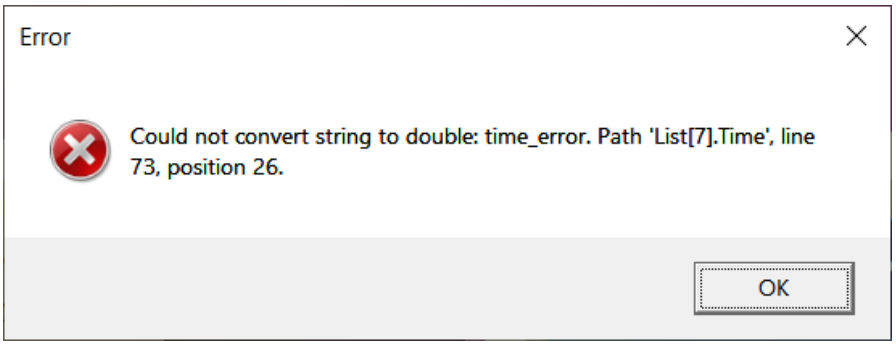

Рисунок 7.3 некоректно заповнений певний рядок у файлі

- Помилка при вводі даних з клавіатури

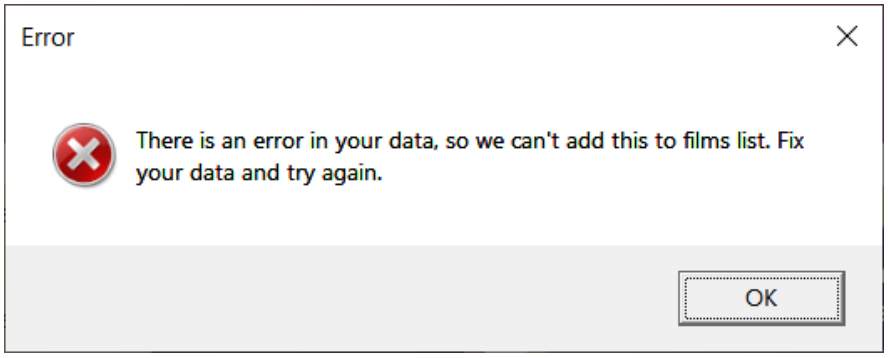

Рисунок 7.4 помилка при некоректних вхідних даних з клавіатури

- Помилка при виклику функцій над пустим списком

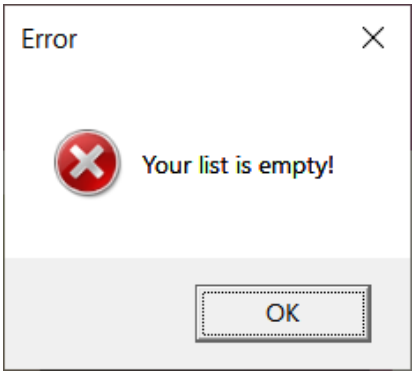

Рисунок 7.5 помилка при спробі виконати дії над пустим списком

- Помилка при пошуку фільмів з некоректно введеної країни

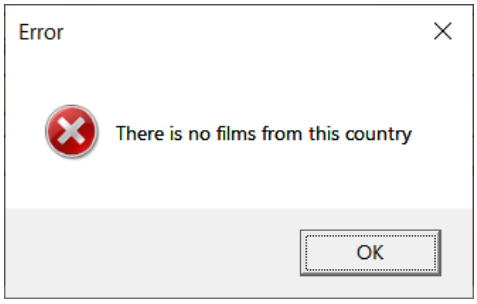

Рисунок 7.6 помилка при пошуку фільмів з некоректної країни

- Помилка при пошуку фільмів з некоректно введеного

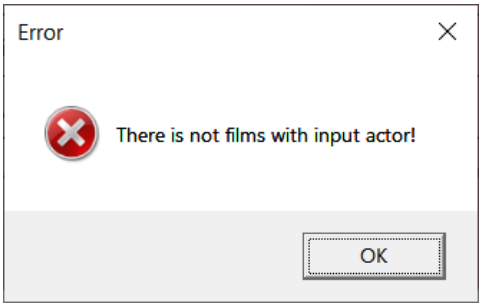

Рисунок 7.7 помилка при пошуку некоректного актора

8. Висновки

Під час виконання даної роботи, я розробив програму FilmApp, у візуальному середовищі розробити – Visual Studio 2019 з використанням технології WPF. Програма створена, щоб працювати з базою даних фільмів та їх параметрами(назва, режисер, рік випуску, список акторів, бюджет, країна розробки, час). Програма написано мовою C#.

При написанні коду, я дотримався стилю об’єктно орієнтованого програмування: для представлення моделей використав різні класи, та для взаємодії між ними використав їхні методи, дотримався принципу інкапсуляції. До написаних класів входять наступні класи: FilmData, ActorsList, FilmList, FilmsData.

9. Список використаної літератури

1. http://vns.lp.edu.ua/ - віртуальне середовище навчання НУ «ЛП».
2. https://docs.microsoft.com/ru-ru/dotnet/csharp/ - документація мови програмування C# від Microsoft.
3. https://docs.microsoft.com/ru-ru/dotnet/desktop/wpf/?view=netdesktop-5.0 – документація технології WPF від Microsoft.
4. https://metanit.com/sharp/ - інструкція мови C# та технології .NET від сайту matanit.