



ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

**НАПРАВЛЕНИЕ „Компютърно и софтуерно
инженерство”**

Практически проект по „ Програмни среди”

**Тема: Система за извеждане на контроли за потребителя, в
които да се въвеждат критерии за търсене**

Изработил: Иван Леонов, фак. номер: 121220051, група: 43

Приел: гл. ас. д-р Петър Маринов

Система за извеждане на контроли за потребителя, в които да се въвеждат критерии за търсене (Документация)

1. Изходна постановка - описвате функционалностите и основните класове/таблицы/структури на примерното приложение върху което ще демонстрирате вашата система.

Приложението представлява система за управление на библиотеки, която позволява на потребителите да търсят книги и автори по дадени критерии от база данни. Системата е изградена с помощта на рамката WPF (Windows Presentation Foundation) и следва архитектурния модел MVVM (Model-View-ViewModel).

1.1 Функционалности

1.1.1. Търсене на книги (SearchBook):

- Потребителите могат да търсят книги въз основа на различни критерии, като заглавие, автор, година на публикуване. Функционалността за търсене е реализирана с помощта на класа SearchBookViewModel, който произлиза от класа SearchViewModelBase<T>, който е абстрактен. Класът SearchBookViewModel капсулира логиката за търсене на книги и прилага филтри към резултатите от търсенето. Съответният изглед за търсене на книги е представен от два потребителски контрола (UserControls), които са DropDownMenusAndFilterView (за избор на критерии, за поле за търсене, бутони за добавяне и премахване на избран филтър, както и списък с добавените филтри) и ResultGridView (включващ списък с филтрираните елементи, който се обновява в реално време в зависимост от действията на потребителя).

1.1.2. Търсене на автори (SearchAuthor):

- Потребителите могат да търсят автори въз основа на критерии като име, държава и рождена дата. Функционалността за търсене на автори е реализирана с помощта на класа SearchAuthorViewModel, който също произлиза от класа SearchViewModelBase<T>, по същия начин като SearchBook. Класът SearchAuthorViewModel обработва логиката за търсене на автори и прилагане на филтри към резултатите от търсенето. Съответният изглед за търсене на автори е представен от два потребителски контрола (UserControls), които са DropDownMenusAndFilterView (за избор на критерии, за поле за търсене, бутони за добавяне и премахване на избран филтър, както и списък с добавените филтри) и ResultGridView (включващ списък с филтрираните елементи, който се обновява в реално време в зависимост от действията на потребителя).

1.1.3. Филтриране на резултатите:

- Потребителите могат да прилагат филтри, за да прецизират допълнително резултатите от търсенето въз основа на специфични изисквания. Класът `SearchViewModelBase` предоставя обща функционалност за прилагане на филтри към резултатите от търсенето. В него е цялата функционалност за работа с полета/свойства от база данни, което го прави много полезен за използване с много на брой различни по себе си обекти. Той е абстрактен клас, който може да бъде наследен много лесно от всеки друг и да бъде използвана цялата му функционалност. Към него имаме помощен клас – `SearchHelper`, който служи за прилагане на филтри и търсене по колекции от елементи. Той се използва с цел да се спазят MVVM принципите и да вземат заявки от базата данни.

1.1.4. Потребителски интерфейс:

- Основният прозорец на приложението е представен от класа `MainWindow`. В него потребителският интерфейс се състои от `ComboBox` (поле за избор) за избор на типа търсене по даден модел (в нашето приложение книги или автори). `ContentControl` служи за динамично показване на съответния изглед за търсене, като нашия изглед е разделен на два потребителски контрола. Класът `MainWindow` задава контекста на данните и свързва `ComboBox` (поле за избор) с колекцията от налични типове за търсене. Моделите и изгледите (`Views`) обработват логиката на показване и взаимодействие за търсене. функционалността за търсене.

1.2 Основни класове/таблицы/структури

1.2.1. Models (Модели):

- `SearchBook` (Търсене на книга): Представява книга в библиотеката със свойства като заглавие, автор, дата на публикуване, статус.
- `SearchAuthor` (Търсене на автор): Представява автор със свойства като име, държава, рождена дата. Тези модели определят структурата на същностите от данни, използвани в системата за управление на библиотеката.
- `SearchField` (Търсене на поле): Този клас се използва за дефиниране на полетата за търсене върху моделите в приложението. Всяко поле за търсене има име, свързан модел, възможност за вложени полета и тип на стойността. Това позволява гъвкавост при дефинирането на търсенето, като можем да имаме различни типове стойности за различни полета (например `string`, `int`, `DateTime` и др.).
- `SearchFilter` (Филтър за търсене): Този клас се използва за дефиниране на филтър върху полето за търсене. Всякакъв филтър има асоциирано поле за търсене и стойност, която се използва за филтриране на данните. Можем да създадем множество филтри, които се прилагат върху моделите за търсене.
- `MyDbContext` (Контекст на базата данни): Представява контекста на базата данни за взаимодействие с базовата база данни с помощта на `Entity Framework` и `Microsoft SQL Server Management`. Класът `DbContext` осигурява достъп до същностите `SearchBook` (Книга) и `SearchAuthor` (Автор), като позволява извличане и манипулиране на данни.

1.2.2. ViewModels (Модели на изгледа):

- `SearchViewModelBase<T>`: Абстрактен базов клас за `ViewModel` за търсене, който предоставя обща функционалност за търсене и филтриране на елементи.
- `SearchBookViewModel`: Специфичен модел на изглед за търсене на книги, който наследява абстрактния клас `SearchViewModelBase<Book>`, за да използва цялата му функционалност.
- `SearchAuthorViewModel`: Специфичен модел на изглед за търсене на автори, който наследява абстрактния клас `SearchViewModelBase<Author>`, за да използва цялата му функционалност.
- `MainWindowViewModel`: Специфичен модел на изглед, който служи за управление на главния прозорец на приложението и осигурява връзка между изгледите и моделите. Има свойства `ObjectTypes` и `SelectedObjectType`, `SelectedViewModel`, които се използват за избор на тип обект от списъка налични типове и избор на изглед на модела (`ViewModel`).
- `RelayCommand`: Предоставя начин за дефиниране и обработка на команди в `ViewModel` чрез капсулиране на логиката за изпълнение и определяне на способността за изпълнение на команда. Той следва общата реализация на интерфейса `ICommand`, позволявайки свойствата на `ViewModel` да бъдат свързани със свойствата на командата в потребителския интерфейс, като бутони или елементи от менюто, за да задействат конкретни събития, когато бъдат извикани.

1.2.3. Views (Изгледи):

- `MainWindow`: Основният прозорец на приложението, съдържащ потребителския интерфейс за избор на типа търсене и показване на съответния изглед за търсене.
- `DropDownMenusAndFilterView`: `UserControl`, който се използва за показване на падащо меню за даден критерии, полета за търсене, визуализиране на списък с избрани филтри и свързани действия в графичен потребителски интерфейс. Специфичното поведение и функционалност на `UserControl` се реализират в съответния `ViewModel`, който трябва да бъде зададен като `DataContext` за този изглед.
- `ResultGridView`: `UserControl`, който се използва за показване на списък с филтрирани елементи в резултат на операция за търсене или филтриране. Специфичните елементи, които трябва да бъдат показани, се предоставят чрез свойството `"FilteredItems"` на свързания `ViewModel`, който трябва да бъде зададен като `DataContext` за този изглед.

1.2.4. Extensions (Разширения):

- `DataGridExtensions`: Този клас предоставя метод за разширение за контролата `DataGrid` в WPF за автоматично генериране и форматиране на колони за свойствата `DateTime`.

1.2.5. Helpers (Разширения):

- `SearchHelper`: Този клас предоставя функционалност за прилагане на филтри и извършване на търсене в колекция от елементи, като позволява филтриране и търсене въз основа на избрани филтри, полета за търсене и стойности за търсене, като позволява лесна имплементация в друг `ViewModel`.

2. Обща структура на предложеното решение - основните положения и изисквания които сте приели за необходими за да работи вашето решение

Предложеното решение за система за извеждане на контроли за потребителя, в които да се въвеждат критерии за търсене е изградено с помощта на следните компоненти:

2.1. Технологичен пакет

- Windows Presentation Foundation (WPF): Рамката, използвана за разработване на графичния потребителски интерфейс (GUI) на приложението.
- Entity Framework: Технология за достъп до данни, която осигурява обектно-реляционно съпоставяне (ORM) за взаимодействие с основната база данни.
- C#: Езикът за програмиране, използван за реализиране на бизнес логиката и функционалността на приложението.

2.2. Архитектура

- MVVM (Model-View-ViewModel): Архитектурният модел, следван в решението.
- Модели (Models): Представяват единиците данни, използвани в системата за управление на библиотеката, като например книги и автори.
- Изгледи (Views): Осигуряват визуалното представяне и оформление на елементите на потребителския интерфейс.
- Модели на изгледи (ViewModels): Обслужват логиката на представяне, свързването на данни и взаимодействието между изгледите и моделите.
- Помощни (Helpers): Включва помощни методи за прилагане на методи като търсене в колекция от елементи / заявки към база данни и т.н.
- Разширения (Extensions): Служат за добавяне на разширени функционалности към DataGridView контрола.

2.3. Основни положения и изисквания

2.3.1. Свързаност с база данни

- Решението изисква връзка с база данни, за да се съхраняват и извличат данни.
- Entity Framework се използва за безпроблемна интеграция с базата данни.
- Класът DbContext отговаря за установяването на връзката с базата данни и за управлението на контекста на данните.

2.3.2. Функционалност за търсене

- Решението осигурява функционалност за търсене на книги и автори.
- Потребителите могат да търсят книги въз основа на всички критерии в базата данни за дадения модел.
- Функционалността за търсене е реализирана в класовете SearchBookViewModel и SearchAuthorViewModel, които наследяват абстрактния клас SearchViewModelBase<T>.

който пък има помощен статичен клас SearchHelper, който съдържа цялостната функционалността за търсене и прилагане на критерии.

- Решението позволява на потребителите да прилагат филтри за прецизиране на резултатите от търсенето, като по този начин подобряват преживяването при търсене.

2.3.3. Потребителски интерфейс

- Решението разполага с удобен за потребителя интерфейс, използващ рамката WPF.
- Класът MainWindow служи като основен прозорец на приложението.
- Съответният изглед за търсене се появява динамично въз основа на избрания тип търсене.
- Изгледите са изградени от два потребителски контрола (UserControls), с изгледи съответно за избор на критерии, поле за търсене, списък с избрани филтри, както и такъв с филтрираните елементи, който се променя в реално време спрямо действията на потребителя.

2.3.4. Разширяемост и възможност за поддържане

- Решението следва модулен и разширяем подход, което улеснява бъдещите подобрения.
- Нови типове търсене, като например търсене по жанр или по издател, могат да се добавят лесно чрез разширяване на класовете и изгледите на решението.
- Използването на модели на изгледи и разделянето на проблемите позволява поддържане и по-лесно модифициране на функционалността.

3: Описание на решението - описание и предназначени на предложените класове, снопети от код с детайлно обяснение на ключовите моменти от кода

3.1. MainWindow

Класът MainWindow служи като главен прозорец на приложението. Той осигурява потребителския интерфейс за избор на типа търсене (книги или автори) и показва съответния изглед за търсене въз основа на избора. Наследява се от класа Window. Конструкторът инициализира MainWindowViewModel и го задава като контекст на данните за MainWindow. Този клас е отговорен за инициализирането и показването на главния прозорец на приложението.

```

using UniversalSearchCriteria_PS_IvanLeonov_43.ViewModels;
using System.Windows;

namespace UniversalSearchCriteria_PS_IvanLeonov_43.Views
{
    2 references
    public partial class MainWindow : Window
    {
        private MainWindowViewModel viewModel;

        0 references
        public MainWindow()
        {
            InitializeComponent();
            viewModel = new MainWindowViewModel();
            DataContext = viewModel; // Set the MainWindowViewModel as the DataContext
        }
    }
}

```

MainWindow.xaml: Дефинира главния прозорец на приложението за система за търсене по дадени критерии. Осигурява потребителски интерфейс с падащо меню за избор на типа на обекта за търсене и динамично показва съответните изгледи за филтриране и показване на резултатите от търсенето. Изгледите се избират и показват въз основа на избрания модел на изглед с помощта на DataTemplates, което позволява модулен и гъвкав дизайн.

Необходимите пространства от имена (namespaces) се дефинират чрез атрибутите xmlns и xmlns:x. Атрибутът xmlns:local конкретно дефинира пространство от имена за препратка към пространството от имена "UniversalSearchCriteria_PS_IvanLeonov_43.Views", което съдържа изгледите, използвани в този прозорец.

Основният прозорец е дефиниран със заглавие "Search Criteria" (Критерии за търсене) и е с определена височина и ширина. Свойството DataContext е зададено на инстанция на MainWindowViewModel, която служи като контекст на данните за прозореца.

Основното съдържание на прозореца е затворено в елемент Grid. Дефинициите на Grid.RowDefinitions определят два реда: първият ред ще има височина, която отговаря на съдържанието му (Auto), а вторият ред ще заема останалото пространство (*).

Елементът ComboBox се използва като падащо меню за избор на типа на обекта за търсене. Той е свързан със свойството "ObjectTypes" на MainWindowViewModel, което предоставя списък с наличните типове обекти. Свойството SelectedItem е обвързано със свойството "SelectedObjectType", което съдържа текущо избрания тип обект.

Елементът ContentControl се използва за показване на избрания изглед въз основа на избрания тип обект. Свойството Content е свързано със свойството "SelectedViewModel" на MainWindowViewModel. Това свойство съхранява текущо избрания модел на изглед и съответният изглед ще се показва динамично под ComboBox във втория ред на мрежата.

В ресурсите на ContentControl са дефинирани два DataTemplates. Тези DataTemplates указват как трябва да се визуализират изгледите за различните модели изгледи. Първият DataTemplate е

свързан с класа "SearchBookViewModel" и указва, че "DropDownMenusAndFilterView" и "ResultGridView" трябва да се показват, когато този модел на изглед е активен. Вторият DataTemplate е свързан с класа "SearchAuthorViewModel" и указва, че същите "DropDownMenusAndFilterView" и "ResultGridView" трябва да се показват, когато този модел на изглед е активен. Чрез използването на DataTemplates подходящите изгледи се избират и показват динамично въз основа на избрания тип обект. "DropDownMenusAndFilterView" и "ResultGridView" са изгледи за многократна употреба, които могат да се споделят между различни модели на изгледи.

```
<Window x:Class="UniversalSearchCriteria_PS_IvanLeonov_43.Views.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:UniversalSearchCriteria_PS_IvanLeonov_43.Views"
        xmlns:vm="clr-namespace:UniversalSearchCriteria_PS_IvanLeonov_43.ViewModels"
        Title="Search Criteria" Height="450" Width="800">
    <Window.DataContext>
        <vm:MainWindowViewModel />
    </Window.DataContext>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <ComboBox ItemsSource="{Binding ObjectTypes}" SelectedItem="{Binding SelectedObjectType}" />
        <ContentControl Grid.Row="1" Content="{Binding SelectedViewModel}">
            <ContentControl.Resources>
                <DataTemplate DataType="{x:Type vm:SearchBookViewModel}">
                    <StackPanel>
                        <local:DropDownMenusAndFilterView />
                        <local:ResultGridView />
                    </StackPanel>
                </DataTemplate>
                <DataTemplate DataType="{x:Type vm:SearchAuthorViewModel}">
                    <StackPanel>
                        <local:DropDownMenusAndFilterView />
                        <local:ResultGridView />
                    </StackPanel>
                </DataTemplate>
            </ContentControl.Resources>
        </ContentControl>
    </Grid>
</Window>
```

3.2. MainWindowViewModel:

Класът MainWindowViewModel отговаря за управлението на данните и логиката, свързани с главния прозорец на приложението. Той имплементира интерфейса INotifyPropertyChanged, за да осигури уведомяване за промени при свързването на данни. Конструкторът инициализира модела на изгледа. Първоначално се извиква методът GetModelTypesWithViewModels, за да се извлекат наличните типове обекти със съответните им модели на изгледи. След това задава свойството ObjectTypes на получения списък, задава SelectedObjectType на първия елемент в списъка ObjectTypes и накрая извиква метода InitializeSelectedViewModel. Свойството ObjectTypes (Типове обекти) представя наличните типове обекти в проекта. Свойството SelectedObjectType (Избран тип обект) е текущо избрания тип обект. Свойството SelectedViewModel предоставя getter и setter за

полето `selectedViewModel`. Задаващият елемент актуализира стойността на `selectedViewModel` и предизвиква събитието `PropertyChanged` за свойството `"SelectedViewModel"`.

Методът `InitializeSelectedViewModel` инициализира модела на избрания изглед въз основа на избрания тип обект. Той конструира напълно името на модела на изгледа, като добавя `"ViewModel"` към избрания тип обект. Ако типът на модела на изгледа е намерен, се създава негова инстанция с помощта на `Activator.CreateInstance` и се присвоява на `SelectedViewModel`. Ако типът на модела на изгледа не е намерен, `SelectedViewModel` се задава като `null`. Методът `OnPropertyChanged` се използва за предизвикване на събитието `PropertyChanged` и уведомяване на абонатите за промени в свойствата.

Методът `GetModelTypesWithViewModels` извлича наличните типове модели в проекта, които имат съответни модели и `ViewModel`-и. За всеки тип модел той конструира имената на свързаните типове модели на изгледи и проверява дали те съществуват. Ако моделите на изгледа (`ViewModels`) и модел (`Models`) съществуват, типът на модела се добавя към списъка. `MainWindowViewModel` играе решаваща роля в решението на системата. Той управлява списъка с типове обекти, позволява избора на тип обект и осигурява известяване за промени при свързването на данни.

Като цяло, класът `MainWindowViewModel` осигурява съществен компонент за системата за управление на библиотеката, като подобрява работата на потребителя и позволява безпроблемна работа с различни обекти в приложението.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Reflection;

namespace UniversalSearchCriteria_PS_IvanLeonov_43.ViewModels
{
    3 references
    public class MainWindowViewModel : INotifyPropertyChanged
    {
        private List<string> objectTypes; // List for available object types
        private string selectedObjectType;
        private object selectedViewModel;

        public event PropertyChangedEventHandler PropertyChanged;

        1 reference
        public MainWindowViewModel()
        {
            ObjectTypes = GetModelTypesWithViewModels(); // Get the available object types with corresponding view models
            SelectedObjectType = ObjectTypes.FirstOrDefault();
            InitializeSelectedViewModel();
        }

        3 references
        public List<string> ObjectTypes
        {
            get { return objectTypes; }
            set
            {
                objectTypes = value;
                OnPropertyChanged(nameof(ObjectTypes));
            }
        }

        3 references
        public string SelectedObjectType
        {
            get { return selectedObjectType; }
            set
            {
                selectedObjectType = value;
                InitializeSelectedViewModel(); // Initialize the selected view model based on the new selected object type
                OnPropertyChanged(nameof(SelectedObjectType));
            }
        }
    }
}

```

```

    3 references
    public object SelectedViewModel
    {
        get { return selectedViewModel; }
        set
        {
            selectedViewModel = value;
            OnPropertyChanged(nameof(SelectedViewModel));
        }
    }

    // Method to initialize the selected view model based on the selected object type
    2 references
    private void InitializeSelectedViewModel()
    {
        string viewModelTypeName = "Library_PS_Project.ViewModels." + SelectedObjectType + "ViewModel";
        Type viewModelType = Type.GetType(viewModelTypeName);

        if (viewModelType != null)
        {
            SelectedViewModel = Activator.CreateInstance(viewModelType); // Create an instance of the selected view model
        }
        else
        {
            // Handle the case when the ViewModel type is not found
            SelectedViewModel = null;
        }
    }
}

```

```

3 references
public object SelectedViewModel
{
    get { return selectedViewModel; }
    set
    {
        selectedViewModel = value;
        OnPropertyChanged(nameof(SelectedViewModel));
    }
}

// Method to initialize the selected view model based on the selected object type
2 references
private void InitializeSelectedViewModel()
{
    string viewModelTypeName = "UniversalSearchCriteria_PS_IvanLeonov_43.ViewModels." + SelectedObjectType + "ViewModel";
    Type viewModelType = Type.GetType(viewModelTypeName);

    if (viewModelType != null)
    {
        SelectedViewModel = Activator.CreateInstance(viewModelType); // Create an instance of the selected view model
    }
    else
    {
        // Handle the case when the ViewModel type is not found
        SelectedViewModel = null;
    }
}

// Method to get the available object types with corresponding view models
1 reference
private List<string> GetModelTypesWithViewModels()
{
    var modelTypes = Assembly.GetExecutingAssembly()
        .GetTypes()
        .Where(t => t.IsClass && !t.IsAbstract && t.Namespace == "UniversalSearchCriteria_PS_IvanLeonov_43.Models")
        .ToList();

    var modelTypesWithViewModels = new List<string>();

    foreach (var modelType in modelTypes)
    {
        var viewModelTypeName = $"UniversalSearchCriteria_PS_IvanLeonov_43.ViewModels.{modelType.Name}ViewModel";
        var viewModelType = Assembly.GetExecutingAssembly().GetType(viewModelTypeName);

        if (viewModelType != null)
        {
            modelTypesWithViewModels.Add(modelType.Name);
        }
    }

    return modelTypesWithViewModels;
}

// Helper method to raise the PropertyChanged event
3 references
protected virtual void OnPropertyChanged(string propertyName)
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

```

3.3. DropDownMenusAndFilterView

Потребителският контрол "DropDownMenusAndFilterView" осигурява визуално представяне на падащо меню с критерии, полета за търсене и филтри. Той позволява на потребителите да избират поле за търсене, да въвеждат стойности за търсене, да добавят филтри и да премахват филтри. Контролът е предназначен да се използва в рамките на главния прозорец на приложението, като осигурява функционалност за търсене. В него са импортирани необходимите пространства от имена (namespaces), включително пространството от имена по подразбиране на XAML и пространствата от имена за моделите на изгледи и разширенията, използвани в контролата. Оформлението му се определя с помощта на контейнер Grid. Той има четири реда и три колони. Редовете и колоните на Grid се дефинират с помощта на елементите Grid.RowDefinitions и Grid.ColumnDefinitions, като се задават съответно височината и ширината на всеки ред и колона.

Дефинирани са и два елемента за управление на етикетите (labels), за да се показва текст за етикетите "Search By" и "Search".

Контролът ComboBox се използва за показване и избор на полета за търсене. Той е свързан със свойството SearchFields в модела на изгледа чрез свойството ItemsSource. Свойството DisplayMemberPath определя свойството, което да се показва за всеки елемент в ComboBox. Свойството SelectedItem е обвързано със свойството SelectedSearchField в модела на изгледа, което позволява двупосочно обвързване. ContentControl се използва за динамично показване на избрания контрол за стойността на търсенето въз основа на избраното поле за търсене. Той е свързан със свойството SelectedSearchValueControl в модела на изгледа. Дефинирани са два бутона за управление. Първият бутон със съдържание "Add Filter" (Добавяне на филтър) е свързан с AddFilterCommand (Добавяне на филтър) в модела на изгледа. Вторият бутон със съдържание "Clean Search Field" (Изчистване на полето за търсене) е свързан с командата RemoveSearchValueCommand в модела на изгледа и използва SelectedSearchValueControl (Избрана стойност на търсенето) като CommandParameter. За показване на избраните филтри се използва контрол ListBox. Той е свързан със свойството SelectedFilters в модела на изгледа чрез свойството ItemsSource. Свойството SelectedItem е свързано със свойството SelectedFilter в модела на изгледа. ListBox също така прилага стил, който задава свойството MaxHeight, за да ограничи височината на ListBox. Дефиниран е бутон за управление със съдържание "Remove Filter". Той е свързан с командата RemoveFilterCommand в модела на изгледа и използва SelectedFilter като CommandParameter.

```
<UserControl x:Class="UniversalSearchCriteria_PS_IvanLeonov_43.Views.DropDownMenusAndFilterView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:vm="clr-namespace:UniversalSearchCriteria_PS_IvanLeonov_43.ViewModels"
    xmlns:extensions="clr-namespace:UniversalSearchCriteria_PS_IvanLeonov_43.Extensions"
    xmlns:sys="clr-namespace:System;assembly=mscorlib">

    <Grid Height="110">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="0"*/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="*"*/>
        </Grid.ColumnDefinitions>
        <Label Grid.Row="0" Grid.Column="0" Content="Search By:" Margin="4,3,6,7"/>
        <ComboBox ItemsSource="{Binding SearchFields}"
            DisplayMemberPath="Name"
            SelectedItem="{Binding SelectedSearchField, Mode=TwoWay}"
            Grid.Column="2" Margin="3,4,550,5" */>

        <Label Grid.Row="1" Grid.Column="0" Content="Search:" Margin="10,5,16,5"/>
        <ContentControl Grid.Row="1" Grid.Column="1" Content="{Binding SelectedSearchValueControl}" Margin="5,6,550,4" Grid.ColumnSpan="2"/>
        <Button Grid.Row="1" Grid.Column="2" Content="Add Filter" Margin="285,5,224,5" Command="{Binding AddFilterCommand}"/>
        <Button Grid.Row="1" Grid.Column="2" Content="Clean Search Field" Margin="170,6,437,5" Command="{Binding RemoveSearchValueCommand}" CommandParameter="{Binding SelectedSearchValueControl}" */>

        <ListBox Grid.Column="2" ItemsSource="{Binding SelectedFilters}"
            SelectedItem="{Binding SelectedFilter}" Margin="500,4,10,24" Grid.RowSpan="3">
            <ListBox.Style>
                <Style TargetType="ListBox">
                    <Setter Property="MaxHeight" Value="220"/>
                </Style>
            </ListBox.Style>
        </ListBox>
        <Button Content="Remove Filter" Margin="285,4,224,1" Command="{Binding RemoveFilterCommand}" CommandParameter="{Binding SelectedFilter}" Grid.Column="2" Grid.Row="0"/>
    </Grid>
</UserControl>
```

3.4 ResultGridView

Потребителският контрол "ResultGridView" осигурява визуално представяне на DataGrid, който показва филтрирани елементи. Той е предназначен да се използва в главния прозорец на приложението UniversalSearchCriteria_PS_IvanLeonov_43, като осигурява изглед за резултатите от търсенето. DataGrid е персонализиран с разширението AutoGenerateDateTimeColumns, за да обработва автоматичното генериране на колони за типове данни DateTime.

Импортират се необходимите пространства от имена (namespaces), включително пространството от имена по подразбиране на XAML и пространствата от имена за моделите на изгледи и разширенията, използвани в контролата. Потребителският контрол съдържа един контейнер Grid. Той има един ред, дефиниран чрез елемента Grid.RowDefinitions. Височината на реда е зададена на "*", за да заеме цялото налично вертикално пространство. Контролът DataGrid се използва за показване на филтрираните елементи. Той е свързан със свойството FilteredItems в модела на изгледа чрез свойството ItemsSource. DataGrid се поставя в първия (и единствен) ред на решетката. DataGrid дефинира секция "Ресурси" чрез елемента DataGrid.Resources. Дефинира се стил, насочен към типа DataGrid. Вътре в стила се използва Setter, за да се зададе приложеното свойство AutoGenerateDateTimeColumns от пространството от имена extensions на "True". Това разширение осигурява персонализирано поведение, свързано с автоматично генериране на колони за типове данни DateTime.

```
<UserControl x:Class="UniversalSearchCriteria_PS_IvanLeonov_43.Views.ResultGridView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:vm="clr-namespace:UniversalSearchCriteria_PS_IvanLeonov_43.ViewModels"
    xmlns:extensions="clr-namespace:UniversalSearchCriteria_PS_IvanLeonov_43.Extensions"
    xmlns:sys="clr-namespace:System;assembly=mscorlib">

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>

        <DataGrid Grid.Row="0" ItemsSource="{Binding FilteredItems}" Margin="4,5,229,0" MaxHeight="260" VerticalScrollBarVisibility="Auto">
            <DataGrid.Resources>
                <Style TargetType="DataGrid">
                    <Setter Property="extensions:DataGridExtensions.AutoGenerateDateTimeColumns" Value="True" />
                </Style>
            </DataGrid.Resources>
        </DataGrid>
    </Grid>
</UserControl>
```

3.5. SearchViewModelBase<T>: използва се като абстрактен базов клас за модели на изгледи, които имплементират функционалност за търсене.

Класът е дефиниран като SearchViewModelBase<T>, където T е типов параметър, който представя типа на търсените елементи. Класът е общ и е ограничен до референтен тип (където T : клас).

Той включва няколко полета и свойства, които се използват за управление на функционалността за търсене. Те включват полета за DbContext, полета за търсене, избрано поле за търсене, избрана стойност за търсене, контрол на избраната стойност за търсене (елемент на потребителския интерфейс), елементи, избрани филтри, избран филтър и филтрирани елементи. Свойствата, свързани с тези полета, осигуряват необходимото свързване на данните и предизвикват известия за промяна на свойствата.

Класът предоставя два конструктора. Първият конструктор инициализира DbContext, полетата за търсене, елементите, избраното поле за търсене, избраната стойност за търсене, избраната стойност за търсене, контролата за избраната стойност за търсене, избраните филтри и филтрираните елементи със стойности по подразбиране. Вторият конструктор позволява инжектиране на предварително инициализирана инстанция на DbContext.

Включва различни методи за поддръжка на функционалността за търсене. Тези методи включват инициализиране на полета за търсене, извличане на полета за търсене за даден тип обект, получаване на подходящ контрол на стойността на търсене въз основа на избраното поле за

търсене, обработчици на събития за промени в текста и промени в избора на дата, добавяне и премахване на филтри и получаване на символно представяне на стойност на свойство.

Дефинира три свойства ICommand за добавяне на филтър, премахване на филтър и премахване на избраната стойност за търсене. Тези команди се изпълняват с помощта на класа RelayCommand, който е потребителска реализация на интерфейса ICommand.

Класът също така имплементира интерфейса INotifyPropertyChanged и предизвиква събитието PropertyChanged, когато стойностите на свойствата се променят. Това позволява правилно свързване на данните и актуализиране на потребителския интерфейс при промяна на свойствата.

Като цяло SearchViewModelBase<T> осигурява основа за реализиране на функционалност за търсене в моделите на изгледи. Той обработва взаимодействието с базата данни, управлява полетата за търсене, филтрите и стойностите за търсене и предоставя методи и свойства за достъп и манипулиране на данните, свързани с търсенето. Производните класове могат да наследяват от този базов клас и да персонализират поведението му според нуждите.

```
using UniversalSearchCriteria_PS_IvanLeonov_43.Helpers;
using UniversalSearchCriteria_PS_IvanLeonov_43.Models;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;

namespace UniversalSearchCriteria_PS_IvanLeonov_43.ViewModels
{
    // This is an abstract base class for view models that implement search functionality
    // The type parameter 'T' represents the type of items being searched
    8 references
    public abstract class SearchViewModelBase<T> : INotifyPropertyChanged where T : class
    {
        protected readonly MyDbContext dbContext;
        protected List<SearchField> searchFields;
        protected SearchField selectedSearchField;
        protected object selectedSearchValue;
        protected UIElement selectedSearchValueControl;
        protected ObservableCollection<T> items;
        protected ObservableCollection<SearchFilter> selectedFilters;
        protected SearchFilter selectedFilter;
        protected ObservableCollection<T> filteredItems;

        public event PropertyChangedEventHandler PropertyChanged;

        2 references
        public SearchViewModelBase()
        {
            dbContext = new MyDbContext();
            InitializeSearchFields();
            Items = new ObservableCollection<T>();
            selectedSearchField = searchFields.FirstOrDefault();
            selectedSearchValue = null;
            selectedSearchValueControl = null;
            SelectedFilters = new ObservableCollection<SearchFilter>();
            Items = new ObservableCollection<T>(dbContext.Set<T>().ToList());
            FilteredItems = SearchHelper.ApplyFiltersAndSearch(dbContext, Items, SelectedFilters, selectedSearchField, selectedSearchValue);
        }
    }
}
```

```

// Overloaded constructor that allows injecting a pre-initialized DbContext instance
2 references
public SearchViewModelBase(MyDbContext dbContext)
{
    this.dbContext = dbContext;
    InitializeSearchFields();
    Items = new ObservableCollection<T>();
    selectedSearchField = searchFields.FirstOrDefault();
    selectedSearchValue = null;
    selectedSearchValueControl = null;
    SelectedFilters = new ObservableCollection<SearchFilter>();
    Items = new ObservableCollection<T>(dbContext.Set<T>().ToList());
    FilteredItems = SearchHelper.ApplyFiltersAndSearch(dbContext, Items, SelectedFilters, selectedSearchField, selectedSearchValue);
}

// Property for the list of search fields
1 reference
public List<SearchField> SearchFields
{
    get { return searchFields; }
    set { searchFields = value; OnPropertyChanged(nameof(SearchFields)); }
}

// Property for selected search field
7 references
public SearchField SelectedSearchField
{
    get { return selectedSearchField; }
    set
    {
        selectedSearchField = value;
        SelectedSearchValue = null; // Reset the selected search value
        SelectedSearchValueControl = GetSearchValueControl(selectedSearchField);
        OnPropertyChanged(nameof(SelectedSearchField));
        FilteredItems = SearchHelper.ApplyFiltersAndSearch(dbContext, Items, SelectedFilters, selectedSearchField, selectedSearchValue);
    }
}

// Property for the selected search value
8 references
public object SelectedSearchValue
{
    get { return selectedSearchValue; }
    set
    {
        selectedSearchValue = value;
        OnPropertyChanged(nameof(SelectedSearchValue));
        FilteredItems = SearchHelper.ApplyFiltersAndSearch(dbContext, Items, SelectedFilters, selectedSearchField, selectedSearchValue);
    }
}

// Property for the selected search value control (UI element)
4 references
public UIElement SelectedSearchValueControl
{
    get
    {
        if (selectedSearchValueControl == null)
        {
            selectedSearchValueControl = GetDefaultSearchValueControl();
        }
        return selectedSearchValueControl;
    }
    set { selectedSearchValueControl = value; OnPropertyChanged(nameof(SelectedSearchValueControl)); }
}

// Get the default search value control (UI element)
1 reference
private UIElement GetDefaultSearchValueControl()
{
    if (SelectedSearchField != null)
    {
        return GetSearchValueControl(SelectedSearchField);
    }
    var defaultControl = new TextBlock();
    defaultControl.Text = "Select a search field";
    return defaultControl;
}

```

```

// Property for the collection of items
11 references
public ObservableCollection<T> Items
{
    get { return items; }
    set { items = value; OnPropertyChanged(nameof(Items)); }
}

// Property for the collection of selected filters
11 references
public ObservableCollection<SearchFilter> SelectedFilters
{
    get { return selectedFilters; }
    set { selectedFilters = value; OnPropertyChanged(nameof(SelectedFilters)); }
}

// Property for the selected filter
4 references
public SearchFilter SelectedFilter
{
    get { return selectedFilter; }
    set
    {
        selectedFilter = value;
        OnPropertyChanged(nameof(SelectedFilter));
        OnPropertyChanged(nameof(SelectedFilterText));
    }
}

// Property for the text representation of the selected filter
1 reference
public string SelectedFilterText => SelectedFilter?.ToString();

// Property for the collection of filtered items
7 references
public ObservableCollection<T> FilteredItems
{
    get { return filteredItems; }
    set { filteredItems = value; OnPropertyChanged(nameof(FilteredItems)); }
}

```

```

// Initialize the list of search fields
2 references
protected virtual void InitializeSearchFields()
{
    searchFields = GetSearchFields(typeof(T));
}

// Method to retrieve the list of search fields for a given object type
1 reference
protected virtual List<SearchField> GetSearchFields(Type objectType)
{
    var searchFields = new List<SearchField>();
    var properties = objectType.GetProperties();

    foreach (var property in properties)
    {
        var searchField = new SearchField(property.Name, objectType, property.PropertyType);
        searchFields.Add(searchField);
    }
    return searchFields;
}

```



```

protected virtual List<SearchField> GetSearchFields(Type objectType)
{
    var searchFields = new List<SearchField>();
    var properties = objectType.GetProperties();

    foreach (var property in properties)
    {
        var searchField = new SearchField(property.Name, objectType, property.PropertyType);
        searchFields.Add(searchField);
    }

    return searchFields;
}

// Method to get the appropriate search value control (UI element) based on the selected search field
4 references
protected virtual UIElement GetSearchValueControl(SearchField searchField)
{
    // Create a TextBox control for string fields
    if (searchField.FieldType == typeof(string))
    {
        var textBox = new TextBox();
        textBox.TextChanged += TextBox_TextChanged;
        return textBox;
    }

    // Create a DatePicker control for DateTime fields
    else if (searchField.FieldType == typeof(DateTime))
    {
        var datePicker = new DatePicker();
        datePicker.SelectedDateChanged += DatePicker_SelectedDateChanged;
        return datePicker;
    }

    // Create a TextBox control for integer fields
    else if (searchField.FieldType == typeof(int))
    {
        var textBox = (new TextBox());
        textBox.TextChanged += TextBox_TextChanged;
        return textBox;
    }

    // Create a TextBox control for double fields
    else if (searchField.FieldType == typeof(double))
    {
        var textBox = (new TextBox());
        textBox.TextChanged += TextBox_TextChanged;
        return textBox;
    }

    else
    {
        // Return an appropriate UI element for other field types
        return new TextBlock();
    }
}

```

```

// Event handler for the text changed event of the text box
3 references
private void TextBox_TextChanged(object sender, TextChangedEventArgs e)
{
    SelectedSearchValue = ((TextBox)sender).Text;
}

// Event handler for the selected date changed event of the date picker
1 reference
private void DatePicker_SelectedDateChanged(object sender, SelectionChangedEventArgs e)
{
    SelectedSearchValue = ((DatePicker)sender).SelectedDate;
}

// Command for adding a filter
0 references
public ICommand AddFilterCommand => new RelayCommand(AddFilter);

// Method for adding a filter
1 reference
private void AddFilter(object parameter)
{
    if (SelectedSearchField != null && SelectedSearchValue != null)
    {
        var filter = new SearchFilter(SelectedSearchField, SelectedSearchValue);
        SelectedFilters.Add(filter);
        FilteredItems = SearchHelper.ApplyFiltersAndSearch(dbContext, Items, SelectedFilters, selectedSearchField, selectedSearchValue);
    }
    else
    {
        MessageBox.Show("Please select a search field and value.");
    }
}

// Command for removing a filter
0 references
public ICommand RemoveFilterCommand => new RelayCommand(RemoveFilter);

// Method for removing a filter
1 reference
private void RemoveFilter(object parameter)
{
    if (SelectedFilter != null)
    {
        SelectedFilters.Remove(SelectedFilter);
        FilteredItems = SearchHelper.ApplyFiltersAndSearch(dbContext, Items, SelectedFilters, selectedSearchField, selectedSearchValue);
        SelectedSearchValueControl = GetSearchValueControl(SelectedSearchField); // Update the search value control with the latest search field
        SelectedSearchValue = null;
    }
}

// Command for removing the selected search value
0 references
public ICommand RemoveSearchValueCommand => new RelayCommand(RemoveSearchValue);

// Method for removing the selected search value
1 reference
private void RemoveSearchValue(object parameter)
{
    SelectedSearchValue = null;
    SelectedSearchValueControl = GetSearchValueControl(SelectedSearchField); // Update the search value control with the latest search field
}

// Get the string representation of a property value
0 references
protected virtual string GetPropertyStringValue(T item, string propertyName)
{
    var property = typeof(T).GetProperty(propertyName);
    var value = property.GetValue(item);
    return value?.ToString() ?? string.Empty;
}

// Notify property changed event
9 references
protected virtual void OnPropertyChanged(string propertyName)
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
}

```

3.6. SearchBookViewModel и SearchAuthorViewModel:

Класовете SearchAuthorViewModel и SearchBookViewModel са дефинирани в пространството от namespace UniversalSearchCriteria_PS_IvanLeonov_43.ViewModels.

И двата ViewModels наследяват абстрактния SearchViewModelBase<T>, което им дава възможност да използват всички свойства, методи и цялостната функционалност за търсене, филтриране и т.н.

Съдържат два конструктора: конструктор по подразбиране и конструктор, който приема параметър ApplicationDbContext. И двата конструктора извикват съответните конструктори на базовия клас, като използват ключовата дума base(), предавайки аргументите на конструктора на базовия клас. Конструкторът по подразбиране няма никаква допълнителна логика и просто извиква конструктора по подразбиране на базовия клас. Конструкторът, който приема параметър ApplicationDbContext, позволява използването на предварително инициализирана инстанция на ApplicationDbContext в модела на изгледа. Това може да бъде полезно, за споделяне на една и съща инстанция на контекста на базата данни между няколко модела на изгледи или когато искаме да тестваме различни неща с контекста на базата данни.

Като цяло SearchAuthorViewModel и SearchBookViewModel разширяват базовата функционалност за търсене, предоставена от класа SearchViewModelBase<T>. Това позволява търсене и филтриране на автори въз основа на различни критерии и може да бъде интегриран в приложение, за да осигури възможности за търсене на автори/търсене на книги.

```

using UniversalSearchCriteria_PS_IvanLeonov_43.Models;

namespace UniversalSearchCriteria_PS_IvanLeonov_43.ViewModels
{
    2 references
    public class SearchAuthorViewModel : SearchViewModelBase<SearchAuthor>
    {
        0 references
        public SearchAuthorViewModel() : base()
        {
            // Default constructor of the SearchAuthorViewModel class
        }

        0 references
        public SearchAuthorViewModel(MyDbContext dbContext) : base(dbContext)
        {
            // Constructor of the SearchAuthorViewModel class that accepts a MyDbContext parameter
        }
    }
}

```

```

using UniversalSearchCriteria_PS_IvanLeonov_43.Models;

namespace UniversalSearchCriteria_PS_IvanLeonov_43.ViewModels
{
    2 references
    public class SearchBookViewModel : SearchViewModelBase<SearchBook>
    {
        0 references
        public SearchBookViewModel() : base()
        {
            // Default constructor of the SearchBookViewModel class
        }

        0 references
        public SearchBookViewModel(MyDbContext dbContext) : base(dbContext)
        {
            // Constructor of the SearchBookViewModel class that accepts a MyDbContext parameter
        }
    }
}

```

3.7. RelayCommand

Класът RelayCommand е обща реализация на интерфейса ICommand, използван в приложенията с шаблона MVVM (Model-View-ViewModel). Класът има две частни полета: `_execute` от тип `Action<object>` и `_canExecute` от тип `Func<object, bool>`. Тези полета съдържат препратки към делегатите, които ще бъдат изпълнени при извикване на командата. Класът съдържа събитието `CanExecuteChanged`, което се предизвиква, когато възможността за изпълнение на командата се промени. Събитието `CommandManager.RequerySuggested` служи за обработка на това събитие. Класът има конструктор, който приема делегат `Action<object>` за логиката за изпълнение и незадължителен делегат `Func<object, bool>` за логиката за изпълнение. Методът `CanExecute` определя дали командата може да бъде изпълнена с дадения параметър. Той проверява дали делегатът `_canExecute` е null или дали връща true, когато се извиква с параметъра. Методът `Execute` изпълнява командата с дадения параметър чрез извикване на делегата `_execute`.

Като цяло класът RelayCommand предоставя начин за дефиниране и обработка на команди в ViewModel чрез капсулиране на логиката за изпълнение и определяне на способността за изпълнение на команда. Той следва общата реализация на интерфейса ICommand, позволявайки

свойствата на ViewModel да бъдат свързани със свойствата на командата в потребителския интерфейс, като бутони или елементи от менюто, за да задействат конкретни събития, когато бъдат извикани.

```
using System;
using System.Windows.Input;

namespace UniversalSearchCriteria_PS_IvanLeonov_43.ViewModels
{
    4 references
    public class RelayCommand : ICommand
    {
        private readonly Action<object> _execute;
        private readonly Func<object, bool> _canExecute;

        // Event that is raised when the ability to execute the command changes
        public event EventHandler CanExecuteChanged
        {
            add { CommandManager.RequerySuggested += value; }
            remove { CommandManager.RequerySuggested -= value; }
        }

        // Constructor
        3 references
        public RelayCommand(Action<object> execute, Func<object, bool> canExecute = null)
        {
            _execute = execute ?? throw new ArgumentNullException(nameof(execute));
            _canExecute = canExecute;
        }

        // Determine if the command can execute with the given parameter
        0 references
        public bool CanExecute(object parameter)
        {
            return _canExecute == null || _canExecute(parameter);
        }

        // Execute the command with the given parameter
        0 references
        public void Execute(object parameter)
        {
            _execute(parameter);
        }
    }
}
```

3.8. SearchFilter

Този клас се използва за дефиниране на филтри за търсене в приложението. Свойствата SearchField и Value пазят връзка с полето за търсене и стойността, използвани за филтриране. Конструкторът приема параметри за SearchField и Value и ги задава в съответните свойства.

Методът ToString() се използва за предоставяне на текстово представяне на филтъра, което включва името на полето за търсене и стойността. Това е полезно за визуализиране на филтрите в

потребителския интерфейс или за отпечатване в конзолата.

```
namespace UniversalSearchCriteria_PS_IvanLeonov_43.Models
{
    9 references
    public class SearchFilter
    {
        // The search field associated with the filter
        3 references
        public SearchField SearchField { get; set; }

        // The value used for filtering
        3 references
        public object Value { get; set; }

        1 reference
        public SearchFilter(SearchField searchField, object value)
        {
            SearchField = searchField;
            Value = value;
        }

        // ToString method to show data in DataGrid with filters.
        1 reference
        public override string ToString()
        {
            return $"{SearchField.Name}: {Value}";
        }
    }
}
```

3.9. SearchField

Представява модел за поле за търсене. Този клас се използва за дефиниране на полета за търсене в приложението. Свойствата Name, ModelType, Fields и FieldType представят информация за полето за търсене. Има няколко конструктора, които позволяват създаването на полета за търсене.

Конструкторът SearchField(string name, Type modelType) създава основно поле за търсене с тип на стойността string. Конструкторът SearchField(string name, Type modelType, List<SearchField> fields) създава поле за търсене с вложени полета. Конструкторът SearchField(string name, Type modelType, Type fieldType) създава поле за търсене с конкретен тип на стойността.

Този клас е полезен при дефинирането на списък от полета за търсене, които могат да бъдат използвани за съставяне на филтри и търсене в приложението.

```
using System;
using System.Collections.Generic;

namespace UniversalSearchCriteria_PS_IvanLeonov_43.Models
{
    16 references
    public class SearchField
    {
        6 references
        public string Name { get; set; }

        // Type of the model associated with the search field
        3 references
        public Type ModelType { get; set; }

        1 reference
        public List<SearchField> Fields { get; set; }

        // Type of the field's value
        6 references
        public Type FieldType { get; set; }

        // Constructor for a basic search field with a string value type
        0 references
        public SearchField(string name, Type modelType)
        {
            Name = name;
            ModelType = modelType;
            FieldType = typeof(string);
        }

        // Constructor for a search field with nested fields
        0 references
        public SearchField(string name, Type modelType, List<SearchField> fields)
        {
            Name = name;
            ModelType = modelType;
            Fields = fields;
        }

        // Constructor for a search field with a specific value type
        1 reference
        public SearchField(string name, Type modelType, Type fieldType)
        {
            Name = name;
            ModelType = modelType;
            FieldType = fieldType;
        }
    }
}
```

3.10. DataGridExtensions

Този клас предоставя метод за разширение за контролата DataGrid в WPF за автоматично генериране и форматиране на колони за свойствата DateTime. Той е статичен, което показва, че той е клас за разширение и не може да бъде инстанциран. Съдържа само статични членове, включително метода за разширение и свързаните с него помощни методи. Дефинира свойство за зависимост, наречено AutoGenerateDateTimeColumnsProperty, което позволява прикрепянето на булева стойност към контролата DataGrid. Това свойство ще определи дали автоматичното генериране на колони DateTime е разрешено или забранено. Дефинирани са и два допълнителни метода: GetAutoGenerateDateTimeColumns и SetAutoGenerateDateTimeColumns. Тези методи предоставят getter и setter за AutoGenerateDateTimeColumnsProperty. Методът

OnAutoGenerateDateTimeColumnsChanged е регистриран като обратна връзка за промени в свойствата на AutoGenerateDateTimeColumnsProperty. Той се изпълнява при промяна на стойността на свойството. Този метод обработва логиката за свързване или отделяне на обработчика на събитието въз основа на новата стойност. Методът DataGrid_AutoGeneratingColumn е обработчикът на събитието за събитието AutoGeneratingColumn на DataGrid. Той се извиква, когато колоните се генерират автоматично. Този метод проверява дали типът на свойството, което се свързва с колоната, е DateTime. Ако типът на свойството е DateTime, се създава нова колона DataGridTextBoxColumn. Заглавието и обвързването на колоната се задават на съответните стойности от автоматично генерираната колона. Освен това към връзката се прилага StringFormat, за да се форматира стойностите на DateTime като "dd/MM/yyyy". Накрая автоматично генерираната колона се заменя с новосъздадената колона DataGridTextBoxColumn, като се гарантира, че колоните DateTime се показват с желания формат. С използването на този метод на разширение, може да се активира автоматичното генериране и форматиране на колони DateTime в контролата DataGrid, като това се случва със задаване на приложеното свойство AutoGenerateDateTimeColumns на true. Това опростява процеса на персонализиране на показването на DateTime свойствата в DataGrid, без да се налага изрично ръчно дефиниране на колоните.


```

using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;

namespace UniversalSearchCriteria_PS_IvanLeonov_43.Extensions
{
    1 reference
    public static class DataGridExtensions
    {
        // DependencyProperty for the AutoGenerateDateTimeColumns property
        public static readonly DependencyProperty AutoGenerateDateTimeColumnsProperty =
            DependencyProperty.RegisterAttached(
                "AutoGenerateDateTimeColumns",
                typeof(bool),
                typeof(DataGridExtensions),
                new PropertyMetadata(false, OnAutoGenerateDateTimeColumnsChanged));

        // Getter for AutoGenerateDateTimeColumns property
        0 references
        public static bool GetAutoGenerateDateTimeColumns(DependencyObject obj)
        {
            return (bool)obj.GetValue(AutoGenerateDateTimeColumnsProperty);
        }

        // Setter for AutoGenerateDateTimeColumns property
        0 references
        public static void SetAutoGenerateDateTimeColumns(DependencyObject obj, bool value)
        {
            obj.SetValue(AutoGenerateDateTimeColumnsProperty, value);
        }

        // Event handler for the AutoGenerateDateTimeColumns property change
        1 reference
        private static void OnAutoGenerateDateTimeColumnsChanged(DependencyObject d, DependencyPropertyChangedEventArgs e)
        {
            if (d is DataGrid dataGrid)
            {
                if ((bool)e.NewValue)
                {
                    // Subscribe to the AutoGeneratingColumn event when the property is set to true
                    dataGrid.AutoGeneratingColumn += DataGrid_AutoGeneratingColumn;
                }
                else
                {
                    // Unsubscribe from the AutoGeneratingColumn event when the property is set to false
                    dataGrid.AutoGeneratingColumn -= DataGrid_AutoGeneratingColumn;
                }
            }
        }

        // Event handler for the AutoGeneratingColumn event
        2 references
        private static void DataGrid_AutoGeneratingColumn(object sender, DataGridAutoGeneratingColumnEventArgs e)
        {
            if (e.PropertyType == typeof(DateTime))
            {
                // Create a new DataGridTextColumn for DateTime properties
                var dateColumn = new DataGridTextColumn();

                // Set the column header to the same as the original column
                dateColumn.Header = e.Column.Header;

                // Set the column's binding to the same as the original column
                dateColumn.Binding = new Binding(e.PropertyName);

                // Apply the desired date format to the binding
                dateColumn.Binding.StringFormat = "dd/MM/yyyy";

                // Replace the original column with the new date column
                e.Column = dateColumn;
            }
        }
    }
}

```

3.11. Search Helper

Този клас предоставя методи за прилагане на филтри и извършване на търсения върху колекция от елементи. Включва метод `ApplyFiltersAndSearch`. Този метод приема различни параметри, включително инстанция на `DbContext`, колекция от елементи, избрани филтри, избрано поле за търсене и избрана стойност за търсене. Той прилага избраните филтри към колекцията, като филтрира елементите, които не отговарят на критериите на филтъра. Той също така извършва търсене въз основа на избраното поле за търсене и стойност, като филтрира елементите, които не съдържат стойността за търсене в посоченото свойство. След това филтрираните елементи се преобразуват в `ObservableCollection` и се връщат.

Метод `GetPropertyStringValue`: Този помощен метод извлича символната стойност на дадено свойство от обект. Той приема елемент и името на свойството и използва отражение, за да получи стойността на това свойство от елемента. Ако стойността не е `null`, той връща нейното символно представяне в низ; в противен случай връща празен низ.

Като цяло, класът `"SearchHelper"` предоставя помощни методи за прилагане на филтри и извършване на търсения върху колекция от елементи въз основа на избрани филтри, полета за търсене и стойности за търсене. Тези методи могат да се използват от моделите на изгледи, за да реализират функционалността за търсене и съответно да актуализират колекцията от филтрирани елементи.

```

using UniversalSearchCriteria_PS_IvanLeonov_43.Models;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;

namespace UniversalSearchCriteria_PS_IvanLeonov_43.Helpers
{
    6 references
    public static class SearchHelper
    {
        // Method to apply filters and perform search on a collection of items
        6 references
        public static ObservableCollection<T> ApplyFiltersAndSearch<T>(MyDbContext dbContext,
            IEnumerable<T> items, IEnumerable<SearchFilter> selectedFilters, SearchField selectedSearchField,
            object selectedSearchValue)
            where T : class
        {
            var filteredItems = items.ToList();

            // Apply selected filters to the collection
            foreach (var filter in selectedFilters)
            {
                var propertyName = filter.SearchField.Name;
                var propertyValue = filter.Value.ToString().ToLower();

                var property = typeof(T).GetProperty(propertyName);

                // Filter the items based on the selected filter criteria
                filteredItems = filteredItems.Where(item =>
                {
                    var propertyStringValue = GetPropertyStringValue(item, propertyName);
                    return propertyStringValue != null && propertyStringValue.ToLower().Contains(propertyValue);
                }).ToList();
            }

            // Perform search based on selected search field and value
            if (selectedSearchField != null && selectedSearchValue != null)
            {
                var searchPropertyName = selectedSearchField.Name;
                var searchValue = selectedSearchValue.ToString().ToLower();

                // Filter the items based on the search field and value
                filteredItems = filteredItems.Where(item =>
                {
                    var propertyStringValue = GetPropertyStringValue(item, searchPropertyName);
                    return propertyStringValue != null && propertyStringValue.ToLower().Contains(searchValue);
                }).ToList();
            }

            // Convert the filtered items to an ObservableCollection and return
            return new ObservableCollection<T>(filteredItems);
        }
    }

    // Helper method to retrieve the string value of a property from an object
    2 references
    private static string GetPropertyStringValue<T>(T item, string propertyName)
    {
        var property = typeof(T).GetProperty(propertyName);
        var value = property.GetValue(item);

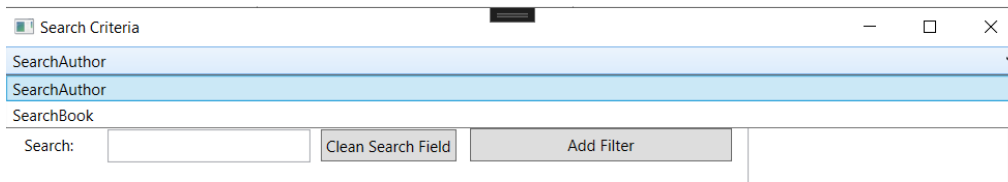
        return value?.ToString() ?? string.Empty;
    }
}

```

4. Ръководството за потребителя - така предложеното решение как се ползва през графичния интерфейс. Тук се демонстрират и функционалните изисквания от заданието

За да използвате системата за динамично търсене по даден критерий, следвайте следните стъпки:

- 1.Стартирайте приложението, като стартирате изпълнимия файл.
- 2.При отваряне на главния прозорец ще видите ComboBox (списък с модели), на който от падащото меню може да направите избор на типа на обекта (в случая книга или автор).



Search Criteria

SearchAuthor

SearchAuthor

SearchBook

Search:

Id	Name	Country	BirthDate	
1	J.K. Rowling	UK	31/07/1965	
2	Stephen King	USA	21/09/1947	
3	Agatha Christie	UK	15/09/1890	
4	Charles Dickens	UK	07/02/1812	
5	Leo Tolstoy	Russia	09/09/1828	
6	Mark Twain	USA	30/11/1835	
7	George R.R. Martin	USA	20/09/1948	
8	J.R.R. Tolkien	UK	03/01/1892	
9	Virginia Woolf	UK	25/01/1882	
10	Ernest Hemingway	USA	21/07/1899	
11	Fyodor Dostoevsky	Russia	11/11/1821	
12	Oscar Wilde	Ireland	16/10/1854	

3.Изберете желания тип обект и съответният интерфейс за търсене ще се покаже по-долу.

Search Criteria

SearchBook

Search By: Id Remove Filter

Search: Clean Search Field Add Filter

Id	Title	Author	Status	PublicationDate	
1	The Hobbit	J.R.R. Tolkien	Available	21/09/1937	
2	The Old Man and the Sea	Ernest Hemingway	CheckedOut	01/09/1952	
3	Brave New World	Aldous Huxley	Available	17/06/1932	
5	To the Lighthouse	Virginia Woolf	Available	05/05/1927	
7	Moby-Dick	Herman Melville	Available	18/10/1851	
8	The Catcher in the Rye	J.D. Salinger	CheckedOut	16/07/1951	
17	To the Lighthouse	Virginia Woolf	Available	05/05/1927	
21	The Lord of the Rings	J.R.R. Tolkien	Available	29/07/1954	
22	To Kill a Mockingbird	Harper Lee	Available	11/07/1960	
33	The Great Gatsby	F. Scott Fitzgerald	Available	10/04/1925	
34	1984	George Orwell	Checked Out	08/06/1949	
35	Pride and Prejudice	Jane Austen	Available	28/01/1813	

Search Criteria

SearchAuthor

Search By: Id Remove Filter

Search: Clean Search Field Add Filter

	ID	Name	Country	BirthDate	
1	J.K. Rowling	UK	31/07/1965		
2	Stephen King	USA	21/09/1947		
3	Agatha Christie	UK	15/09/1890		
4	Charles Dickens	UK	07/02/1812		
5	Leo Tolstoy	Russia	09/09/1828		
6	Mark Twain	USA	30/11/1835		
7	George R.R. Martin	USA	20/09/1948		
8	J.R.R. Tolkien	UK	03/01/1892		
9	Virginia Woolf	UK	25/01/1882		
10	Ernest Hemingway	USA	21/07/1899		
11	Fyodor Dostoevsky	Russia	11/11/1821		
12	Oscar Wilde	Ireland	16/10/1854		

4. След като сте избрали правилния обект, трябва да виждате ComboBox за търсене а до него и избор на критерии, по който да търсите. Отдолу трябва да виждате списък със съответните неща, които са в базата данни, които ще бъдат с автоматично генерирани колонии.

5. След като виждате списък с елементите в базата данни, изберете поле за търсене от ComboBox (падащото меню) в интерфейса за търсене и изберете критериите за търсене.

Search Criteria

SearchBook

Search By: Author

Id

Title

Author

Status

PublicationDate

Clean Search Field

Add Filter

Remove Filter

Id	Title	Author	Status	PublicationDate
1	The Hobbit	J.R.R. Tolkien	Available	21/09/1937
2	The Old Man and the Sea	Ernest Hemingway	CheckedOut	01/09/1952
3	Brave New World	Aldous Huxley	Available	17/06/1932
5	To the Lighthouse	Virginia Woolf	Available	05/05/1927
7	Moby-Dick	Herman Melville	Available	18/10/1851
8	The Catcher in the Rye	J.D. Salinger	CheckedOut	16/07/1951
17	To the Lighthouse	Virginia Woolf	Available	05/05/1927
21	The Lord of the Rings	J.R.R. Tolkien	Available	29/07/1954
22	To Kill a Mockingbird	Harper Lee	Available	11/07/1960
33	The Great Gatsby	F. Scott Fitzgerald	Available	10/04/1925
34	1984	George Orwell	Checked Out	08/06/1949
35	Pride and Prejudice	Jane Austen	Available	28/01/1813

Search Criteria

SearchAuthor

Search By: Id

Id

Name

Country

BirthDate

Clean Search Field

Add Filter

Remove Filter

Id	Name	Country	BirthDate
1	J.K. Rowling	UK	31/07/1965
2	Stephen King	USA	21/09/1947
3	Agatha Christie	UK	15/09/1890
4	Charles Dickens	UK	07/02/1812
5	Leo Tolstoy	Russia	09/09/1828
6	Mark Twain	USA	30/11/1835
7	George R.R. Martin	USA	20/09/1948
8	J.R.R. Tolkien	UK	03/01/1892
9	Virginia Woolf	UK	25/01/1882
10	Ernest Hemingway	USA	21/07/1899
11	Fyodor Dostoevsky	Russia	11/11/1821
12	Oscar Wilde	Ireland	16/10/1854

6. След като изберете даденото поле по което искате да филтрирате, ще виждате поле за търсене, в което може да въведете стойността, по която искате да търсите, като търсенето ще се случва в реално време. Real-time searching-a, както и прилагането на филтрите са направени non-case sensitive, което показва, че няма значение как въвеждате стойността на полето, която искате да намерите/филтрирате. Също така те са направени на база Contains, което означава, че може да въведете само част от дума, която искате да видите, и резултатът ще е видим в реално време.

Search Criteria

SearchBook

Search By: Author

Remove Filter

Search: J.R.R. Tolkien

Clean Search Field

Add Filter

Id	Title	Author	Status	PublicationDate	
1	The Hobbit	J.R.R. Tolkien	Available	21/09/1937	
21	The Lord of the Rings	J.R.R. Tolkien	Available	29/07/1954	
43	The Hobbit	J.R.R. Tolkien	Available	21/09/1937	

7. За да приложите филтъра, щракнете върху бутона "Add Filter" (Добавяне на филтър), като е възможно да добавяте няколко филтъра последователно по различни критерии.

8. Филтрираните резултати от търсенето ще бъдат показани в списъка с данни.

Search Criteria

SearchBook

Search By: Status

Remove Filter

Author: J.R.R. Tolkien

Search:

Clean Search Field

Add Filter

Id	Title	Author	Status	PublicationDate
1	The Hobbit	J.R.R. Tolkien	Available	21/09/1937
21	The Lord of the Rings	J.R.R. Tolkien	Available	29/07/1954
43	The Hobbit	J.R.R. Tolkien	Available	21/09/1937

Search Criteria

SearchBook

Search By: Status

Remove Filter

Author: J.R.R. Tolkien
Status: Available

Search: Available

Clean Search Field

Add Filter

Id	Title	Author	Status	PublicationDate
1	The Hobbit	J.R.R. Tolkien	Available	21/09/1937
21	The Lord of the Rings	J.R.R. Tolkien	Available	29/07/1954
43	The Hobbit	J.R.R. Tolkien	Available	21/09/1937

9. За да премахнете филтър, изберете го от полето със списъка с филтри и щракнете върху бутона "Remove Filter" (Премахване на филтър).

Search Criteria

SearchBook

Search By: Status

Remove Filter

Status: Available

Search:

Clean Search Field

Add Filter

Id	Title	Author	Status	PublicationDate
1	The Hobbit	J.R.R. Tolkien	Available	21/09/1937
3	Brave New World	Aldous Huxley	Available	17/06/1932
5	To the Lighthouse	Virginia Woolf	Available	05/05/1927
7	Moby-Dick	Herman Melville	Available	18/10/1851
17	To the Lighthouse	Virginia Woolf	Available	05/05/1927
21	The Lord of the Rings	J.R.R. Tolkien	Available	29/07/1954
22	To Kill a Mockingbird	Harper Lee	Available	11/07/1960
33	The Great Gatsby	F. Scott Fitzgerald	Available	10/04/1925
35	Pride and Prejudice	Jane Austen	Available	28/01/1813
36	Crime and Punishment	Fyodor Dostoevsky	Available	05/02/1866
38	War and Peace	Leo Tolstoy	Available	01/01/1869
39	The Picture of Dorian Gray	Oscar Wilde	Available	01/07/1890

5. Указания за използване/внедряване - необходимите стъпки от бъдещ програмист за прилагане на предложената система с ръководещи примерни снипети от код.

Следвайки указанията за прилагане на предложената система за извеждане на контроли за потребителя, в които да се въвеждат критерии за търсене във вашата система за управление на библиотеката, можете да използвате следните стъпки:

Създаване на среда за разработка:

- Инсталирайте необходимия софтуер и библиотеки за разработване на WPF приложение.
- Създайте нов проект в предпочитаната от вас интегрирана среда за разработка (IDE).

Създаване на необходимите класове и пространства от имена (namespaces):

- Създайте namespace "ViewModels", който ще съдържа класовете на моделите на изгледа.
- Създайте абстрактен клас "SearchViewModelBase<T>" в пространството от имена "ViewModels", който служи като базов клас за моделите на изгледи, реализиращи функционалност за търсене. Важно е да създадем и статичен SearchHelper клас, за да можем да използваме функционалността за търсене и филтриране и следователно да използваме namespace Helper.
- Създайте допълнителни класове за модели на изгледи (ViewModels), които наследяват от "SearchViewModelBase<T>" и осигуряват специфични функционалности за търсене за различни обекти в зависимост от вашата цел.

Реализиране на класа "SearchViewModelBase<T>":

- Импортирайте необходимите пространства от имена, включително "System", "System.Collections.Generic", "System.Collections.ObjectModel", UniversalSearchCriteria_PS_IvanLeonov_43.Helper и други.
- Дефинирайте класа с подходящ модификатор за достъп и наследяване от "INotifyPropertyChanged".
- Добавете необходимите полета, свойства и колекции, за да съхранявате данните, свързани с търсенето, филтрите и резултатите от търсенето.
- Реализирайте конструкторите, включително един, който позволява инжектиране на предварително инициализирана инстанция на DbContext, ако е приложимо.

- Реализирайте методи за инициализиране на полета за търсене, извличане на полета за търсене за даден тип обект и получаване на контроли за стойности за търсене въз основа на избрани полета за търсене.
- Реализирайте обработчици на събития за промени в стойността на търсенето, като "TextBox_TextChanged" и "DatePicker_SelectedDateChanged".
- Добавете команди и съответните методи за добавяне и премахване на филтри.
- Реализиране на статичния клас SearchHelper, който съдържа метода "ApplyFiltersAndSearch" за прилагане на избраните филтри и критерии за търсене за филтриране на колекцията от елементи, а с него и метода "GetPropertyStringValue", за символното представяне на стойността на дадено свойство
- Реализирайте метода "OnPropertyChanged", за да предизвикате събитие за промяна на свойството.

Реализиране на специфичните класове на модела на изгледа:

- Създайте класове като "SearchAuthorViewModel" в пространството от имена "ViewModels" и наследете ги от "SearchViewModelBase<T>", като посочите подходящия тип на същността (например "SearchAuthor") като параметър "T". Ако е необходимо, реализирайте конструктори, като извикате конструктора на базовия клас със съответните параметри.

Реализиране на графичния потребителски интерфейс:

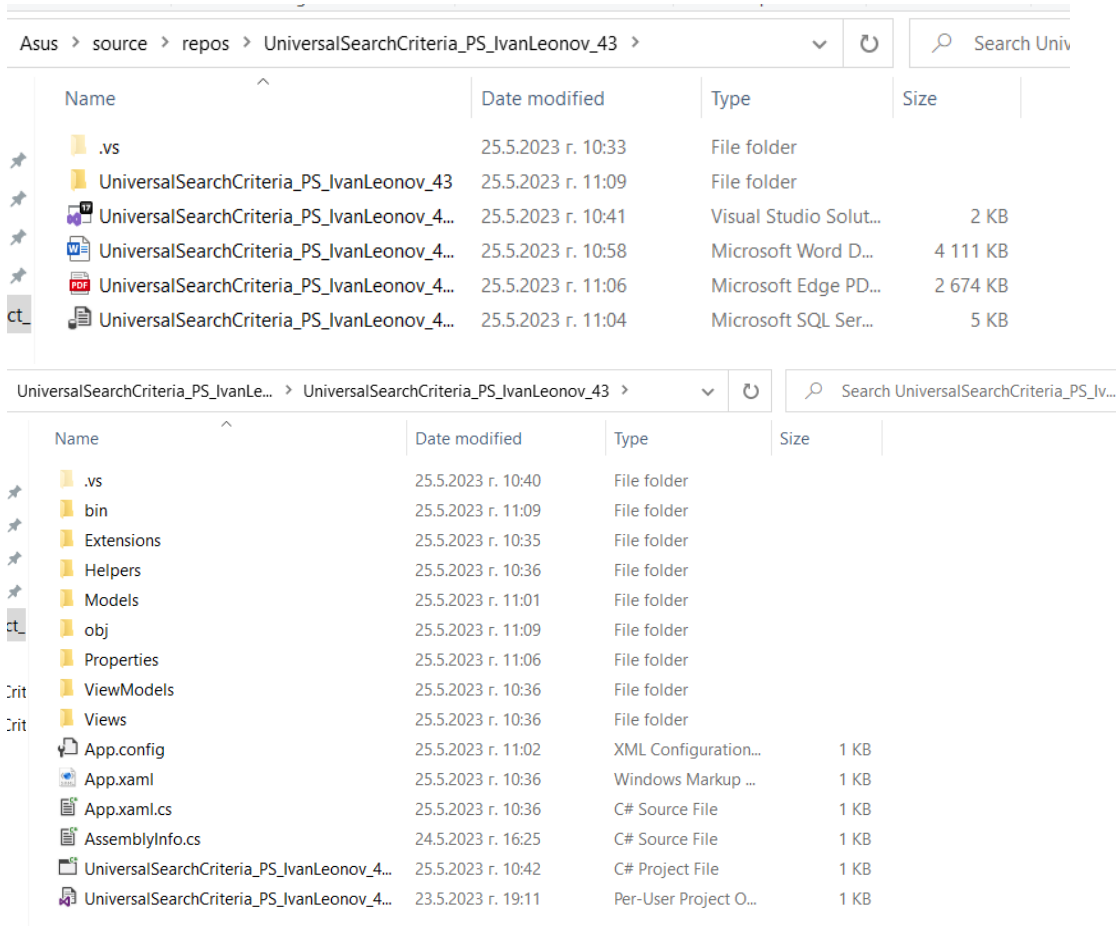
- Създайте изгледи в XAML (потребителски контроли, прозорци), съответстващи на създадените модели на изгледи.
- Проектирайте оформлението на графичния потребителски интерфейс, включително полетата за търсене, контролите за въвеждане, решетката с данни за показване на резултатите от търсенето и бутоните за добавяне/премахване на филтри.
- Свържете елементите на изгледа със свойствата и командите в моделите на изгледа.

Интегриране със системата за управление на библиотеката:

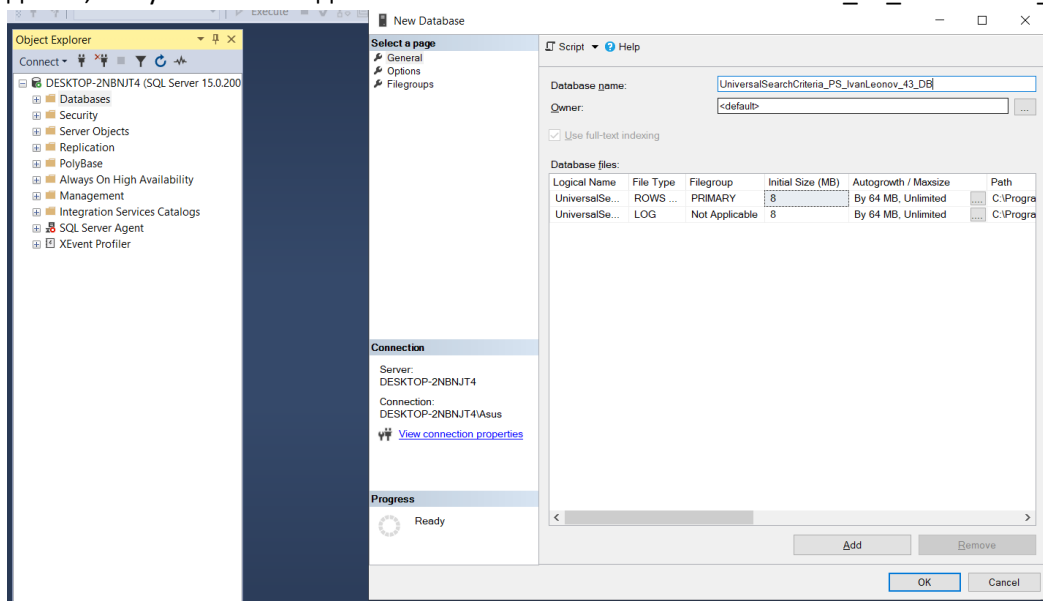
- Свържете се с подходящ източник на данни или база данни, като използвате Entity Framework или подобна рамка.
- Настройте връзката с базата данни, като промените connection string-a в настройките на проекта. (Properties/Settings.Settings).
- Може да използвате миграции, ако е необходимо, за да приложите промените във вашата база данни.

Демонстрация на свързването на проекта с Microsoft SQL Server Management:

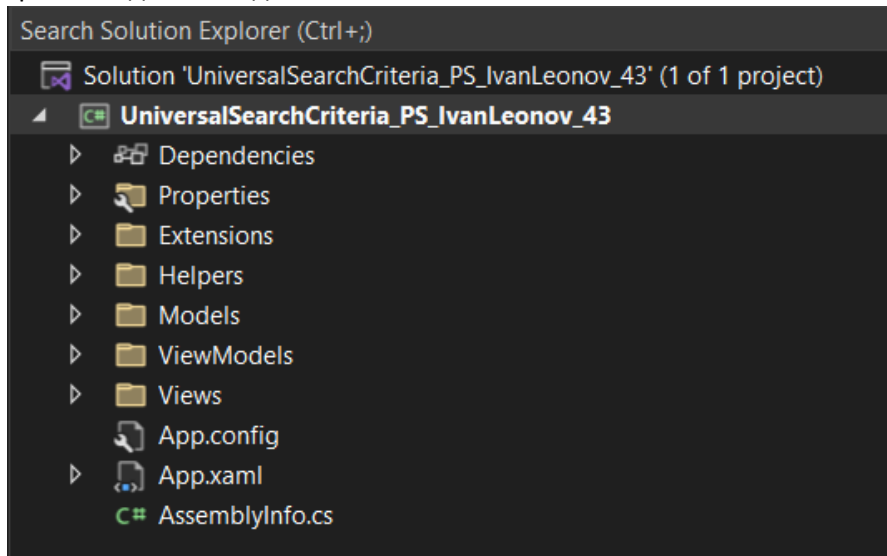
1. Изтегляме си проекта. Папката с проекта след изтеглянето:



2. Добра идея е първо да си отворим Microsoft SQL Server Management и да си създадем база данни, в случая аз си създавам такава с име UniversalSearchCriteria_PS_IvanLeonov_43_DB:



2. Сега отваряме проекта, като препоръчително е да отворите Solution File, след отварянето би трябвало да изглежда по този начин:



3. Отиваме в Properties/Settings.Settings и натискаме върху Value :

Name	Type	Scope	Value
DbConnection	(Connection...)	Application	Data Source=DESKTOP-2NBNJT4\Initial Catalog=UniversalSearchCriteria_PS_IvanLeonov_43_DB;Integrated Security=True
Setting	string	User	

4. Важно е да променим Server name, спрямо името на нашия Microsoft SQL Server Managemt сървър и си избираме базата данни, която искаме да използваме, след което натискаме OK и стартираме нашето приложение.

Connection Properties

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Microsoft SQL Server (SqlClient) Change...

Server name: DESKTOP-2NBNJT4 Refresh

Log on to the server

Authentication: Windows Authentication

User name: Password: Save my password

Connect to a database

Select or enter a database name: UniversalSearchCriteria_PS_IvanLeonov_43_DB

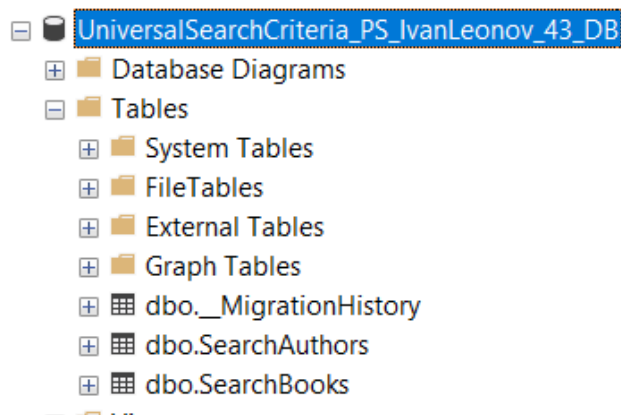
Attach a database file: Browse...

Logical name:

Advanced...

Test Connection OK Cancel

5. Ако всичко е преминало успешно и приложението е стартирало, трябва да виждаме базата данни с три таблици – SearchAuthors, SearchBooks, MigrationHistory, които в момента са празни, но сега ще ги напълним с нашия файл със заявки:



6. Отваряме UniversalSearchCriteria_PS_IvanLeonov_43_MicrosoftSQL_Query от нашата папка с проекта, като след отварянето трябва да изглежда по следния начин:

```
UniversalSearchCri...2NBNJT4\Asus (53))  X
USE UniversalSearchCriteria_PS_IvanLeonov_43_DB;

-- Inserting books
INSERT INTO SearchBooks (Id, Title, Author, Status, PublicationDate)
VALUES
(5, 'To the Lighthouse', 'Virginia Woolf', 'Available', '1927-05-05'),
(7, 'Moby-Dick', 'Herman Melville', 'Available', '1851-10-18'),
(8, 'The Catcher in the Rye', 'J.D. Salinger', 'Checked Out', '1951-07-16'),
(21, 'The Lord of the Rings', 'J.R.R. Tolkien', 'Available', '1954-07-29'),
(3, 'Brave New World', 'Aldous Huxley', 'Available', '1932-06-17'),
(1, 'The Hobbit', 'J.R.R. Tolkien', 'Available', '1937-09-21'),
(17, 'To the Lighthouse', 'Virginia Woolf', 'Available', '1927-05-05'),
(2, 'The Old Man and the Sea', 'Ernest Hemingway', 'Checked Out', '1952-09-01'),
(76, 'The Odyssey', 'Homer', 'Available', '1650-04-02'),
(99, 'One Hundred Years of Solitude', 'Gabriel Garcia Marquez', 'Available', '1967-05-30'),
(77, 'Jane Eyre', 'Charlotte Bronte', 'Available', '1847-10-16'),
(98, 'The Grapes of Wrath', 'John Steinbeck', 'Available', '1939-04-14'),
(33, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Available', '1925-04-10'),
(22, 'To Kill a Mockingbird', 'Harper Lee', 'Available', '1960-07-11'),
(34, '1984', 'George Orwell', 'Checked Out', '1949-06-08'),
(35, 'Pride and Prejudice', 'Jane Austen', 'Available', '1813-01-28'),
(36, 'Crime and Punishment', 'Fyodor Dostoevsky', 'Available', '1866-02-05'),
(37, 'The Stranger', 'Albert Camus', 'Checked Out', '1942-06-19'),
(38, 'War and Peace', 'Leo Tolstoy', 'Available', '1869-01-01'),
(39, 'The Picture of Dorian Gray', 'Oscar Wilde', 'Available', '1890-07-01'),
(40, 'The Adventures of Huckleberry Finn', 'Mark Twain', 'Available', '1884-12-10'),
(41, 'Frankenstein', 'Mary Shelley', 'Available', '1818-01-01'),
(42, 'The Divine Comedy', 'Dante Alighieri', 'Available', '1320'),
(44, 'The Alchemist', 'Paulo Coelho', 'Available', '1988-01-01'),
(45, 'The Book Thief', 'Markus Zusak', 'Available', '2005-01-01'),
(46, 'Les Miserables', 'Victor Hugo', 'Checked Out', '1862-03-30'),
(47, 'The Count of Monte Cristo', 'Alexandre Dumas', 'Available', '1844-01-28');

-- Inserting authors
INSERT INTO SearchAuthors (Id, Name, Country, BirthDate)
VALUES
(1, 'J.K. Rowling', 'UK', '1965-07-31'),
(2, 'Stephen King', 'USA', '1947-09-21'),
(3, 'Agatha Christie', 'UK', '1890-09-15'),
(4, 'Charles Dickens', 'UK', '1812-02-07'),
(5, 'Leo Tolstoy', 'Russia', '1828-09-09'),
(6, 'Mark Twain', 'USA', '1835-11-30'),
(7, 'George R.R. Martin', 'USA', '1948-09-20'),
(8, 'J.R.R. Tolkien', 'UK', '1892-01-03'),
```

Важно! Ако използваме база данни с друго име, трябва да променим името след USE да съвпада с базата, която използваме.

7. Натискаме Execute и ако всичко е правилно, сме добавили успешно данните в базата данни и вече може да ги използваме пълноценно в нашето приложение и да търсим и филтрираме по тях. Готови сме за работа 😊

6. Обобщение на резултатите и предложения за бъдещо развитие

Обобщение на резултатите:

- Системата за извеждане на контроли за потребителя, в които да се въвеждат критерии за търсене предоставя удобен графичен потребителски интерфейс за търсене на обекти (в случая книги и автори) по определени критерии. Тя позволява на потребителите да изберат типа на обекта, да изберат полето за търсене и да приложат филтри, за да покажат съответните резултати от търсенето в мрежа с данни. Системата е проектирана така, че да следва архитектурния модел MVVM, осигуряващ разделение на проблемите и възможност за поддръжка. С помощта на `SearchViewModelBase<T>`, и Helper клас – `SearchHelper`, можем без проблем да използваме нашата система за извличане на всякаква информация от различни бази данни и да търсим по различни критерии. Нужно е да наследим този абстрактен клас и след това да си настроим базата с правилния `connection string` и да използваме подходящите `namespace`. Важно е да използваме моделите `SearchField.cs` и `SearchFilter.cs`, защото те са тези, които са в основата на функционалността на `SearchViewModelBase<T>`.

Предложения за бъдещо развитие:

- Въвеждане на удостоверяване и оторизиране на потребителите за защита на системата.
- Интеграция на системата с външни API или източници на данни, за подобряване на базата данни.
- Добавяне на допълнителни функции, като заемане и връщане на книги, препоръки за книги и потребителски прегледи, свързване на Автор с книга.
- Въвеждане на разширени опции за търсене, включително разширено търсене или разширени филтри.
- Добавяне поддръжка за CRUD операции, позволяващи на потребителите да добавят, редактират и изтриват записи на книги и автори.
- Добавяне на обработката на грешки `event handler` и валидирането на потребителските данни.