

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
учреждение образования «Гродненский государственный университет
имени Янки Купалы»

Факультет математики и информатики
Кафедра современных технологий программирования

Сакович Артём Сергеевич

Разработка каталогизатора продуктов киноиндустрии

Курсовая работа
по дисциплине «Языки программирования»
студента 2 курса специальности
1-40 01 01 «Программное обеспечение информационных технологий»
дневной формы получения образования

Научный руководитель

Карканица Анна Викторовна,
доцент кафедры
современных технологий
программирования,
кандидат технических наук

Гродно, 2020

РЕЗЮМЕ

Сакович Артём Сергеевич

Курсовая работа – «Разработка каталогизатора продуктов киноиндустрии», 37 страниц, 37 иллюстраций, 5 использованных источников.

Ключевые слова: каталог, фильм, жанр, пользователь, фильмотека, с#, wpf, xml.

Цель исследования – разработка приложения, позволяющего просматривать содержимое каталога продуктов киноиндустрии, а также осуществлять добавление, поиск и редактирование информации о конкретном продукте.

Объект исследования: приложения на языке С#.

Предмет исследования: каталогизаторы продуктов киноиндустрии.

Методы исследования: анализ, синтез, системный подход.

Данное приложение разработано с целью предоставить пользователю возможность получить информацию о продукте киноиндустрии. Оно позволяет пользователям добавлять, удалять и редактировать содержимое каталога.

Данное приложение представляет собой каталог продуктов киноиндустрии. Оно было написано на языке программирования С# с использованием технологии Windows Presentation Foundation для более удобной и привлекательной реализации пользовательского интерфейса.

SUMMARY

Artem Sakovich

Course work - "Development of a catalogue of film industry products", 26 pages, 12 illustrations, 6 sources used.

Key words: catalog, film, genre, user, film library, c #, wpf, xml.

The purpose of the study is to develop an application that simulates the work of a travel agency and allows you to view current offers and book tours.

Object of study: C# applications.

Subject of research: film catalogers, film libraries.

Research methods: analysis, synthesis, systems approach.

This application is designed to provide an opportunity to get information about the movie. It allows users to delete and edit catalog products.

This application is a catalog of the film industry products. It was written in the C # programming language using Windows Presentation Foundation technology for a more convenient and attractive user interface implementation.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ	5
ВВЕДЕНИЕ.....	6
ГЛАВА 1. ОБЗОР ПРИЛОЖЕНИЙ, ПРЕДНАЗНАЧЕННЫХ ДЛЯ КАТАЛОГИЗАЦИИ ПРОДУКТОВ КИНОИНДУСТРИИ.....	8
1.1 Обзор существующих решений.....	8
1.2 Видение продукта	9
1.3 Выводы по главе 1.....	10
ГЛАВА 2. ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ-КАТАЛОГИЗАТОРА ПРОДУКТОВ КИНОИНДУСТРИИ.....	12
2.1 Проектирование функциональной части приложения.....	12
2.2 Проектирование архитектуры приложения.....	12
2.3 Проектирование графического интерфейса приложения	13
2.4 Выводы по главе 2.....	18
ГЛАВА 3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ- КАТАЛОГИЗАТОРА ПРОДУКТОВ КИНОИНДУСТРИИ	19
3.1 Обзор средств реализации.....	19
3.2 Бизнес-сущности приложения	19
3.3 Формат хранения исходных данных	20
3.4 Программная реализация уровней архитектуры	20
3.5 Выводы по главе 3.....	36
ЗАКЛЮЧЕНИЕ	37
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	38

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

WPF (Windows Presentation Foundation) – аналог WinForms, система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем, графическая (презентационная) подсистема в составе . NET Framework (начиная с версии 3.0), использующая язык XAML.[\[3\]](#)

XML – eXtensible Markup Language, расширяемый язык разметки.[\[4\]](#)

XAML (eXtensible Application Markup Language) – расширяемый язык разметки для приложений, основанный на XML язык разметки для декларативного программирования приложений, разработанный Microsoft.

ВВЕДЕНИЕ

Минули времена, когда возможностей обычного файлового менеджера было вполне достаточно для удобной ориентации среди многочисленных папок и файлов. Современные компьютеры только на жестком диске, объем которого исчисляется десятками гигабайт, хранят невероятное количество разноплановой информации, где помимо обычных рабочих материалов фигурируют дистрибутивы, фотографии, музыка и пр. Кроме того, у каждого пользователя имеется собственная коллекция CD- и DVD-дисков, вполне возможно, что часть информации хранится на Zip-дисках и т.д. Очевидно, что какую бы удобную и разумную систему организации вложенных каталогов пользователь ни придумал, разобраться в такой массе информации просто нереально, тем более что коллекции быстро растут.

Такое положение дел означает, что на поиски какой-то программы, удачного изображения, подходящей к случаю мелодии и т.п. затрачивается немало времени, что неразумно. И хорошо еще, если искать, например, изображение придется только в определенных каталогах жесткого диска. А если нужно найти, скажем, фильм или редко используемую программу, то придется просматривать массу дисков, каждый раз вставляя их в CD-ROM, что займет много времени и на пользу дискам не пойдет. Вместе с тем решить проблему можно довольно просто — нужно воспользоваться подходящим каталогизатором. Основная задача каталогизаторов — упорядоченное хранение данных. Для пользователя это означает получение возможности быстрого поиска нужной информации, причем в случае нахождения данных на CD или DVD не придется даже доставать соответствующий диск. Параллельно с этим появляется очень простая возможность отслеживать диски, отданные на время друзьям или знакомым, и в любой момент быть в курсе того, кому был отдан диск и был ли он возвращен. Правда, все это предполагает предварительное заполнение каталога и внесение в него интересующей информации — нужно просканировать компакт-диски,

заполнить базу музыкальных файлов или изображений и т.п., но без этого не обойтись.

Цель работы: разработка приложения-каталогизатора продуктов киноиндустрии, позволяющего организовать упорядоченное хранение большой коллекции данных, доступ и редактирование коллекции.

Для достижения поставленной цели необходимо решить следующие задачи:

- выполнить обзор и анализ наиболее популярных программных решений, предоставляющих доступ к информации о продуктах и новинках киноиндустрии;
- сформулировать функциональные требования к разрабатываемому приложению-каталогизатору, учитывая наличие полезных функций существующих аналогов;
- определить структуру данных и формат их хранения;
- спроектировать структуру приложения, выделив основные уровни архитектуры: модели данных, доступ к данным внешнего файла, сервисы для работы с коллекцией фильмов, уровень представления;
- выполнить программную реализацию приложения средствами языка программирования C# и библиотеки Windows Presentation Foundation.

ГЛАВА 1.

ОБЗОР ПРИЛОЖЕНИЙ, ПРЕДНАЗНАЧЕННЫХ ДЛЯ КАТАЛОГИЗАЦИИ ПРОДУКТОВ КИНОИНДУСТРИИ

1.1 Обзор существующих решений

Каталог фильмов, сериалов и анимации ivi.ru

Данный сервис по поиску и просмотру продуктов киноиндустрии предоставляет возможность просматривать фильмы, сериалы от различных компаний и фильтровать их под желания пользователя ([рисунок 1.1.1](#)). При переходе на страницу конкретного продукта пользователь может ознакомиться с подробной информацией о нем, фото, рейтингом, отзывами и также просмотреть трейлер или фильм, предварительно оплатив просмотр.

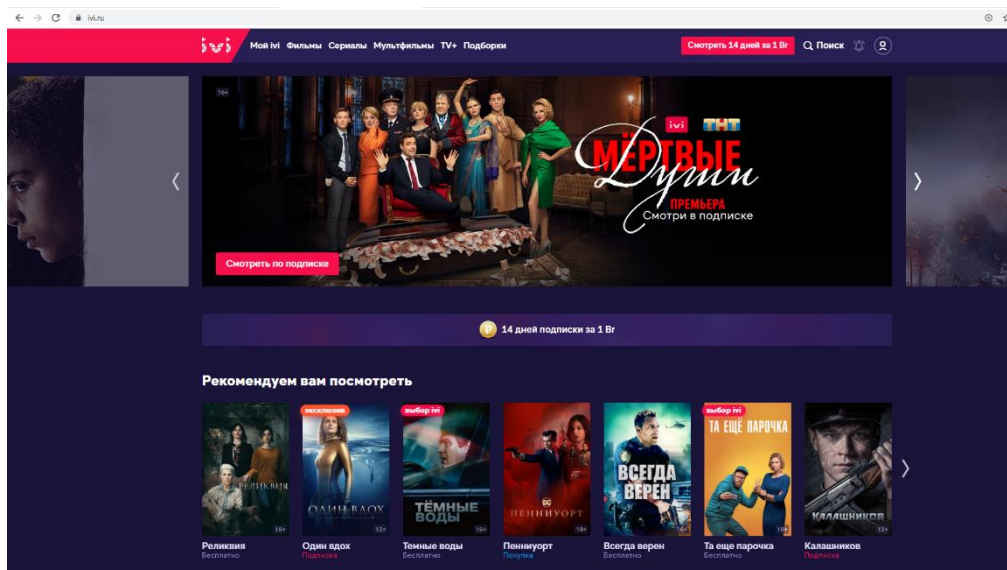


Рисунок 1.1.1 – скриншот сайта ivi.ru

К достоинствам данного web-сайта можно отнести простой интерфейс, широкий список критериев для поиска, наличие отзывов, рейтинга и полной информации о продукте, возможность выбора продукта по различным критериям. Основным недостатком является то, что просмотр фильмов не бесплатный.

Web-сайт imdb.com

Данный сервис, как и предыдущий, предоставляет возможность просматривать информацию о продуктах киноиндустрии от различных компаний и фильтровать их под желания пользователя ([рисунок 1.1.2](#)). Каждый продукт содержит полное описание и информацию о нём. Также есть возможность сохранять понравившиеся продукты и просматривать их на отдельной странице.

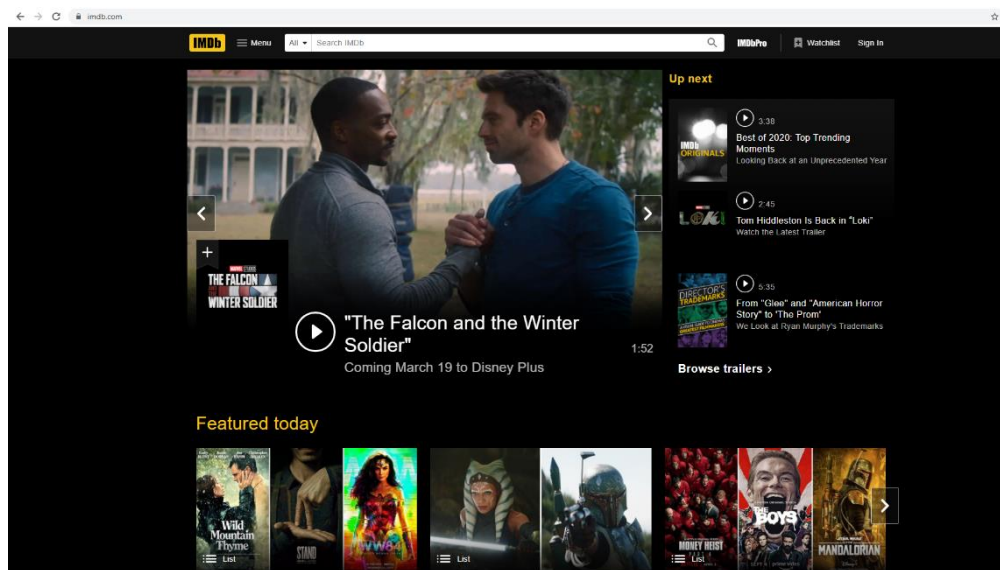


Рисунок 1.1.2 – скриншот сайта imdb.com

К достоинствам данного сервиса можно отнести большое количество параметров для поиска подходящего продукта и возможность сохранить заинтересовавшие продукты. Основными недостатками являются отсутствие возможности просмотреть заинтересовавший Вас продукт.

1.2 Видение продукта

В данной работе рассматривается система по поиску и просмотру информации о продуктах киноиндустрии. Хранение информации об объектах системы осуществляется с помощью XML файла. Такой подход актуален и оправдан в связи с быстротой работы и простотой использования.

Работа с приложением будет построена следующим образом: пользователь выбирает подходящий ему жанр или же смотрит популярные продукты, далее из предложенного списка выбирает понравившийся ему

продукт и по клику получает окно с полной информацией о продукте, постером, возможность просмотреть трейлер, возможностью редактирования информации о выбранном продукте и возможность удаления выбранного продукта из нашего каталога. Следует добавить, что если пользователь захочет внести какие-либо изменения в информацию о продукте, то после окончания он должен будет нажать кнопку “Save”, чтобы эти изменения произошли и в исходном файле.

Каждый продукт описывается следующим набором полей:

- название продукта;
- название компании производителя;
- год выпуска продукта;
- жанр продукта;
- длительность продукта в минутах;
- формат продукта;
- качество продукта в пикселях;
- постер;
- трейлер.

При работе с приложением пользователю предоставляются следующие возможности:

- добавлять новые продукты в каталог;
- редактировать информацию о продукте;
- удалить продукт из каталога;
- отсортировать каталог по выбранным критериям;
- расширенный поиск по каталогу.

1.3 Выводы по главе 1

Проанализировав существующие каталогизаторы продуктов киноиндустрии, можно выделить основные функции, которыми должно

обладать приложением подобного типа. Большинство представленных решений реализуют следующие базовые функции:

- просмотр каталога продуктов и информации о них;
- расширенный поиск по каталогу;
- сортировка по выбранным критериям.

Кроме того, были выявлены положительные и отрицательные стороны программ. Основным недостатком большинства веб-сайтов является неудобный пользовательский интерфейс и большое количество рекламы. Всё это позволит написать оптимальные требования к разрабатываемой программе.

К базовым требованиям к приложению с точки зрения пользователя относятся: возможность поиска продукта по критериям: название продукта, название компании, год выпуска продукта, длительность в минутах, формат, качество в пикселях. Кроме того, должно быть реализованы сортировки по всем вышеперечисленным критериям. В приложении также должна присутствовать реализация управления данными приложения.

ГЛАВА 2.

ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ-КАТАЛОГИЗАТОРА ПРОДУКТОВ КИНОИНДУСТРИИ

2.1 Проектирование функциональной части приложения

Прежде всего, следует отметить, что приложение будет реализовано с функционалом администратора, в дальнейшем от обычного пользователя реализацию управления данными приложения следует скрыть.

В приложении должны быть реализованы такие функции как:

- чтение всего перечня продуктов из XML файла;
- сериализация (сохранение данных в файл) XML файла;
- просмотр каталога продуктов по определённым категориям;
- просмотр информации о выбранном продукте;
- расширенный поиск по каталогу;
- сортировки по всем параметрам продукта;
- добавление нового продукта в каталог;
- удаление продукта из каталога;
- редактирование информации о продукте;
- просмотр трейлера к продукту.

2.2 Проектирование архитектуры приложения

С точки зрения архитектуры ([рисунок 2.2.1](#)), код приложения будет разделен на несколько основных уровней:

- модель — содержит класс, описывающий основные сущности предметной области (Film);

- сервис — содержит классы, отвечающие за реализацию методов, необходимых для работы с коллекцией фильмов (getFilms(), saveFilms(), addNewFilm() и т.д.);
- представление — отображение данных пользователю.

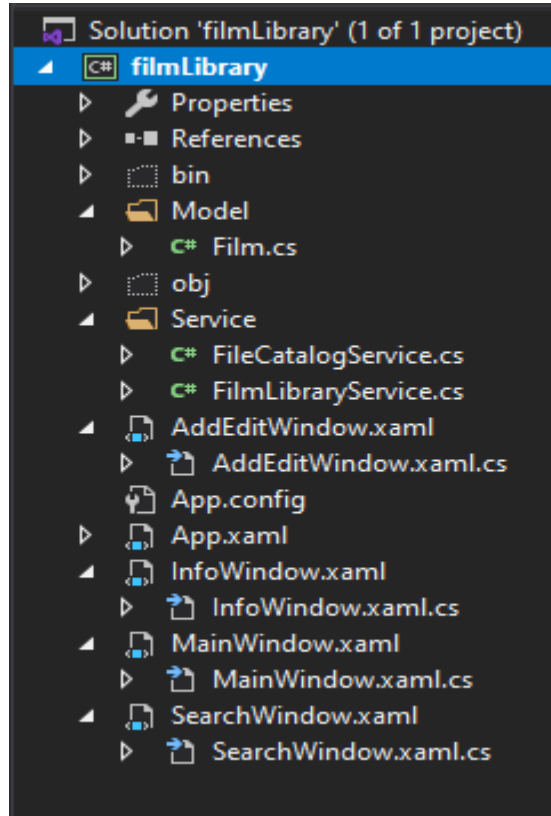


Рисунок 2.2.1 – архитектура проекта

Использование такой архитектуры дает следующие преимущества: чёткое разделение логики представления (интерфейса пользователя) и логики приложения, код получается гораздо более структурированным, и, тем самым, облегчается поддержка, тестирование и повторное использование решений.

2.3 Проектирование графического интерфейса приложения

Для данного приложения должен быть реализован интерфейс администратора с реализацией управления данными приложения. Должно быть реализовано главное окно ([рисунок 2.3.1](#)) с отображением на нём каталога продуктов (изначально популярных). Должен быть реализован выбор по жанрам и перерисовка главного окна под выбранный пользователем список продуктов. Так же должны присутствовать кнопки различных сортировок,

добавления продукта в каталог и быстрый поиск по названию продукта.

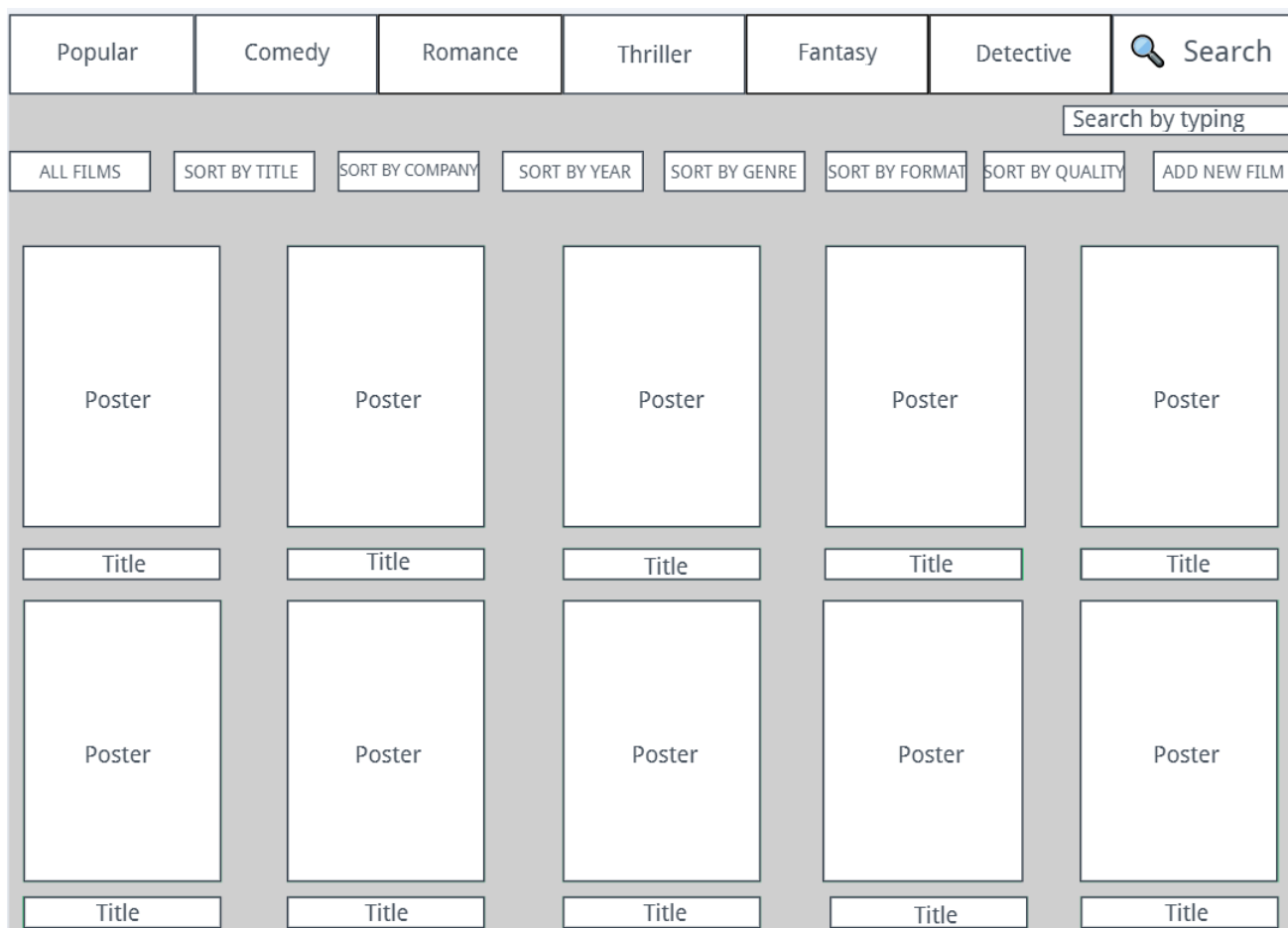


Рисунок 2.3.1 – мокап главного окна приложения

Так же должно быть реализовано окно с информацией о продукте ([рисунок 2.3.2](#)), которое отображается при нажатии пользователем на один из продуктов. В этом окне должна быть показана вся информация о выбранном продукте, и так же должны быть реализованы кнопки удаления, редактирования информации и просмотра трейлера.

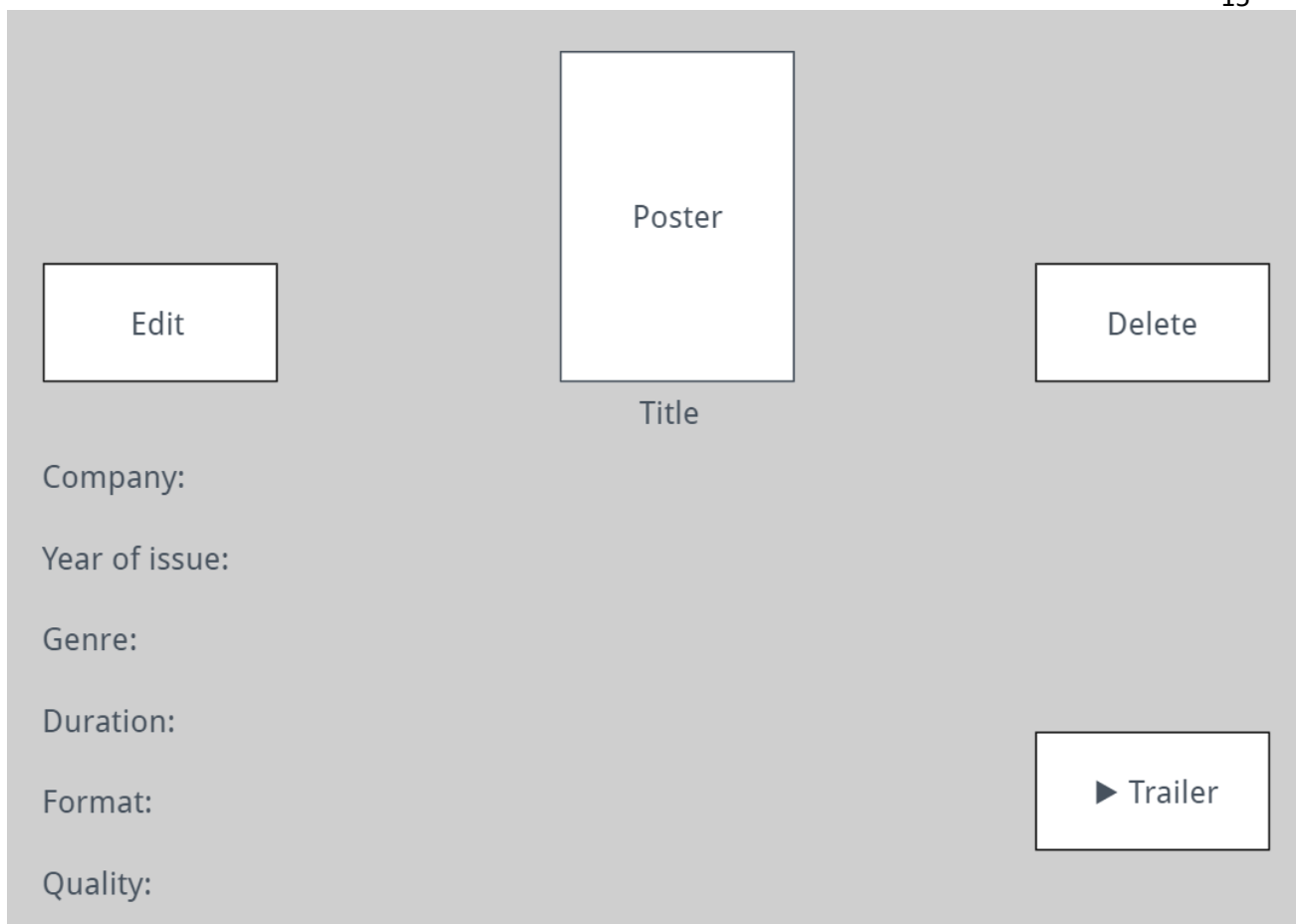
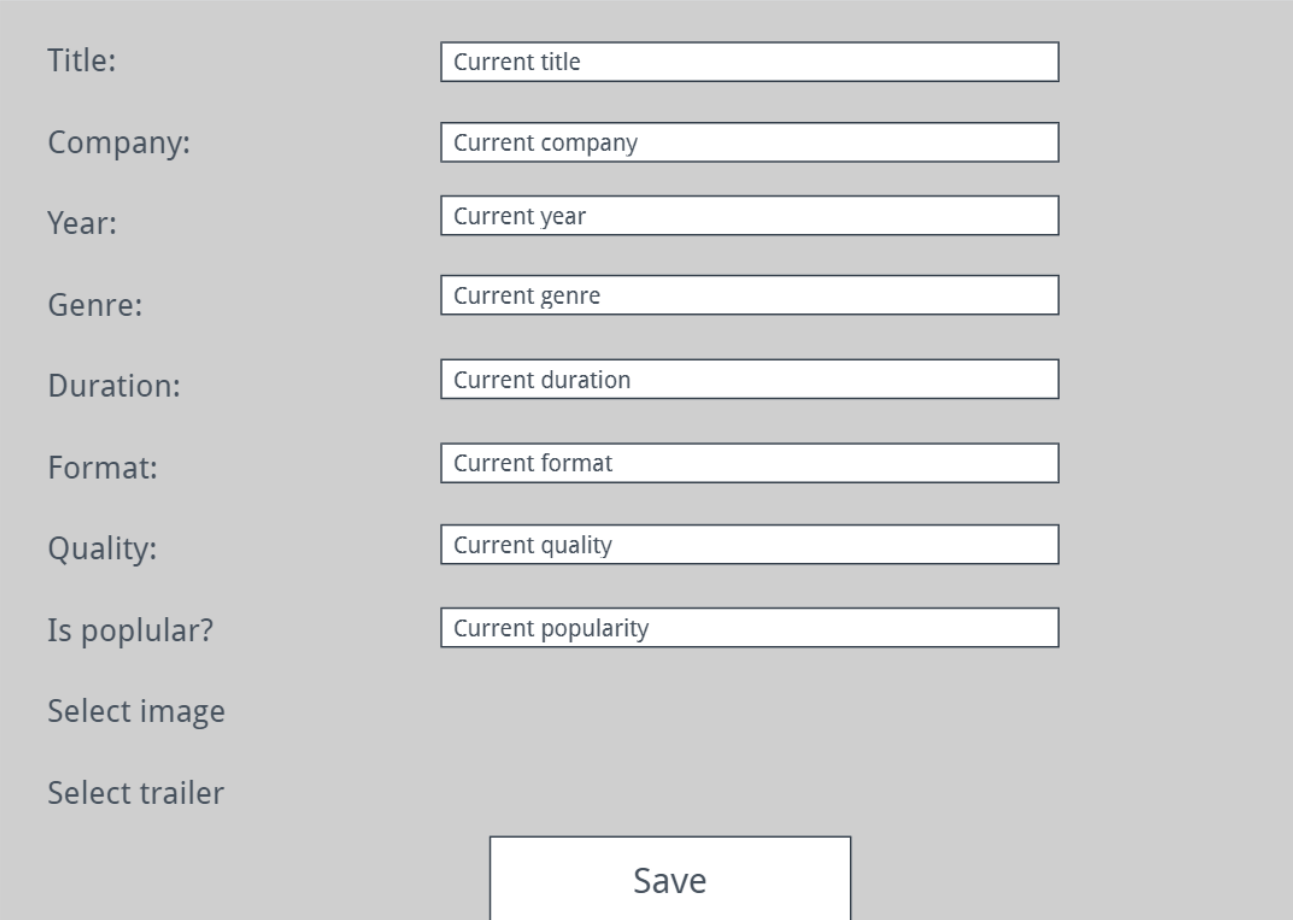


Рисунок 2.3.2 – мокап окна, содержащего информацию о продукте

Необходимо реализовать окно редактирования информации ([рисунок 2.3.3](#)), которое должно появляться при нажатии на соответствующую кнопку. В нем пользователю должна быть предоставлена текущая информация с возможностью изменить ее и сохранить изменения посредством кнопки. Следует отметить, что окно добавления продукта в каталог ([рисунок 2.3.4](#)) и окно редактирования информации ([рисунок 2.3.3](#)) были реализованы одним окном, так как имеют абсолютно идентичную структуру строения. Отличие лишь в том, что при редактировании информации пользователю предоставляется информация о выбранном продукте.



The mockup shows a product editing interface on a light gray background. It consists of several labels on the left and corresponding input fields on the right. The labels are: 'Title:', 'Company:', 'Year:', 'Genre:', 'Duration:', 'Format:', 'Quality:', 'Is poplular?', 'Select image', and 'Select trailer'. The input fields for the first seven labels contain the text 'Current title', 'Current company', 'Current year', 'Current genre', 'Current duration', 'Current format', and 'Current quality' respectively. The 'Is poplular?' label has an input field containing 'Current popularity'. Below these fields are two more labels, 'Select image' and 'Select trailer', which do not have input fields. At the bottom center is a 'Save' button.

Title:	<input type="text" value="Current title"/>
Company:	<input type="text" value="Current company"/>
Year:	<input type="text" value="Current year"/>
Genre:	<input type="text" value="Current genre"/>
Duration:	<input type="text" value="Current duration"/>
Format:	<input type="text" value="Current format"/>
Quality:	<input type="text" value="Current quality"/>
Is poplular?	<input type="text" value="Current popularity"/>
Select image	
Select trailer	
<input type="button" value="Save"/>	


Рисунок 2.3.3 – мокап окна редактирования информации о продукте

A mockup of a product addition window. It features a light gray background with a series of labels on the left and corresponding white input fields on the right. The labels are: 'Title:', 'Company:', 'Year:', 'Genre:', 'Duration:', 'Format:', 'Quality:', and 'Is poplular?'. Below these is a 'Select image' label with a small square icon, and a 'Select trailer' label with a play button icon. At the bottom center is a 'Save' button.

Title:	<input type="text"/>
Company:	<input type="text"/>
Year:	<input type="text"/>
Genre:	<input type="text"/>
Duration:	<input type="text"/>
Format:	<input type="text"/>
Quality:	<input type="text"/>
Is poplular?	<input type="text"/>
Select image	<input type="checkbox"/>
Select trailer	<input type="checkbox"/>
<input type="button" value="Save"/>	

Рисунок 2.3.4 – мокап окна добавления нового продукта в каталог

Так же должно быть реализовано окно расширенного поиска по критериям ([рисунок 2.3.5](#)), в котором пользователь, написав информацию в нужное поле, и нажав на соответствующую кнопку должен вернуться на главный экран и увидеть результаты поиска.



Search by title	<input type="text"/>
Search by company	<input type="text"/>
Search by year	<input type="text"/>
Search by duration	<input type="text"/>
Search by format	<input type="text"/>
Search by quality	<input type="text"/>

Рисунок 2.3.5 – мокап окна расширенного поиска

Каждая страница должна соответствовать общим требованиям, которым удовлетворяет удобный интерфейс, а именно: принцип структуризации, простоты, видимости, обратной связи, толерантности, повторного использования. Что касается анализа размещения информации на экране, во-первых, необходимые элементы отличаются большим размером. Во-вторых, никаких лишних элементов, только кликабельные.

2.4 Выводы по главе 2

Таким образом, в процессе проектирования были составлены требования к функциональным характеристикам.

Также была определена архитектура приложения и выделены преимущества в ее использовании, заключающиеся в улучшении структуризации кода, упрощении поддержки, отладки и тестирования. На основе анализа существующих решений, проведенного в главе 1, были спроектированы интерфейсы пользователя.

ГЛАВА 3.

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ- КАТАЛОГИЗАТОРА ПРОДУКТОВ КИНОИНДУСТРИИ

3.1 Обзор средств реализации

Для реализации данного приложения был выбран язык программирования C#. Будущее приложение подразумевает в себе наличие классов, отвечающих за те или иные аспекты реализации. Язык C# поддерживает ООП, что позволяет выбрать его в качестве языка написания данного приложения.

Для реализации пользовательского интерфейса было решено использовать технологию WPF, которая использует язык разметки XAML.

3.2 Бизнес-сущности приложения

Основная сущность приложения представлена классом `Film`. В нём определены все характеристики продукта:

- `Title` - название продукта;
- `Company` - имя компании производителя;
- `Year` - год выпуска продукта;
- `Genre` - жанр продукта;
- `DurationInMinutes` - длительность в минутах;
- `Format` - формат продукта;
- `Quality` - качество в пикселях;
- `ImagePath` - путь к постеру продукта;
- `TrailerPath` - путь к трейлеру продукта.

3.3 Формат хранения исходных данных

- В качестве инструмента для хранения данных приложения был выбран XML файл ([рисунок 3.3.1](#)). В заголовок документа было помещено объявление XML, в котором указывается язык разметки документа, номер его версии и дополнительная информация. Файл содержит корневой элемент <ArrayOfFilm>, в котором содержится массив элементов <Film> .

```

1  <?xml version="1.0"?>
2  <ArrayOfFilm xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
4  <Film>
5      <Title>Avangers: Endgame</Title>
6      <Company>Marvel</Company>
7      <Year>2018</Year>
8      <Genre>Fantasy</Genre>
9      <DurationInMinutes>180</DurationInMinutes>
10     <Format>BDriP</Format>
11     <Quality>1080</Quality>
12     <IsPopular>true</IsPopular>
13     <ImagePath>C:\workspace\avangers1.jpg</ImagePath>
14     <ThraillerPath>C:\workspace\avangersThrailler.mov</ThraillerPath>
15 </Film>
16 <Film>
17     <Title>Avatar</Title>
18     <Company>20th Fox Century</Company>
19     <Year>2008</Year>
20     <Genre>Fantasy</Genre>
21     <DurationInMinutes>143</DurationInMinutes>
22     <Format>DVDRip</Format>
23     <Quality>1080</Quality>
24     <IsPopular>true</IsPopular>
25     <ImagePath>C:\workspace\avatar.jpg</ImagePath>
26     <ThraillerPath>C:\workspace\avatar.mov</ThraillerPath>
27 </Film>

```

Рисунок 3.3.1 – содержимое XML файла

3.4 Программная реализация уровней архитектуры

Для реализации вспомогательных методов были реализованы классы FileCatalogService ([рисунок 3.4.1](#)) и FilmLibraryService. Первый отвечает за работу с XML файлом:

```

class FileCatalogService
{
    1 reference
    public static List<Film> getFilms()
    {
        List<Film> allFilms = new List<Film>();

        XmlSerializer formatter = new XmlSerializer(typeof(List<Film>));

        using (FileStream fs = new FileStream(System.IO.Path.Combine(Environment.CurrentDirectory, "filmCatalog.xml"), FileMode.Open))
        {
            allFilms = (List<Film>)formatter.Deserialize(fs);
        }

        return allFilms;
    }

    3 references
    public static void saveFilms(List<Film> films)
    {
        XmlSerializer formatter = new XmlSerializer(typeof(List<Film>));

        using (FileStream fs = new FileStream(System.IO.Path.Combine(Environment.CurrentDirectory, "filmCatalog.xml"), FileMode.Create))
        {
            formatter.Serialize(fs, films);
        }
    }
}

```

Рисунок 3.4.1 – содержимое класса FileCatalogService

- getFilms – метод, отвечающий за десериализацию данных;
- saveFilms – метод, отвечающий за сериализацию данных.

FilmLibraryService содержит все необходимые и вспомогательные методы:

- parseFilmsFromCatalog – возвращает значение метода getFilms, упомянутого выше ([рисунок 3.4.2](#));

```

private static List<Film> parseFilmsFromCatalog()
{
    return FileCatalogService.getFilms();
}

```

Рисунок 3.4.2 – реализация метода parseFilmsFromCatalog

- getAllFilms – возвращает список всех продуктов ([рисунок 3.4.3](#));

```

public static List<Film> getAllFilms()
{
    return films;
}

```

Рисунок 3.4.3 – реализация метода getAllFilms

- getFilmsByGenre – возвращает список продуктов, исходя из запрашиваемого жанра ([рисунок 3.4.4](#));

```

public static List<Film> getFilmsByGenre(string genre, bool isPopularOnly)
{
    List<Film> resultCollection = new List<Film>();

    foreach (var film in films)
    {
        if (isPopularOnly)
        {
            if (film.IsPopular)
            {
                resultCollection.Add(film);
            }
        }
        else
        {
            if (film.Genre.Equals(genre))
            {
                resultCollection.Add(film);
            }
        }
    }
    return resultCollection;
}

```

Рисунок 3.4.4 – реализация метода getFilmsByGenre

- getFilmsByTitle – возвращает список продуктов, исходя из запрашиваемого названия ([рисунок 3.4.5](#));

```

internal static List<Film> getFilmsByTitle(string[] userWords)
{
    List<Film> resultCollection = new List<Film>();

    foreach (var film in films)
    {
        string[] filmWords = film.Title.ToLower().Split(' ');
        foreach (var userWord in userWords)
        {
            foreach (var filmWord in filmWords)
            {
                if (filmWord.Equals(userWord))
                {
                    resultCollection.Add(film);
                }
            }
        }
    }
    return resultCollection;
}

```

Рисунок 3.4.5 – реализация метода getFilmsByTitle

- getFilmsByCompany – возвращает список продуктов, исходя из запрашиваемого названия компании производителя ([рисунок 3.4.6](#));

```

internal static List<Film> getFilmsByCompany(string[] userWords)
{
    List<Film> resultCollection = new List<Film>();

    foreach (var film in films)
    {
        string[] filmWords = film.Company.ToLower().Split(' ');
        foreach (var userWord in userWords)
        {
            foreach (var filmWord in filmWords)
            {
                if (filmWord.Equals(userWord))
                {
                    resultCollection.Add(film);
                }
            }
        }
    }
    return resultCollection;
}

```

Рисунок 3.4.6 – реализация метода getFilmsByCompany

- getFilmsByFormat – возвращает список продуктов, исходя из запрашиваемого формата ([рисунок 3.4.7](#));

```

internal static List<Film> getFilmsByFormat(string[] userWords)
{
    List<Film> resultCollection = new List<Film>();

    foreach (var film in films)
    {
        string[] filmWords = film.Format.ToLower().Split(' ');
        foreach (var userWord in userWords)
        {
            foreach (var filmWord in filmWords)
            {
                if (filmWord.Equals(userWord))
                {
                    resultCollection.Add(film);
                }
            }
        }
    }
    return resultCollection;
}

```

Рисунок 3.4.7 – реализация метода getFilmsByFormat

- getFilmsByYear – возвращает список продуктов, исходя из запрашиваемого года выпуска ([рисунок 3.4.8](#));

```
internal static List<Film> getFilmsByYear(int year)
{
    List<Film> resultCollection = new List<Film>();

    foreach (var film in films)
    {
        if (film.Year.Equals(year))
        {
            resultCollection.Add(film);
        }
    }
    return resultCollection;
}
```

Рисунок 3.4.8 – реализация метода getFilmsByYear

- getFilmsByDuration - возвращает список продуктов, исходя из запрашиваемой длительности ([рисунок 3.4.9](#));

```
internal static List<Film> getFilmsByDuration(int duration)
{
    List<Film> resultCollection = new List<Film>();

    foreach (var film in films)
    {
        if (film.DurationInMinutes.Equals(duration))
        {
            resultCollection.Add(film);
        }
    }
    return resultCollection;
}
```

Рисунок 3.4.9 – реализация метода getFilmsByDuration

- getFilmsByQuality - возвращает список продуктов, исходя из запрашиваемого качества ([рисунок 3.4.10](#));

```
internal static List<Film> getFilmsByQuality(int quality)
{
    List<Film> resultCollection = new List<Film>();

    foreach (var film in films)
    {
        if (film.Quality.Equals(quality))
        {
            resultCollection.Add(film);
        }
    }
    return resultCollection;
}
```

Рисунок 3.4.10 – реализация метода getFilmsByQuality

- sortFilmsByTitle – возвращает список продуктов отсортированный по названию продукта в алфавитном порядке ([рисунок 3.4.11](#));


```

internal static List<Film> sortFilmsByTitle()
{
    List<Film> allFilms = FilmLibraryService.getAllFilms();
    List<string> titles = new List<string>();

    foreach (var film in allFilms)
    {
        titles.Add(film.Title);
    }
    titles.Sort();

    List<string> distinct = titles.Distinct().ToList();
    List<Film> sortedByTitle = new List<Film>();

    foreach (var title in distinct)
    {
        foreach (var film in allFilms)
        {
            if (title == film.Title)
            {
                sortedByTitle.Add(film);
            }
        }
    }
    return sortedByTitle;
}

```

Рисунок 3.4.11 – реализация метода sortFilmsByTitle

- sortFilmsByCompany – возвращает список продуктов, отсортированный по названию компании производителя в алфавитном порядке ([рисунок 3.4.12](#));

```

internal static List<Film> sortFilmsByCompany()
{
    List<Film> allFilms = FilmLibraryService.getAllFilms();
    List<string> companies = new List<string>();

    foreach (var film in allFilms)
    {
        companies.Add(film.Company);
    }
    companies.Sort();

    List<string> distinct = companies.Distinct().ToList();
    List<Film> sortedByCompany = new List<Film>();

    foreach (var company in distinct)
    {
        foreach (var film in allFilms)
        {
            if (company == film.Company)
            {
                sortedByCompany.Add(film);
            }
        }
    }
    return sortedByCompany;
}

```

Рисунок 3.4.12 – реализация метода sortFilmsByCompany

- sortFilmsByYear – возвращает список продуктов, отсортированный по году выпуска продукта в порядке возрастания ([рисунок 3.4.13](#));

```
internal static List<Film> sortFilmsByYear()
{
    List<Film> allFilms = FilmLibraryService.getAllFilms();
    List<int> years = new List<int>();

    foreach (var film in allFilms)
    {
        years.Add(film.Year);
    }
    years.Sort();

    List<int> distinct = years.Distinct().ToList();
    List<Film> sortedByYear = new List<Film>();

    foreach (var year in distinct)
    {
        foreach (var film in allFilms)
        {
            if (year == film.Year)
            {
                sortedByYear.Add(film);
            }
        }
    }
    return sortedByYear;
}
```

Рисунок 3.4.13 – реализация метода sortFilmsByYear

- sortFilmsByDuration – возвращает список продуктов, отсортированный по длительности продукта в порядке возрастания ([рисунок 3.4.14](#));

```

internal static List<Film> sortFilmsByDuration()
{
    List<Film> allFilms = FilmLibraryService.getAllFilms();
    List<int> durations = new List<int>();

    foreach (var film in allFilms)
    {
        durations.Add(film.DurationInMinutes);
    }
    durations.Sort();

    List<int> distinct = durations.Distinct().ToList();
    List<Film> sortedByDuration = new List<Film>();

    foreach (var duration in distinct)
    {
        foreach (var film in allFilms)
        {
            if (duration == film.DurationInMinutes)
            {
                sortedByDuration.Add(film);
            }
        }
    }
    return sortedByDuration;
}

```

Рисунок 3.4.14 – реализация метода sortFilmsByDuration

- sortFilmsByFormat – возвращает список продуктов, отсортированный по формату продукта в алфавитном порядке ([рисунок 3.4.15](#));

```

internal static List<Film> sortFilmsByFormat()
{
    List<Film> allFilms = FilmLibraryService.getAllFilms();
    List<string> formats = new List<string>();

    foreach (var film in allFilms)
    {
        formats.Add(film.Format);
    }
    formats.Sort();

    List<string> distinct = formats.Distinct().ToList();
    List<Film> sortedByFormat = new List<Film>();

    foreach (var format in distinct)
    {
        foreach (var film in allFilms)
        {
            if (format == film.Format)
            {
                sortedByFormat.Add(film);
            }
        }
    }
    return sortedByFormat;
}

```

Рисунок 3.4.15 – реализация метода sortFilmsByFormat

- `sortFilmsByQuality` – возвращает список продуктов, отсортированный по качеству продукта в порядке возрастания ([рисунок 3.4.16](#));

```
internal static List<Film> sortFilmsByQuality()
{
    List<Film> allFilms = FilmLibraryService.getAllFilms();
    List<int> qualities = new List<int>();

    foreach (var film in allFilms)
    {
        qualities.Add(film.Quality);
    }
    qualities.Sort();

    List<int> distinct = qualities.Distinct().ToList();
    List<Film> sortedByQuality = new List<Film>();

    foreach (var quality in distinct)
    {
        foreach (var film in allFilms)
        {
            if (quality == film.Quality)
            {
                sortedByQuality.Add(film);
            }
        }
    }
    return sortedByQuality;
}
```

Рисунок 3.4.16 – реализация метода `sortFilmsByQuality`

- `searchByTyping` – метод, реализующий быстрый поиск по названию продукта ([рисунок 3.4.17](#));

```
internal static List<Film> searchByTyping(string textToSearch)
{
    List<Film> allFilms = getAllFilms();
    List<Film> resultList = new List<Film>();
    foreach (var film in allFilms)
    {
        if (film.Title.ToLower().Contains(textToSearch.ToLower()))
        {
            resultList.Add(film);
        }
    }
    return resultList;
}
```

Рисунок 3.4.17 – реализация метода `searchByTyping`

- `deleteFilm` – метод, реализующий удаление продукта из каталога ([рисунок 3.4.18](#));

```
internal static void deleteFilm(Film filmToDelete)
{
    films.Remove(filmToDelete);
    FileCatalogService.saveFilms(films);
}
```

Рисунок 3.4.18 – реализация метода deleteFilm

- addNewFilm – метод, реализующий добавления нового продукта в каталог ([рисунок 3.4.19](#));

```
internal static void addNewFilm(Film newFilm)
{
    films.Add(newFilm);
    FileCatalogService.saveFilms(films);
}
```

Рисунок 3.4.19 – реализация метода addNewFilm

- editFilm - метод, реализующий редактирование информации о продукте ([рисунок 3.4.20](#));

```
internal static void editFilm(Film filmToEdit, string title,
    string company, int year, string genre, int duration, string format,
    int quality, bool isPopular, string imagePath, string thrailerPath)
{
    filmToEdit.Title = title;
    filmToEdit.Company = company;
    filmToEdit.Year = year;
    filmToEdit.Genre = genre;
    filmToEdit.DurationInMinutes = duration;
    filmToEdit.Format = format;
    filmToEdit.Quality = quality;
    filmToEdit.IsPopular = isPopular;
    filmToEdit.ImagePath = imagePath;
    filmToEdit.ThrailerPath = thrailerPath;

    FileCatalogService.saveFilms(films);
}
```

Рисунок 3.4.20 – реализация метода editFilm

- helpToValidate – вспомогательный метод для валидации полей ввода информации ([рисунок 3.4.21](#));

```

internal static bool helpToValidate(TextBox currentTextBox, ComboBox currentComboBox)
{
    if (currentTextBox != null)
    {
        if (currentTextBox.Name.Equals("titleTextBox") || currentTextBox.Name.Equals("companyTextBox") ||
            currentTextBox.Name.Equals("yearTextBox") || currentTextBox.Name.Equals("durationInMinutesTextBox") ||
            currentTextBox.Name.Equals("formatTextBox") || currentTextBox.Name.Equals("qualityTextBox"))
        {
            if ((!currentTextBox.Text.Equals(String.Empty)) && isValidField(currentTextBox.Text, currentTextBox))
            {
                currentTextBox.BorderBrush = System.Windows.Media.Brushes.LightGray;
                return true;
            }
            else
            {
                currentTextBox.BorderBrush = System.Windows.Media.Brushes.Red;
                return false;
            }
        }
    }
    if (currentComboBox.Name.Equals("genreComboBox") || currentComboBox.Name.Equals("isPopularComboBox"))
    {
        if (currentComboBox.Name.Equals("genreComboBox") && currentComboBox.Text.Equals(String.Empty))
        {
            MessageBox.Show("Please select genre");
            return false;
        }
        else if (currentComboBox.Name.Equals("isPopularComboBox") && currentComboBox.Text.Equals(String.Empty))
        {
            MessageBox.Show("Please select popularity of the film");
            return false;
        }
        else
        {
            return true;
        }
    }
    return false;
}

```

Рисунок 3.4.21 – реализация метода helpToValidate

- isValidField – вспомогательный метод, упрощающий реализацию метода helpToValidate ([рисунок 3.4.22](#)).

```

internal static bool isValidField(string fieldText, TextBox textBox)
{
    int helpToParseNumber;
    if (textBox.Name.Equals("titleTextBox"))
    {
        return textBox.Text.ToString().Length <= 50;
    }
    if (textBox.Name.Equals("companyTextBox"))
    {
        return textBox.Text.ToString().Length <= 50;
    }
    if (textBox.Name.Equals("yearTextBox"))
    {
        bool isNumber = int.TryParse(fieldText, out helpToParseNumber);
        if (isNumber)
        {
            return helpToParseNumber >= 1895 && helpToParseNumber <= 2020;
        }
    }
    if (textBox.Name.Equals("durationInMinutesTextBox"))
    {
        bool isNumber = int.TryParse(fieldText, out helpToParseNumber);
        if (isNumber)
        {
            return helpToParseNumber > 0 && helpToParseNumber <= 600;
        }
    }
    if (textBox.Name.Equals("formatTextBox"))
    {
        return textBox.Text.ToString().Length <= 15;
    }
    if (textBox.Name.Equals("qualityTextBox"))
    {
        bool isNumber = int.TryParse(fieldText, out helpToParseNumber);
        if (isNumber)
        {
            return helpToParseNumber >= 144 && helpToParseNumber <= 4320;
        }
    }
    return false;
}

```

Рисунок 3.4.22 – реализация метода isValidField

На уровне «Модель» реализован класс Film ([рисунок 3.4.23](#)), описывающий основные сущности предметной области.

```

public class Film
{
    6 references
    public string Title { get; set; }
    5 references
    public string Company { get; set; }
    5 references
    public int Year { get; set; }
    3 references
    public string Genre { get; set; }
    5 references
    public int DurationInMinutes { get; set; }
    5 references
    public string Format { get; set; }
    5 references
    public int Quality { get; set; }
    4 references
    public bool IsPopular { get; set; }
    3 references
    public string ImagePath { get; set; }
    4 references
    public string ThraillerPath { get; set; }
    1 reference
    public Film()
    {
        .
    }
    1 reference
    public Film(string title, string company, int year, string genre,
        int durationInMinutes, string format, int quality,
        bool isPopular, string imagePath, string thraillerPath)
    {
        this.Title = title;
        this.Company = company;
        this.Year = year;
        this.Genre = genre;
        this.DurationInMinutes = durationInMinutes;
        this.Format = format;
        this.Quality = quality;
        this.IsPopular = isPopular;
        this.ImagePath = imagePath;
        this.ThraillerPath = thraillerPath;
    }
}

```

Рисунок 3.4.23 – содержимое класса Film

С учетом требований, перечисленных в главе 2, было спроектировано приложение, скриншоты которого представлены на рисунках 3.4.24 – 3.4.28.

Главное окно приложения ([рисунок 3.4.24](#)) изначально включает в себя список популярных продуктов, а также возможность быстрого поиска по названию. В верхнем меню пользователь может выбрать продукты по жанрам. Также в верхнем правом углу по клику на кнопку “Search” откроется окно с расширенным поиском. Также по клику на кнопки под верхним меню пользователь может получить список всех продуктов и отсортированных по любым критериям. Здесь же присутствует кнопка добавления нового продукта в каталог.

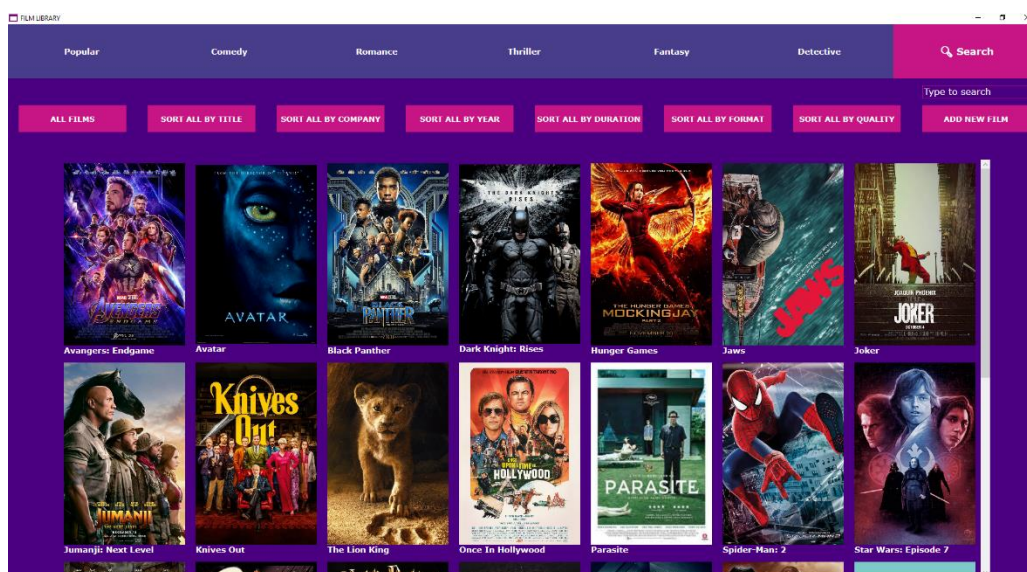


Рисунок 3.4.24 – главное окно приложения

Пользователь имеет возможность просмотреть подробную информацию о заинтересовавшем его продукте, кликнув на постер или название на главном окне. Откроется окно ([рисунок 3.4.25](#)) со всей информацией о выбранном продукте. Здесь представлена возможность редактирования информации, удаления продукта из каталога, а также просмотра трейлера продукта.

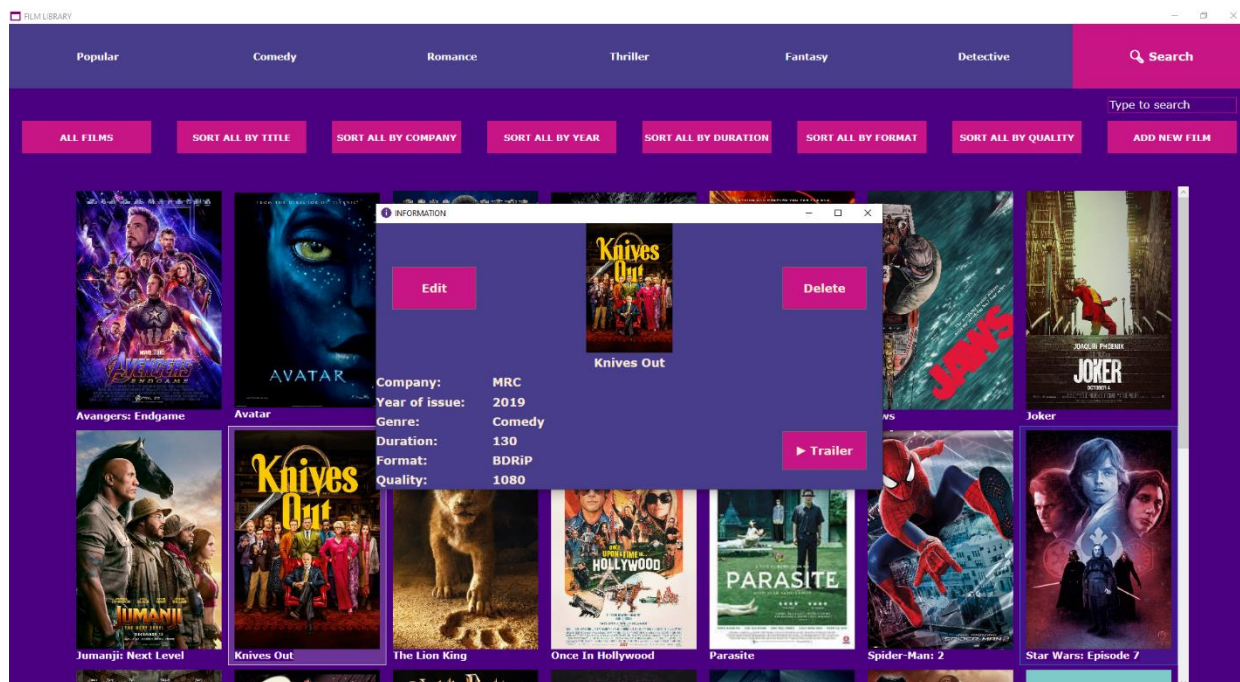


Рисунок 3.4.25 – окно информации о продукте

Нажав на кнопку “*Edit*” в окне информации о продукте, появится окно (рисунок 3.4.26) с текущими данными о продукте и возможностью их изменения. Для применения изменений пользователь должен кликнуть по кнопке “*Save*”.

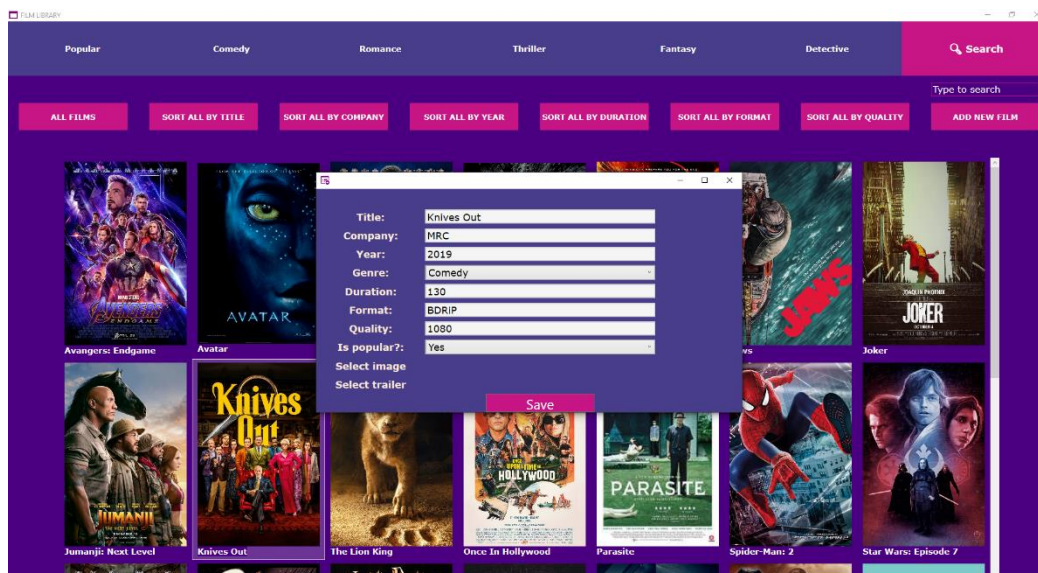


Рисунок 3.4.26 – окно редактирования информации о продукте

При клике на кнопку “*Add new film*” на главном окне, пользователю открывается окно (рисунок 3.4.27) заполнения информации о продукте. Как и упоминалось выше окно редактирования и окно добавления это одно и то же окно, только в первом случае в нем отображаются данные о выбранном продукте.

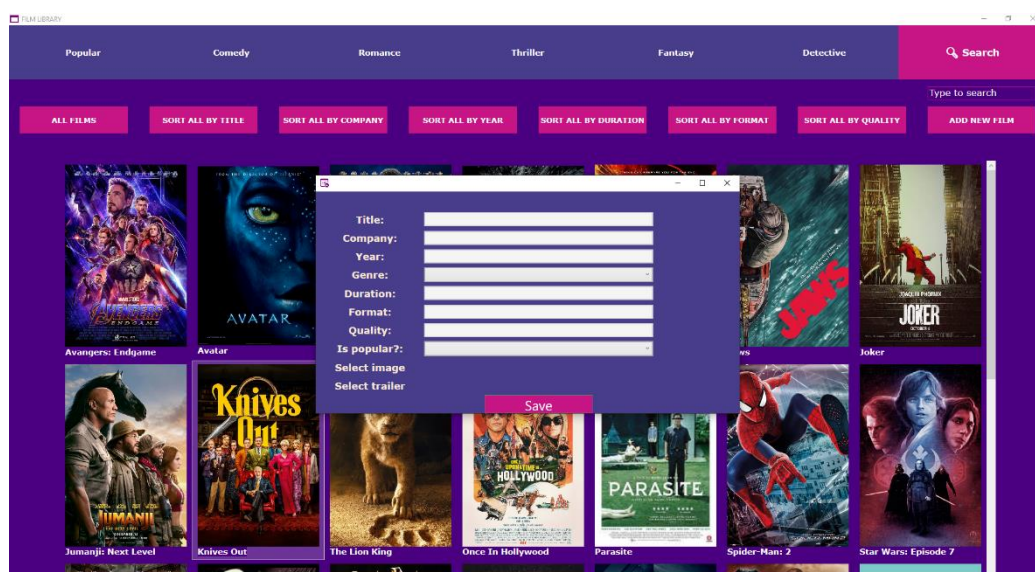


Рисунок 3.4.27 – окно добавления нового продукта в каталог

Кликнув по кнопке “Search” в верхнем правом углу главного окна, пользователю откроется окно ([рисунок 3.4.28](#)) расширенного поиска по желаемым параметрам.

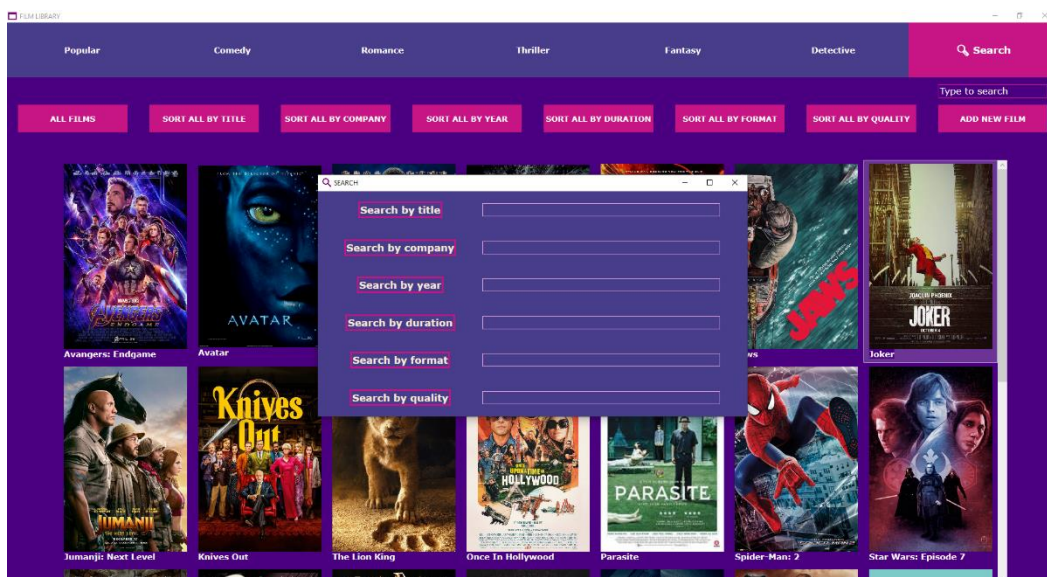


Рисунок 3.4.28 – окно расширенного поиска

3.5 Выводы по главе 3

На основании результатов проектирования приложения, для его разработки были выбраны оптимальные основные средства: язык программирования C# с поддержкой ООП, технология построения графических интерфейсов WPF, язык разметки XML для хранения и передачи данных.

Приложение было разделено на компоненты, каждый из которых находится в собственной зоне ответственности. В ходе работы была успешно реализована основная часть приложения, а также разработан пользовательский интерфейс. Основным требованием к дизайну являлась его интуитивная понятность.

ЗАКЛЮЧЕНИЕ

В ходе данной курсовой работы была поставлена цель – разработать приложение для быстрого и удобного просмотра каталога продуктов киноиндустрии. Было составлено техническое задание с описанием требований к данному проекту.

На основании исследования архитектуры, а также информации, полученной при анализе приложений подобного типа, было разработано собственное приложение, включающее основную часть и пользовательский интерфейс. Результатом данной работы является приложение, имеющее следующие возможности:

- просмотр каталога продуктов и информации о них;
- быстрый поиск по каталогу;
- расширенный поиск по каталогу;
- сортировка по выбранным критериям;
- изменение информации о продукте;
- удаление и добавление продукта;

В будущем данное приложение может иметь практическую пользу и быть использовано как действующий сервис по просмотру каталога продуктов киноиндустрии. Кроме того, приложение можно расширять, добавляя новый функционал.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Троелсен, Э. Язык программирования C# и платформы .NET и .NET Core: книга / Э. Троелсен, Ф. Джепикс. – 2018. – 1314 с.
2. Албахари Дж. C# 7.0. Карманный справочник: учебник / Дж.Албахари, Б. Албахари. – 2017. - 224 с.
3. metanit.com. Руководство по WPF [Электронный ресурс] / metanit.com – Режим доступа: <https://metanit.com/sharp/wpf/> - Дата доступа: 29.11.2020.
4. shwanoff.ru. Работа с XML C# в примерах [Электронный ресурс] / shwanoff.ru – Режим доступа: <https://shwanoff.ru/xml/> - Дата доступа: 01.12.2020.
5. habr.ru. Быстрый старт с WPF. Привязка, INotifyPropertyChanged и MVVM [Электронный ресурс] / habr.ru – Режим доступа: <https://habr.com/ru/post/427325/> – Дата доступа: 30.11.2020.