Изображение выглядит как цепь

Автоматически созданное описание

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный технологический институт (технический университет)»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| УГНС | | 09.00.00 | Информатика и вычислительная техника | | |
| Направление подготовки | | 09.03.01 | Информатика и вычислительная техника | | |
| Направленность (профиль) | |  | Автоматизированные системы обработки информации и управления | | |
| Форма обучения | |  | очная | | |
|  | |  |  | | |
| Факультет | |  | Информационных технологий и управления | | |
| Кафедра | |  | Систем автоматизированного проектирования и управления | | |
| Учебная дисциплина | |  | Разработка программных систем | | |
| Курс | II | | | Группа | 403 |

Отчёт по лабораторной работе № 4

Вариант № 9

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель: |  |  |  |  |
| обучающийся группы 403 |  |  |  | Шишко Даниил Юрьевич |
|  |  | (дата, подпись) |  |  |
|  |  |  |  |  |
| Проверили: |  |  |  | Корниенко Иван Григорьевич |
|  |  | (дата, подпись) |  | Федин Алексей Константинович |

**Оглавление**

[Постановка задачи 3](#_Toc97889599)

[Исходные данные 3](#_Toc97889600)

[Особые ситуации 3](#_Toc97889601)

[Математические методы и алгоритмы решения задач 3](#_Toc97889602)

[Форматы представления данных 3](#_Toc97889603)

[Структура программы 5](#_Toc97889604)

[Блок-схемы алгоритмов программы 6](#_Toc97889605)

[Описание хода выполнения работы 6](#_Toc97889606)

[Результат работы программы 7](#_Toc97889607)

[Исходный текст программы 8](#_Toc97889608)

[Документирование и комментирование исходного текста 19](#_Toc97889609)

# Постановка задачи

Необходимо написать приложение с использованием технологии WinForms реализующие вариант задания. Программа должна позволять добавлять новые сущности с использованием интерфейса и редактировать существующие. Сущности, добавленные в программу должны сохраняться между запусками приложения. Для хранения данных необходимо использовать СУБД SqLite. Необходимо предусмотреть возможность сохранения списка существующих сущностей в файл. Необходимо выполнить задание из вариантов обычной сложности используя паттерн MVP (или MVVM если используется WPF) и библиотеку Autofac для внедрения зависимостей. Необходимо использовать асинхронное подключение к СУБД (SQLiteAsyncConnection).

# Исходные данные

В качестве исходных данных программа использует файл базы данных, который был заранее сгенерирован, в котором сохраняются выбранные и в последствии сохраненные пользователем данные, в данном случае фильмы.

# Особые ситуации

Необходимо рассмотреть следующие особые ситуации.

* не введены данные, а пользователь пытается сохранить результаты;
* данные введены не полностью, а пользователь пытается сохранить результаты.

# Математические методы и алгоритмы решения задач

Используем запросы, которые составляются с использованием языка MySQL.

# Форматы представления данных

Программа использует следующие переменные

Таблица 1 – Переменные, используемые в программе

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | | **Тип** | **Описание** |
| sqlExpression | | string | Запрос MySQL |
| connection | | SqliteConnection | Соединения с базой |
| command | | SqliteCommand | Команда для базы данных |
| CanExecuteChanged | | EventHandler | Событие характеризующее возможность изменения |
| \_Execute | | Action<object> | Выполнени объекта |
| Func<object, bool> | Func<object, bool> | | Может ли объект выполнится | |
| Id | | int | Идентификационный номер |
| Продолжение таблицы 1 – Переменные, используемые в программе | | | |
| **Имя** | | **Тип** | **Описание** |
| Title | | string | Название фильма |
| Description | | string | Описание фильма |
| Icon | | string | Путь к изображению |
| Trailer | | string | Путь к трейлеру |
| Year | | System.DateTime | Дата выпуска в показ |
| PropertyChanged | | PropertyChangedEventHandler | Выполняется при изменении наблюдаемого объекта |
| \_displayRootRegistry | | DisplayRootRegistry | Реализация контейнера для хранения окон |
| \_title | | int | Идентификационный номер |
| \_description | | string | Название фильма |
| \_icon | | string | Описание фильма |
| \_trailer | | string | Путь к изображению |
| \_year | | string | Путь к трейлеру |
| \_oldTitle | | DateTime | Дата выпуска в показ |
| SaveDataCommand | | ICommand | Команда для сохранения данных |
| AddIconCommand | | ICommand | Команда для добавления изображения |
| AddTrailerCommand | | ICommand | Команда для добавления трейлера |
| \_isChecked | | bool | Помечено ли больше не показывать |
| CloseApplicationCommand | | ICommand | Команда для закрытия окна |
| \_concentrationService | | IFilmService | Для взаимодействия с уровнем сервисов |
| \_films | | ObservableCollection<Film> | Коллекция фильмов |
| WhichFilm | | Film | Выбранный фильм |
| \_mainWindowService | | IContainer | Контейнер с классами для передачи из через Autofac |
| vmToWindowMapping | | Dictionary<Type, Type> | Словарь с возможными окнами для открытия |
| \_concentrationService | | IRepository | Для взаимодействия с уровнем репозитория |

Таблица 3 – Пользовательские типы

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| Film | Для взаимодействия уровня ViewModel и View |
| RawFilm | Для взаимодействия уровня Model и ViewModel |

# Структура программы

Программа разделена на следующие модули:

1. Repository.cs;
2. Command.cs;
3. LambdaCommand.cs;
4. IFilmrepository.cs;
5. IRepository.cs;
6. Film.cs;
7. RawFilm.cs;
8. FilmServices.cs;
9. ViewModel.cs;
10. AddFilmViewModel.cs;
11. GreetingsWindowViewModel.cs;
12. InformationAboutFilmWindowViewModel.cs;
13. MainWindowViewModel.cs;

Таблица 4 – Функции, составляющие программу

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| UpdateBase | Обновить значение в базе |
| AddToBase | Добавить строку к базе |
| FindInBase | Найти значение в базе |
| Exist | Проверить существование значения в базе |
| GetCountOfRows | Получить количество строк в базе |
| GetFirstId | Получить первый id в базе |
| DeleteFromBase | Удалить значение из базы |
| CanExecute | Может ли быть выполнена функция |
| Execute | Выполнение функции |
| SaveToFile | Сохранения в файл |
| Alarm | Предупреждение |
| NotificationFileSaved | уведомление о сохранении |
| OnCloseApplicationCommandExecuted | Закрытие окна |
| CanCloseApplicationCommandExecute | Может ли окно закрыться |
| OnSaveDataCommandExecuted | Сохранение данных в базу |
| CanSaveDataCommandExecute | Могут ли данные быть сохранены в файл |
| OnAddIconCommandExecuted | Сохранение изображения в буфер |
| CanAddIconCommandExecute | Может ли выполниться сохранение изображения в буфер |
| OnAddTrailerCommandExecuted | Сохранение трейлера в буфер |
| CanAddTrailerCommandExecute | Может ли выполниться сохранение трейлера в буфер |
| CanInformationAboutFilmCommandExecute | Может ли быть открыто окно с информацией о фильме |
| OnInformationAboutFilmCommandExecuted | Открытие окна с информацией о фильме |
| CanOpenInnerWindowCommandExecute | Может ли быть открыто окно |
| OnOpenInnerWindowCommandExecuted | Открытие стороннего окна |
| CanOpenGreetingWindowCommandExecute | Может ли быть открыто окно приветствия |
| OnOpenGreetingWindowCommandExxecuted | Открытие окна приветствия |
|  |  |
| Продолжение таблицы 4 – Функции, составляющие программу | |
| **Имя** | **Описание** |
| CanEditFilmCommandExecute | Может ли быть открыто окно редактирования параметров фильма |
| OnEditFilmCommandExecuted | Открытие окна редактирования параметров фильма |
| CanRefreshDataBaseCommandExecute | может ли выполниться обновления актуальной информации отображаемой пользователю |
| OnRefreshDataBaseCommandExecuted | Обновлении информации до актуальной |
| CanDeleteDataFromDataBaseCommandExecute | Может ли быть выполнено удаление фильма |
| OnDeleteDataFromDataBaseCommandExecuted | Удаление фильма из базы |
| CanSaveToFileCommandExecute | Может ли быть выполнено сохранение в файл данных из базы |
| OnSaveToFileCommandExecuted | Сохранение в файл данных из быза данных |

# Блок-схемы алгоритмов программы

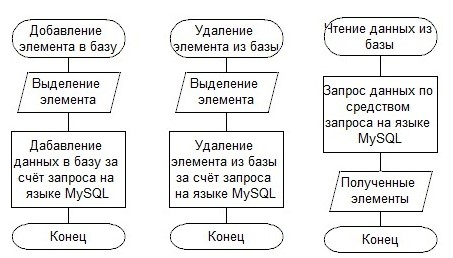


Рисунок 1 – Блок-схема алгоритма решения

# Описание хода выполнения работы

1. В ходе работы было создано решение (Solution) в интегрированной среде разработки Microsoft Visual Studio 2015. В нём был создан проект.
2. Создано оформление главной формы, реализовано масштабирование графика.
3. Создана форма приветствия.
4. Создан DataGrid для хранения данных, которые будут приходить из базы данных.
5. Создано копирование афиши фильма и его трейлера в внутреннюю папку.
6. Подключен автофак, и реализован только один контейнер для хранения ссылок на нужные классы.
7. Тип string для хранения времени изменён на DateTime и добавлен календарь для выбора даты.

# Результат работы программы

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 – Результат работы программы

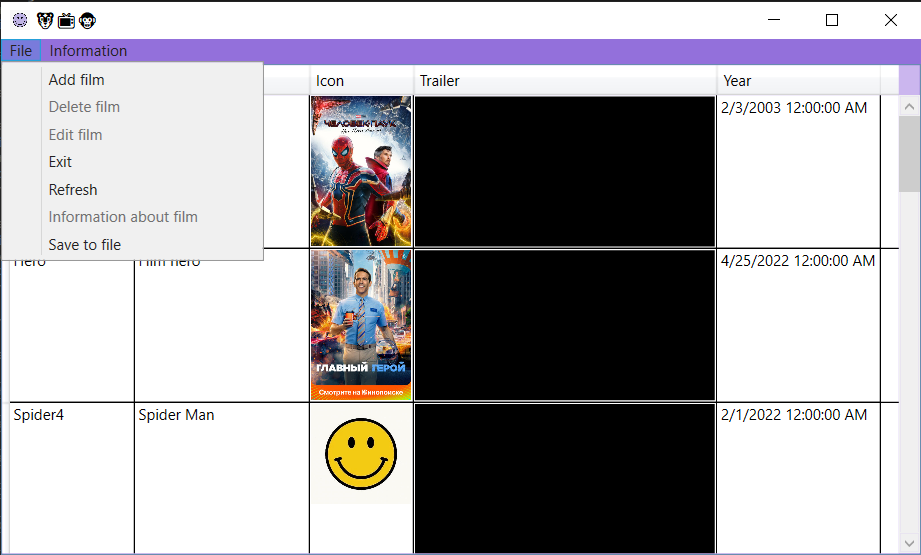


Рисунок 3 – Результат работы программы

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 – Результат работы программы

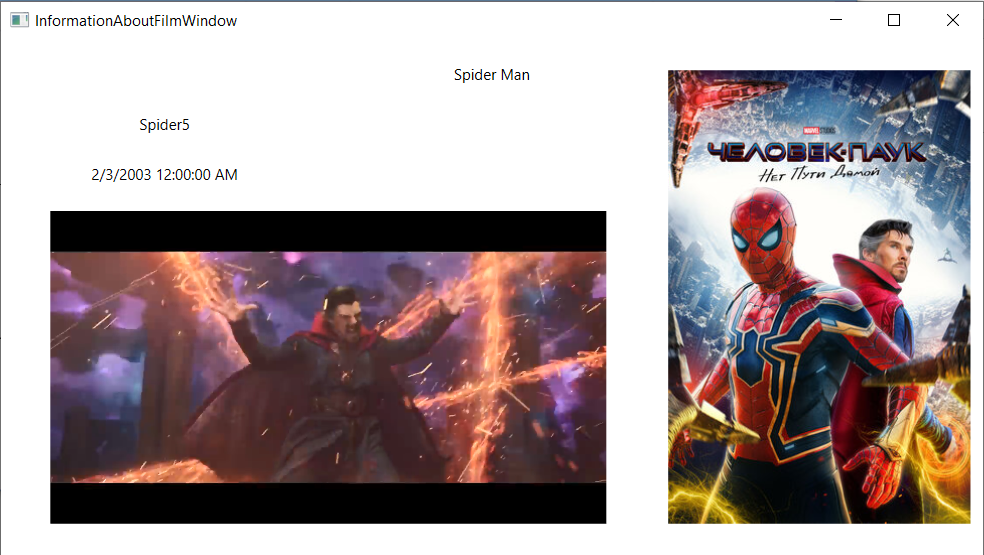


Рисунок 5 – Результат работы программы

# Исходный текст программы

[ Начало программы ---]

[Repository]

using Microsoft.Data.Sqlite;

using System;

using System.Configuration;

using System.Threading.Tasks;

using FilmsDataBase.Models;

using FilmsDataBase.Infrastructure;

namespace FilmsDataBase.Data

{

public class Repository : IRepository

{

public async Task UpdateBase(RawFilm film, int id)

{

string sqlExpression = $"UPDATE Films SET Title = @Title, Description =

@Description" +

$", Icon = @Icon, Trailer = @Trailer, Year = @Year WHERE id = @Id";

using (var connection = new

SqliteConnection(ConfigurationManager.ConnectionStrings["DefaultConnection"].

ConnectionString))

{

connection.Open();

SqliteCommand command = new SqliteCommand(sqlExpression, connection);

command.Parameters.Add(new SqliteParameter("@Title", film.Title));

command.Parameters.Add(new SqliteParameter("@Description",

film.Description));

command.Parameters.Add(new SqliteParameter("@Icon", film.Icon));

command.Parameters.Add(new SqliteParameter("@Trailer", film.Trailer));

command.Parameters.Add(new SqliteParameter("@Year", film.Year));

command.Parameters.Add(new SqliteParameter("@Id", id));

await command.ExecuteNonQueryAsync();

connection.Close();

}

}

public async Task AddToBase(RawFilm film)

{

string sqlExpression = "INSERT INTO Films (Title, Description, Icon, Trailer,

Year) VALUES (@Title, @Description, @PathToTrailer, @PathToImage, @Year)";

SqliteConnection connection = new

SqliteConnection(ConfigurationManager.ConnectionStrings["DefaultConnection"].

ConnectionString);

connection.Open();

SqliteCommand command = new SqliteCommand(sqlExpression, connection);

command.Parameters.Add(new SqliteParameter("@Title", film.Title));

command.Parameters.Add(new SqliteParameter("@Description", film.Description));

command.Parameters.Add(new SqliteParameter("@PathToImage", film.Icon));

command.Parameters.Add(new SqliteParameter("@PathToTrailer", film.Trailer));

command.Parameters.Add(new SqliteParameter("@Year", film.Year));

await command.ExecuteNonQueryAsync();

connection.Close();

}

public RawFilm FindInBase(RawFilm film, int id = -1)

{

film.Title = null;

film.Description = null;

film.Icon = null;

film.Trailer = null;

film.Year = new DateTime();

film.Id = new int();

string sqlExpression = $"SELECT \* FROM Films WHERE (id = @Id)";

using (var connection = new

SqliteConnection(ConfigurationManager.ConnectionStrings["DefaultConnection"].

ConnectionString))

{

connection.Open();

SqliteCommand command = new SqliteCommand(sqlExpression, connection);

if (id == -1)

command.Parameters.Add(new SqliteParameter("@Id", GetFirstId()));

else

command.Parameters.Add(new SqliteParameter("@Id", id));

using (SqliteDataReader reader = command.ExecuteReader())

{

if (!reader.HasRows) return film;

while (reader.Read())

{

film.Id = Convert.ToInt32(reader.GetValue(0));

film.Title = (string)reader.GetValue(1);

film.Description = (string)reader.GetValue(2);

film.Icon = (string)reader.GetValue(3);

film.Trailer = (string)reader.GetValue(4);

film.Year = Convert.ToDateTime(reader.GetValue(5));

if (film.Id == id) break;

}

}

connection.Close();

}

return film;

}

public bool Exist(int id)

{

string sqlExpression = $"SELECT \* FROM Films WHERE (id = @Id)";

using (var connection = new

SqliteConnection(ConfigurationManager.ConnectionStrings["DefaultConnection"].

ConnectionString))

{

connection.Open();

SqliteCommand command = new SqliteCommand(sqlExpression, connection);

command.Parameters.Add(new SqliteParameter("@Id", id));

if (Convert.ToInt32(command.ExecuteScalar()) == 0) return false;

return true;

}

}

public int GetCountOfRows()

{

string sqlExpression = @"SELECT COUNT(\*) FROM Films";

using (var connection = new

SqliteConnection(ConfigurationManager.ConnectionStrings["DefaultConnection"].

ConnectionString))

{

connection.Open();

SqliteCommand command = new SqliteCommand(sqlExpression, connection);

return Convert.ToInt32(command.ExecuteScalar());

}

}

public int GetFirstId()

{

string sqlExpression = @"SELECT MIN(id) FROM Films";

using (var connection = new

SqliteConnection(ConfigurationManager.ConnectionStrings["DefaultConnection"].

ConnectionString))

{

connection.Open();

SqliteCommand command = new SqliteCommand(sqlExpression, connection);

return Convert.ToInt32(command.ExecuteScalar());

}

}

public async Task DeleteFromBase(int id)

{

using (var connection = new

SqliteConnection(ConfigurationManager.ConnectionStrings["DefaultConnection"].

ConnectionString))

{

connection.Open();

string sqlExpression = $"DELETE FROM Films where (id = @Id)";

SqliteCommand command = new SqliteCommand(sqlExpression, connection);

command.Parameters.Add(new SqliteParameter("@Id", id));

await command.ExecuteNonQueryAsync();

connection.Close();

}

}

}

}

[Repository]

[Command]

using System.Windows.Input;

using System;

namespace FilmsDataBase.Infrastructure.Commands.Base

{

internal abstract class Command : ICommand

{

public event EventHandler CanExecuteChanged

{

add => CommandManager.RequerySuggested += value;

remove => CommandManager.RequerySuggested -= value;

}

public abstract bool CanExecute(object parameter);

public abstract void Execute(object parameter);

}

}

[Command]

[LamdbaCommand]

using FilmsDataBase.Infrastructure.Commands.Base;

using System;

namespace FilmsDataBase.Infrastructure.Commands

{

internal class LambdaCommand : Command

{

private readonly Action<object> \_Execute;

private readonly Func<object, bool> \_CanExecute;

public LambdaCommand(Action<object> Execute, Func<object, bool> CanExecute =

null)

{

\_Execute = Execute ?? throw new ArgumentNullException(nameof(Execute));

\_CanExecute = CanExecute;

}

public override bool CanExecute(object parameter) =>

\_CanExecute?.Invoke(parameter) ?? true;

public override void Execute(object parameter) => \_Execute(parameter);

}

}

[LamdbaCommand]

[IFilmService]

using FilmsDataBase.Models;

using System.Collections.Generic;

namespace FilmsDataBase.Infrastructure

{

public interface IFilmService

{

List<Film> GetData();

void SetData(string title, string description, string icon, string trailer,

System.DateTime year);

void DeleteData(int id);

void UpdataDataBase(int id, string newTitle, string newDescription, string

newIcon, string newTrailer, System.DateTime newYear);

bool Exist(int id);

bool IsEmpty();

bool SaveToFile();

}

}

[IFilmService]

[IRepository]

using FilmsDataBase.Models;

using System.Threading.Tasks;

namespace FilmsDataBase.Infrastructure

{

public interface IRepository

{

Task UpdateBase(RawFilm film, int id);

Task AddToBase(RawFilm film);

Task DeleteFromBase(int id);

RawFilm FindInBase(RawFilm film, int id);

bool Exist(int id);

int GetCountOfRows();

int GetFirstId();

}

}

[IRepository]

[Film]

namespace FilmsDataBase.Models

{

public class Film

{

public int Id { get; set; }

public string Title { get; set; }

public string Description { get; set; }

public string Icon { get; set; }

public string Trailer { get; set; }

public System.DateTime Year { get; set; }

public Film(int id, string Title, string Description, string Icon, string

Trailer, System.DateTime Year)

{

this.Year = Year;

this.Title = Title;

this.Description = Description;

this.Icon = Icon;

this.Trailer = Trailer;

this.Id = id;

}

public Film() { }

public object Clone()

{

return new Film(Id, Title, Description, Icon, Trailer, Year);

}

}

}

[Film]

[RawFilm]

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FilmsDataBase.Models

{

public class RawFilm

{

public int Id { get; set; }

public string Title { get; set; }

public string Description { get; set; }

public string Icon { get; set; }

public string Trailer { get; set; }

public System.DateTime Year { get; set; }

}

}

[RawFilm]

[FilmService]

using FilmsDataBase.Models;

using System.Collections.Generic;

using FilmsDataBase.Data;

using FilmsDataBase.Infrastructure;

using System;

using System.Windows.Forms;

using System.IO;

namespace FilmsDataBase.Services

{

internal class FilmService : IFilmService

{

private List<Film> innerFilms;

private static IRepository \_concentrationService = null;

public FilmService(IRepository concentrationService) =>

\_concentrationService = concentrationService ?? throw new

ArgumentNullException(nameof(concentrationService));

public FilmService() { }

public List<Film> GetData()

{

if (innerFilms == null)

innerFilms = new List<Film>();

else

innerFilms.Clear();

int counter = 0;

for (int i = \_concentrationService.GetFirstId(); counter <

\_concentrationService.GetCountOfRows(); i++)

{

RawFilm film = new RawFilm();

if (!(\_concentrationService.FindInBase(film, i).Title != null))

{

continue;

}

innerFilms.Add(new Film

{

Id = i,

Title = film.Title,

Description = film.Description,

Icon = film.Icon,

Trailer = film.Trailer,

Year = film.Year,

});

counter++;

}

return innerFilms;

}

public void SetData(string title, string description, string icon, string

trailer, System.DateTime year) =>

\_concentrationService.AddToBase(new RawFilm() { Title = title, Description =

description, Icon = icon, Trailer = trailer, Year = year }).Wait();

public void DeleteData(int id) =>

\_concentrationService.DeleteFromBase(id).Wait();

public void UpdataDataBase(int id, string newTitle, string newDescription,

string newIcon, string newTrailer, System.DateTime newYear) =>

\_concentrationService.UpdateBase(new RawFilm() { Title = newTitle, Description

= newDescription, Icon = newIcon, Trailer = newTrailer, Year = newYear },

id).Wait();

public bool Exist(int id) => \_concentrationService.Exist(id);

public bool IsEmpty() => \_concentrationService.GetCountOfRows() == 0;

public bool SaveToFile()

{

var films = GetData();

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.InitialDirectory = "SavedFiles";

saveFileDialog.Filter = "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

if (saveFileDialog.ShowDialog() == DialogResult.OK)

{

try

{

var filePath = saveFileDialog.FileName;

StreamWriter file = new StreamWriter(filePath, false);

foreach(var film in films)

{

file.WriteLine(film.Title);

file.WriteLine(film.Description);

file.WriteLine(film.Icon);

file.WriteLine(film.Trailer);

file.WriteLine(film.Year);

file.WriteLine("----------------------------");

}

file.Close();

}

catch

{

return false;

}

}

return true;

}

}

}

[FilmService]

[ViewModel]

using System.ComponentModel;

using System.Runtime.CompilerServices;

namespace FilmsDataBase.ViewModels.Base

{

public abstract class ViewModel : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged([CallerMemberName] string PropertyName

= null)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(PropertyName));

}

protected virtual bool Set<T>(ref T field, T value, [CallerMemberName] string

PropertyName = null)

{

if (Equals(field, value)) return false;

field = value;

OnPropertyChanged(PropertyName);

return true;

}

}

}

[ViewModel]

[AddFilmViewModel]

using FilmsDataBase.ViewModels.Base;

using System.Windows.Input;

using FilmsDataBase.Infrastructure.Commands;

using System.Windows;

using System.Windows.Forms;

using System;

using FilmsDataBase.Infrastructure;

using System.IO;

namespace FilmsDataBase.ViewModels

{

public class AddFilmViewModel : ViewModel

{

#region Autofac

private static IFilmService \_concentrationService;

public AddFilmViewModel(IFilmService concentrationService) =>

\_concentrationService = concentrationService ?? throw new

ArgumentNullException(nameof(concentrationService));

#endregion

private void NotificationFileSaved()

{

System.Windows.MessageBox.Show(

"File saved",

"Notification",

(MessageBoxButton)MessageBoxButtons.OK,

(MessageBoxImage)MessageBoxIcon.Information,

(MessageBoxResult)MessageBoxDefaultButton.Button1,

System.Windows.MessageBoxOptions.DefaultDesktopOnly);

}

private MessageBoxResult Alarm(string message, string caption, MessageBoxButton

button, MessageBoxImage icon) =>

System.Windows.MessageBox.Show(message, caption, button, icon);

#region Properties

private DisplayRootRegistry \_displayRootRegistry;

public DisplayRootRegistry DisplayRootRegistry

{

set { \_displayRootRegistry = value; }

get { return \_displayRootRegistry; }

}

private string \_title;

public string Title { get => \_title; set => Set(ref \_title, value); }

private string \_description;

public string Description { get => \_description; set => Set(ref \_description,

value); }

private string \_icon;

public string Icon { get => \_icon; set => Set(ref \_icon, value); }

private string \_trailer;

public string Trailer { get => \_trailer; set => Set(ref \_trailer, value); }

private DateTime \_year;

public DateTime Year { get => \_year; set => Set(ref \_year, value); }

private string \_oldTitle;

public string OldTitle { get => \_oldTitle; set => Set(ref \_oldTitle, value); }

public int id;

#endregion

#region Commands

#region CloseApplicationCommand

public ICommand CloseApplicationCommand { get; }

private void OnCloseApplicationCommandExecuted(object p)

{

if (Alarm("Are you sure you want to close window?", "Word Processor",

MessageBoxButton.YesNo, MessageBoxImage.Warning) == MessageBoxResult.Yes)

DisplayRootRegistry.HidePresentation(this);

}

private bool CanCloseApplicationCommandExecute(object p) => true;

#endregion

#region SaveDataCommand

public ICommand SaveDataCommand { get; }

private void OnSaveDataCommandExecuted(object p)

{

File.Copy(Icon, Path.Combine("D:\\4

семестр\\РПС\\FilmsDataBase\\FilmsDataBase\\Saved\\Images\\" + Title + ".jpg"));

File.Copy(Trailer, Path.Combine("D:\\4

семестр\\РПС\\FilmsDataBase\\FilmsDataBase\\Saved\\Trailer\\" + Title +

".mp4"));

if (\_concentrationService.IsEmpty() || !\_concentrationService.Exist(id))

\_concentrationService.SetData(Title, Description, "D:\\4

семестр\\РПС\\FilmsDataBase\\FilmsDataBase\\Saved\\Images\\" + Title +

".jpg",

"D:\\4 семестр\\РПС\\FilmsDataBase\\FilmsDataBase\\Saved\\Trailer\\" +

Title + ".mp4", Year);

else

\_concentrationService.UpdataDataBase(id, Title, Description, "D:\\4

семестр\\РПС\\FilmsDataBase\\FilmsDataBase\\Saved\\Images\\" + Title +

".jpg",

"D:\\4 семестр\\РПС\\FilmsDataBase\\FilmsDataBase\\Saved\\Trailer\\" +

Title + ".mp4", Year);

NotificationFileSaved();

DisplayRootRegistry.HidePresentation(this);

}

private bool CanSaveDataCommandExecute(object p)

{

if (Title == string.Empty || Title == null) return false;

if (Description == string.Empty || Description == null) return false;

if (Icon == string.Empty || Icon == null) return false;

if (Trailer == string.Empty || Trailer == null) return false;

return true;

}

#endregion

#region AddIconCommand

public ICommand AddIconCommand { get; }

private void OnAddIconCommandExecuted(object p)

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "Files|\*.jpg;\*.jpeg;\*.png";

openFileDialog.InitialDirectory = "D:\\4 семестр\\РПС\\FilmsDataBase\\";

if(openFileDialog.ShowDialog() == DialogResult.OK)

{

var filePath = openFileDialog.FileName;

Icon = filePath;

}

}

private bool CanAddIconCommandExecute(object p) => true;

#endregion

#region AddTrailerCommand

public ICommand AddTrailerCommand { get; }

private void OnAddTrailerCommandExecuted(object p)

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "All Media Files|\*.wav;\*.aac;\*" +

".wma;\*.wmv;\*.avi;\*.mpg;\*.mpeg;\*.m1v;\*.mp2;\*.mp3;\*.mpa;" +

"\*.mpe;\*.m3u;\*.mp4;\*.mov;\*.3g2;\*.3gp2;\*.3gp;\*.3gpp;\*.m4a;" +

"\*.cda;\*.aif;\*.aifc;\*.aiff;\*.mid;\*.midi;\*.rmi;\*.mkv;\*.WAV;" +

"\*.AAC;\*.WMA;\*.WMV;\*.AVI;\*.MPG;\*.MPEG;\*.M1V;\*.MP2;\*.MP3;\*.MPA;" +

"\*.MPE;\*.M3U;\*.MP4;\*.MOV;\*.3G2;\*.3GP2;\*.3GP;\*.3GPP;\*.M4A;\*.CDA;" +

"\*.AIF;\*.AIFC;\*.AIFF;\*.MID;\*.MIDI;\*.RMI;\*.MKV"; ;

openFileDialog.InitialDirectory = "D:\\4 семестр\\РПС\\FilmsDataBase\\";

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

var filePath = openFileDialog.FileName;

Trailer = filePath;

}

}

private bool CanAddTrailerCommandExecute(object p) => true;

#endregion

#endregion

public AddFilmViewModel()

{

#region Commands

CloseApplicationCommand = new LambdaCommand(OnCloseApplicationCommandExecuted,

CanCloseApplicationCommandExecute);

SaveDataCommand = new LambdaCommand(OnSaveDataCommandExecuted,

CanSaveDataCommandExecute);

AddIconCommand = new LambdaCommand(OnAddIconCommandExecuted,

CanAddIconCommandExecute);

AddTrailerCommand = new LambdaCommand(OnAddTrailerCommandExecuted,

CanAddTrailerCommandExecute);

#endregion

}

}

}

[AddFilmViewModel]

[GreetingsWindowViewModel]

using System.Windows.Input;

using FilmsDataBase.ViewModels.Base;

using FilmsDataBase.Infrastructure.Commands;

using System.Collections.ObjectModel;

using System.IO;

using System;

namespace FilmsDataBase.ViewModels

{

public class GreetingsWindowViewModel : ViewModel

{

#region Properties

private bool \_isChecked;

public bool IsChecked { get => \_isChecked; set => Set(ref \_isChecked, value); }

private DisplayRootRegistry \_displayRootRegistry;

public DisplayRootRegistry DisplayRootRegistry

{

set { \_displayRootRegistry = value; }

get { return \_displayRootRegistry; }

}

#endregion

#region Commands

#region CloseApplicationCommand

public ICommand CloseApplicationCommand { get; }

private void OnCloseApplicationCommandExecuted(object p)

{

StreamWriter file = new StreamWriter("Agreement.txt");

file.WriteLine((IsChecked.ToString()));

file.Close();

DisplayRootRegistry.HidePresentation(this);

}

private bool CanCloseApplicationCommandExecute(object p) => true;

#endregion

#endregion

public GreetingsWindowViewModel()

{

#region Commands

CloseApplicationCommand = new LambdaCommand(OnCloseApplicationCommandExecuted,

CanCloseApplicationCommandExecute);

#endregion

}

}

}

[GreetingsWindowViewModel]

[InformationAboutFilmViewModel]

using System.Windows.Input;

using FilmsDataBase.ViewModels.Base;

using FilmsDataBase.Infrastructure.Commands;

namespace FilmsDataBase.ViewModels

{

public class InformationAboutFilmWindowViewModel : ViewModel

{

#region Properties

private DisplayRootRegistry \_displayRootRegistry;

public DisplayRootRegistry DisplayRootRegistry

{

set { \_displayRootRegistry = value; }

get { return \_displayRootRegistry; }

}

private string \_title;

public string Title { get => \_title; set => Set(ref \_title, value); }

private string \_description;

public string Description { get => \_description; set => Set(ref \_description,

value); }

private string \_icon;

public string Icon { get => \_icon; set => Set(ref \_icon, value); }

private string \_trailer;

public string Trailer { get => \_trailer; set => Set(ref \_trailer, value); }

private System.DateTime \_year;

public System.DateTime Year { get => \_year; set => Set(ref \_year, value); }

//private int \_intYear;

/\*public int IntYear { get => \_intYear; set => Set(ref \_intYear, value); }\*/

private string \_oldTitle;

public string OldTitle { get => \_oldTitle; set => Set(ref \_oldTitle, value); }

#endregion

#region Commands

#region CloseApplicationCommand

public ICommand CloseApplicationCommand { get; }

private void OnCloseApplicationCommandExecuted(object p)

{

DisplayRootRegistry.HidePresentation(this);

}

private bool CanCloseApplicationCommandExecute(object p) => true;

#endregion

#endregion

public InformationAboutFilmWindowViewModel()

{

#region Commands

CloseApplicationCommand = new LambdaCommand(OnCloseApplicationCommandExecuted,

CanCloseApplicationCommandExecute);

#endregion

}

}

}

[InformationAboutFilmViewModel]

[MainWindowViewModel]

using FilmsDataBase.ViewModels.Base;

using System.Windows.Input;

using FilmsDataBase.Infrastructure.Commands;

using System.Windows;

using FilmsDataBase.Models;

using System.Windows.Forms;

using System.Collections.ObjectModel;

using System.IO;

using System;

using FilmsDataBase.Infrastructure;

using System.Collections.Generic;

using Autofac;

using FilmsDataBase.Data;

using FilmsDataBase.Services;

namespace FilmsDataBase.ViewModels

{

public class MainWindowViewModel : ViewModel

{

#region Autofac

private static IFilmService \_concentrationService = null;

public MainWindowViewModel(IFilmService concentrationService) =>

\_concentrationService = concentrationService ?? throw new

ArgumentNullException(nameof(concentrationService));

#endregion

private void NotificationFileUpdated()

{

System.Windows.MessageBox.Show(

"File updated",

"Notification",

(MessageBoxButton)MessageBoxButtons.OK,

(MessageBoxImage)MessageBoxIcon.Information,

(MessageBoxResult)MessageBoxDefaultButton.Button1,

System.Windows.MessageBoxOptions.DefaultDesktopOnly);

}

#region SelectedWindow

private DisplayRootRegistry displayRootRegistry;

private AddFilmViewModel addFilmViewModel;

private InformationAboutFilmWindowViewModel informationAboutFilmWindowViewModel;

private GreetingsWindowViewModel greetingsWindowViewModel;

#endregion

#region Properties

#region Film

private ObservableCollection<Film> \_films;

public ObservableCollection<Film> Films { get => \_films; set => Set(ref \_films,

value); }

#endregion

#region WhichFilm

private Film \_whichFilm;

public Film WhichFilm { get => \_whichFilm; set => Set(ref \_whichFilm, value); }

#endregion

#endregion

#region Commands

#region InformationAboutFilmCommand

public ICommand InformationAboutFilmCommand { get; }

private bool CanInformationAboutFilmCommandExecute(object p) => WhichFilm !=

null && !\_concentrationService.IsEmpty() &&

!displayRootRegistry.CheckForExistingWindows(informationAboutFilmWindowViewMo

del);

private void OnInformationAboutFilmCommandExecuted(object p)

{

if (informationAboutFilmWindowViewModel == null)

informationAboutFilmWindowViewModel = new

InformationAboutFilmWindowViewModel();

informationAboutFilmWindowViewModel.Title = WhichFilm.Title;

informationAboutFilmWindowViewModel.Description = WhichFilm.Description;

informationAboutFilmWindowViewModel.Icon = WhichFilm.Icon;

informationAboutFilmWindowViewModel.Trailer = WhichFilm.Trailer;

informationAboutFilmWindowViewModel.Year = WhichFilm.Year;

informationAboutFilmWindowViewModel.DisplayRootRegistry = displayRootRegistry;

displayRootRegistry.ShowPresentation(informationAboutFilmWindowViewModel);

}

#endregion

#region OpenInnerWindowCommand

public ICommand OpenInnerWindowCommand { get; }

private bool CanOpenInnerWindowCommandExecute(object p) =>

!displayRootRegistry.CheckForExistingWindows(addFilmViewModel);

private void OnOpenInnerWindowCommandExecuted(object p)

{

if (addFilmViewModel == null)

addFilmViewModel = new AddFilmViewModel();

addFilmViewModel.OldTitle = "";

addFilmViewModel.Title = "";

addFilmViewModel.Description = "";

addFilmViewModel.Icon = "";

addFilmViewModel.Trailer = "";

addFilmViewModel.Year = new DateTime();

addFilmViewModel.id = new int();

addFilmViewModel.DisplayRootRegistry = displayRootRegistry;

displayRootRegistry.ShowPresentation(addFilmViewModel);

}

#endregion

#region OpenGreetingWindowCommand

public ICommand OpenGreetingWindowCommand { get; }

private bool CanOpenGreetingWindowCommandExecute(object p) =>

!displayRootRegistry.CheckForExistingWindows(greetingsWindowViewModel);

public void OnOpenGreetingWindowCommandExxecuted(object p)

{

bool Agreement;

FileStream createFile = null;

StreamReader file = null;

try

{

file = new StreamReader("Agreement.txt");

}

catch

{

createFile = File.Create("Agreement.txt");

createFile.Close();

file = new StreamReader("Agreement.txt");

}

Agreement = false;

if (file != null)

{

string yesOrNo = file.ReadLine();

if (yesOrNo != null)

Agreement = bool.Parse(yesOrNo);

file.Close();

}

if(greetingsWindowViewModel == null)

greetingsWindowViewModel = new GreetingsWindowViewModel();

if (Agreement)

greetingsWindowViewModel.IsChecked = true;

else

greetingsWindowViewModel.IsChecked = false;

if (Convert.ToBoolean(p))

{

greetingsWindowViewModel.DisplayRootRegistry = displayRootRegistry;

displayRootRegistry.ShowPresentation(greetingsWindowViewModel);

}

else

{

if (!Agreement)

{

greetingsWindowViewModel.DisplayRootRegistry = displayRootRegistry;

displayRootRegistry.ShowPresentation(greetingsWindowViewModel);

}

}

}

#endregion

#region EditFilmCommand

public ICommand EditFilmCommand { get; }

private bool CanEditFilmCommandExecute(object p) => WhichFilm != null &&

!\_concentrationService.IsEmpty() &&

!displayRootRegistry.CheckForExistingWindows(addFilmViewModel);

private void OnEditFilmCommandExecuted(object p)

{

if(addFilmViewModel == null)

addFilmViewModel = new AddFilmViewModel();

addFilmViewModel.OldTitle = WhichFilm.Title;

addFilmViewModel.id = WhichFilm.Id;

addFilmViewModel.Title = WhichFilm.Title;

addFilmViewModel.Description = WhichFilm.Description;

addFilmViewModel.Icon = WhichFilm.Icon;

addFilmViewModel.Trailer = WhichFilm.Trailer;

addFilmViewModel.Year = WhichFilm.Year;

addFilmViewModel.DisplayRootRegistry = displayRootRegistry;

displayRootRegistry.ShowPresentation(addFilmViewModel);

}

#endregion

#region CloseApplicationCommand

public ICommand CloseApplicationCommand { get;}

private void OnCloseApplicationCommandExecuted(object p)

{

System.Windows.Application.Current.Shutdown();

}

private bool CanCloseApplicationCommandExecute(object p) =>

!displayRootRegistry.CheckForExistingWindows(addFilmViewModel) &&

!displayRootRegistry.CheckForExistingWindows(informationAboutFilmWindowViewMo

del) &&

!displayRootRegistry.CheckForExistingWindows(greetingsWindowViewModel);

#endregion

#region RefreshDataBaseCommand

public ICommand RefreshDataBaseCommand { get; }

private bool CanRefreshDataBaseCommandExecute(object p) =>

!\_concentrationService.IsEmpty();

private void OnRefreshDataBaseCommandExecuted(object p)

{

Films.Clear();

List<Film> someFilms = \_concentrationService.GetData();

for (int i = 0; i < someFilms.Count; i++)

{

Films.Add((Film)someFilms[i].Clone());

}

}

#endregion

#region DeleteDataFromDataBaseCommand

public ICommand DeleteDataFromDataBaseCommand { get; }

private bool CanDeleteDataFromDataBaseCommandExecute(object p) => WhichFilm !=

null;

private void OnDeleteDataFromDataBaseCommandExecuted(object p)

{

\_concentrationService.DeleteData(WhichFilm.Id);

NotificationFileUpdated();

}

#endregion

#region SaveToFileCommand

public ICommand SaveToFileCommand { get; }

private bool CanSaveToFileCommandExecute(object p) =>

!\_concentrationService.IsEmpty();

private void OnSaveToFileCommandExecuted(object p)

{

\_concentrationService.SaveToFile();

}

#endregion

#endregion

public MainWindowViewModel()

{

Films = new ObservableCollection<Film>();

displayRootRegistry = (System.Windows.Application.Current as

App).displayRootRegistry;

if (!\_concentrationService.IsEmpty())

{

List<Film> someFilms = \_concentrationService.GetData();

for(int i = 0; i < someFilms.Count; i++)

{

Films.Add((Film)someFilms[i].Clone());

}

}

OnOpenGreetingWindowCommandExxecuted(false);

#region Commands

CloseApplicationCommand = new LambdaCommand(OnCloseApplicationCommandExecuted,

CanCloseApplicationCommandExecute);

OpenInnerWindowCommand = new LambdaCommand(OnOpenInnerWindowCommandExecuted,

CanOpenInnerWindowCommandExecute);

RefreshDataBaseCommand = new LambdaCommand(OnRefreshDataBaseCommandExecuted,

CanRefreshDataBaseCommandExecute);

DeleteDataFromDataBaseCommand = new

LambdaCommand(OnDeleteDataFromDataBaseCommandExecuted,

CanDeleteDataFromDataBaseCommandExecute);

EditFilmCommand = new LambdaCommand(OnEditFilmCommandExecuted,

CanEditFilmCommandExecute);

InformationAboutFilmCommand = new

LambdaCommand(OnInformationAboutFilmCommandExecuted, CanInformationAboutFilmCommandExecute);

OpenGreetingWindowCommand = new

LambdaCommand(OnOpenGreetingWindowCommandExxecuted,

CanOpenGreetingWindowCommandExecute);

SaveToFileCommand = new LambdaCommand(OnSaveToFileCommandExecuted,

CanSaveToFileCommandExecute);

#endregion

}

}

}

[MainWindowViewModel]

[AddFilmWindow.xaml]

<Window x:Class="FilmsDataBase.Views.Windows.AddFilmWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:FilmsDataBase.Views.Windows"

xmlns:vm="clr-namespace:FilmsDataBase.ViewModels"

mc:Ignorable="d"

Title="AddFilmWindow" Height="480" Width="800">

<Window.DataContext>

<vm:AddFilmViewModel/>

</Window.DataContext>

<Window.InputBindings>

<KeyBinding Command="{Binding SaveDataCommand}" Gesture="Ctrl + S"/>

<KeyBinding Command="{Binding CloseApplicationCommand}" Gesture="ESC"/>

</Window.InputBindings>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition />

<RowDefinition />

<RowDefinition />

</Grid.RowDefinitions>

<Label Content="Title to film:" Height="40" Width="80"

HorizontalAlignment="Left" Margin="0, -100, 0, 0"/>

<TextBox Margin="80,5,10,120" Grid.ColumnSpan="3" Text="{Binding Path=Title,

UpdateSourceTrigger=PropertyChanged}"/>

<Label Content="Description:" Height="40" Width="80"

HorizontalAlignment="Left" Margin="0, -40, 0, 0"/>

<TextBox Margin="80,40,10,85" Grid.ColumnSpan="3" Text="{Binding

Path=Description, UpdateSourceTrigger=PropertyChanged}"/>

<Label Content="Image:" Height="40" Width="80" HorizontalAlignment="Left"

Margin="0, 40, 0, 0"/>

<TextBox Margin="80,80,10,45" Grid.ColumnSpan="3" Text="{Binding Path=Icon,

UpdateSourceTrigger=PropertyChanged, Mode=TwoWay}"/>

<Label Content="Trailer:" Height="40" Width="80" HorizontalAlignment="Left"

Margin="0, 120, 0, 0"/>

<TextBox Margin="80,120,10,0" Grid.ColumnSpan="3" Text="{Binding

Path=Trailer, UpdateSourceTrigger=PropertyChanged, Mode=TwoWay}"/>

<Label Content="Year:" Height="40" Width="80" HorizontalAlignment="Left"

VerticalAlignment="Top" Margin="0, 0, 0, 0" Grid.Row="1"/>

<TextBox Margin="80,5,10,120" Grid.ColumnSpan="3" Grid.Row="1"

Text="{Binding Path=Year, UpdateSourceTrigger=PropertyChanged}"/>

<Button Content="Save" Grid.Column="1" Grid.Row="2" Margin="20, 120, 180, 5"

Command="{Binding SaveDataCommand}"

CommandParameter=""/>

<Button Height="20" Width="20" VerticalAlignment="Center"

HorizontalAlignment="Left" Margin="50, 35, 0, 0" Content="+"

Command="{Binding AddIconCommand}"/>

<Button Height="20" Width="20" VerticalAlignment="Center"

HorizontalAlignment="Left" Margin="50, 115, 0, 0" Content="+"

Command="{Binding AddTrailerCommand}"/>

<Button Content="Return" Grid.Column="1" Grid.Row="2" Margin="180, 120, 20,

5"

Command="{Binding CloseApplicationCommand}"

CommandParameter="{Binding RelativeSource={RelativeSource

AncestorType={x:Type Window}}}"/>

<MediaElement x:Name="Video" Grid.Column="0" Grid.Row="1" Grid.RowSpan="2"

Grid.ColumnSpan="2" Margin="10, 50, 10, 50" Source="{Binding Trailer}"

LoadedBehavior="Manual"

MouseLeftButtonDown="MediaElement\_MouseLeftButtonDown"/>

<Image Grid.Column="2" Grid.Row="1" Grid.RowSpan="2" Margin="10, 50, 10, 50"

Source="{Binding Icon}"/>

<Button Grid.Column="0" Grid.Row="1" Height="20" Width="20"

HorizontalAlignment="Left" Margin="50, -120, 0, 0" Content="+"

Click="Button\_Click"/>

<Calendar Grid.Row="0" Grid.Column="0" Loaded="Button\_Click"

x:Name="Calendar" Margin="0, 70, 450, 0" SelectedDate="{Binding Year}"

HorizontalAlignment="Center" Grid.ColumnSpan="3" Grid.RowSpan="3"

VerticalAlignment="Center"/>

</Grid>

</Window>

[AddFilmWindow.xaml]

[GreetingsWindow.xaml]

<Window x:Class="FilmsDataBase.Views.Windows.GreetingsWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:vm="clr-namespace:FilmsDataBase.ViewModels"

xmlns:local="clr-namespace:FilmsDataBase.Views.Windows"

mc:Ignorable="d"

Title="GreetingsWindow" Height="250" Width="450">

<Window.DataContext>

<vm:GreetingsWindowViewModel/>

</Window.DataContext>

<Window.InputBindings>

<KeyBinding Command="{Binding CloseApplicationCommand}" Gesture="ESC"/>

</Window.InputBindings>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="3\*"/>

<RowDefinition/>

</Grid.RowDefinitions>

<TextBlock Grid.Row="0" Text=" It is necessary to perform a task from the options of the usual complexity using

the MVC pattern (or MVVM if WPF is used) and the Autofac library for

dependency injection. It is necessary to use asynchronous connection to the DBMS

(SQLiteAsyncConnection)" TextWrapping="Wrap" TextAlignment="Justify" Margin="5"/>

<TextBlock Grid.Row="0" Margin="5, 70, 5, 0" Text=" The program should allow you to add new entities

using the interface and edit existing ones. Entities added to the

program must be saved between application launches. To store data,

you need to use the SQLite DBMS. It is necessary to provide for the possibility

of saving a list of existing entities to a file." TextWrapping="Wrap"

TextAlignment="Justify"/>

<CheckBox x:Name="CheckBox" Grid.Row="1" Height="20" Width="20"

HorizontalAlignment="Left" VerticalAlignment="Center" Margin="50, 0, 0,

0"

IsChecked="{Binding IsChecked, Mode=TwoWay}"/>

<TextBlock Grid.Row="1" Margin="80, 19, 0, 10" Text="Check if you don't want to see this again"/>

<Button Grid.Row="1" Height="25" Width="35" HorizontalAlignment="Right"

VerticalAlignment="Center" Margin="0, 0, 50, 0" Content="OK"

Command="{Binding CloseApplicationCommand}"/>

</Grid>

</Window>

[GreetingsWindow.xaml]

[InformationAboutFilmViewModel.xaml]

<Window x:Class="FilmsDataBase.Views.Windows.InformationAboutFilmWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:FilmsDataBase.Views.Windows"

xmlns:vm="clr-namespace:FilmsDataBase.ViewModels"

mc:Ignorable="d"

Title="InformationAboutFilmWindow" Height="450" Width="800">

<Window.DataContext>

<vm:InformationAboutFilmWindowViewModel/>

</Window.DataContext>

<Window.InputBindings>

<KeyBinding Command="{Binding CloseApplicationCommand}" Gesture="ESC"/>

</Window.InputBindings>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

</Grid.RowDefinitions>

<TextBlock Text = "{Binding Title}" TextWrapping="Wrap" Grid.Column="0"

Grid.Row="0" Margin="10, 60, 10, 40" TextAlignment="Center"/>

<TextBlock Text = "{Binding Year}" Margin="0, 100, 0, 0"

HorizontalAlignment="Center" Grid.Row="0"/>

<TextBlock Text = "{Binding Description}" TextWrapping="Wrap"

Grid.Column="1" Grid.Row="0" Margin="10, 20, 10, 10"

HorizontalAlignment="Center" TextAlignment="Justify"/>

<Image Source = "{Binding Icon}" Margin="10" Grid.Column="3"

Grid.RowSpan="3"/>

<MediaElement x:Name="Video" Source = "{Binding Trailer}" Grid.Row="1"

Grid.ColumnSpan="2" LoadedBehavior="Manual" Margin="0, 0, 0, 25"

Grid.RowSpan="2" MouseLeftButtonDown="Video\_MouseLeftButtonDown"/>

</Grid>

</Window>

[InformationAboutFilmViewModel.xaml]

[MainWindow.xaml]

<Window x:Class="FilmsDataBase.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:vm="clr-namespace:FilmsDataBase.ViewModels"

Title="🐯📺🐵" Height="450" Width="645"

Background="#9370DB"

Icon="D:\4 семестр\РПС\FilmsDataBase\Violet-smile.ico"

MinHeight="450"

MaxHeight="450"

MinWidth="750">

<Window.DataContext>

<vm:MainWindowViewModel/>

</Window.DataContext>

<Window.InputBindings>

<KeyBinding Command="{Binding RefreshDataBaseCommand}" Gesture="F5"/>

<KeyBinding Command="{Binding OpenInnerWindowCommand}" Gesture="Ctrl + O"/>

<KeyBinding Command="{Binding DeleteDataFromDataBaseCommand}" Gesture="Ctrl

+ Delete"/>

<KeyBinding Command="{Binding CloseApplicationCommand}" Gesture="ESC"/>

<KeyBinding Command="{Binding EditFilmCommand}" Gesture="Ctrl + E"/>

<KeyBinding Command="{Binding InformationAboutFilmCommand}" Gesture="Ctrl +

I"/>

<KeyBinding Command="{Binding OpenGreetingWindowCommand}" Gesture="Ctrl + G"

CommandParameter="true"/>

<KeyBinding Command="{Binding SaveToFileCommand}" Gesture="Ctrl + S"/>

</Window.InputBindings>

<Grid>

<Grid.Resources>

<DataTemplate x:Key="Icon">

<StackPanel Width="80" Height="120">

<Image Source="{Binding Icon}"/>

</StackPanel>

</DataTemplate>

<DataTemplate x:Key="Trailer">

<StackPanel Width="240" Height="120">

<MediaElement Source="{Binding Trailer}"

LoadedBehavior="Pause"/>

</StackPanel>

</DataTemplate>

</Grid.Resources>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="\*"/>

<RowDefinition Height="\*"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<Menu Grid.Row="0" Background="#9370DB">

<MenuItem Header="File">

<MenuItem Header="Add film" Command="{Binding

OpenInnerWindowCommand}"/>

<MenuItem Header="Delete film" Command="{Binding

DeleteDataFromDataBaseCommand}"/>

<MenuItem Header="Edit film" Command="{Binding EditFilmCommand}"/>

<MenuItem Header="Exit" Command="{Binding CloseApplicationCommand}"

CommandParameter="{Binding RelativeSource={RelativeSource

AncestorType={x:Type Window}}}"/>

<MenuItem Header="Refresh" Command="{Binding

RefreshDataBaseCommand}"/>

<MenuItem Header="Information about film" Command="{Binding

InformationAboutFilmCommand}"/>

<MenuItem Header="Save to file" Command="{Binding

SaveToFileCommand}"/>

</MenuItem>

<MenuItem Header="Information" Command="{Binding

OpenGreetingWindowCommand}" CommandParameter="true"/>

</Menu>

<DataGrid ItemsSource="{Binding Films, UpdateSourceTrigger=PropertyChanged,

Mode=TwoWay}" Grid.ColumnSpan="3" Grid.RowSpan="3" Margin="0, 20, 0, 0" AutoGenerateColumns="False"

CanUserAddRows="False" CanUserReorderColumns="False"

CanUserDeleteRows="False" SelectedItem="{Binding WhichFilm,

Mode=TwoWay}" IsReadOnly="True"

Background="#cbb6ee">

<DataGrid.Columns>

<DataGridTextColumn Header = "Ttile" Width="100" Binding="{Binding

Title}" />

<DataGridTextColumn Header = "Description" Width="140"

Binding="{Binding Description}"/>

<DataGridTemplateColumn Header = "Icon" Width="auto"

CellTemplate="{StaticResource Icon}"/>

<DataGridTemplateColumn Header = "Trailer" Width="auto"

CellTemplate="{StaticResource Trailer}"/>

<DataGridTextColumn Header = "Year" Width="auto" Binding="{Binding

Year}"/>

</DataGrid.Columns>

</DataGrid>

</Grid>

</Window>

[MainWindow.xaml]

[DisplayRootRegistry]

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using System.Windows;

namespace FilmsDataBase

{

public class DisplayRootRegistry

{

private Dictionary<Type, Type> vmToWindowMapping = new Dictionary<Type, Type>();

public void RegisterWindowType<VM, Win>() where Win : Window, new() where VM : class

{

var vmType = typeof(VM);

if (vmType.IsInterface)

throw new ArgumentException("Cannot register interfaces");

if (vmToWindowMapping.ContainsKey(vmType))

throw new InvalidOperationException(

$"Type {vmType.FullName} is already registered");

vmToWindowMapping[vmType] = typeof(Win);

}

public bool CheckForExistingWindows(object vm)

{

if (vm == null) return false;

if (openWindows.ContainsKey(vm)) return true;

return false;

}

public void UnregisterWindowType<VM>()

{

var vmType = typeof(VM);

if (vmType.IsInterface)

throw new ArgumentException("Cannot register interfaces");

if (!vmToWindowMapping.ContainsKey(vmType))

throw new InvalidOperationException(

$"Type {vmType.FullName} is not registered");

vmToWindowMapping.Remove(vmType);

}

public Window CreateWindowInstanceWithVM(object vm)

{

if (vm == null)

throw new ArgumentNullException("vm");

Type windowType = null;

var vmType = vm.GetType();

while (vmType != null && !vmToWindowMapping.TryGetValue(vmType, out

windowType))

vmType = vmType.BaseType;

if (windowType == null)

throw new ArgumentException(

$"No registered window type for argument type {vm.GetType().FullName}");

var window = (Window)Activator.CreateInstance(windowType);

window.DataContext = vm;

return window;

}

private Dictionary<object, Window> openWindows = new Dictionary<object,

Window>();

public void ShowPresentation(object vm)

{

if (vm == null)

throw new ArgumentNullException("vm");

if (openWindows.ContainsKey(vm))

throw new InvalidOperationException("UI for this VM is already displayed");

var window = CreateWindowInstanceWithVM(vm);

window.Show();

openWindows[vm] = window;

}

public void HidePresentation(object vm)

{

Window window;

if (!openWindows.TryGetValue(vm, out window))

throw new InvalidOperationException("UI for this VM is not displayed");

window.Close();

openWindows.Remove(vm);

}

public async Task ShowModalPresentation(object vm)

{

var window = CreateWindowInstanceWithVM(vm);

window.WindowStartupLocation = WindowStartupLocation.CenterScreen;

await window.Dispatcher.InvokeAsync(() => window.ShowDialog());

}

}

}

[DisplayRootRegistry]

[ --- Конец программы]

# Документирование и комментирование исходного текста