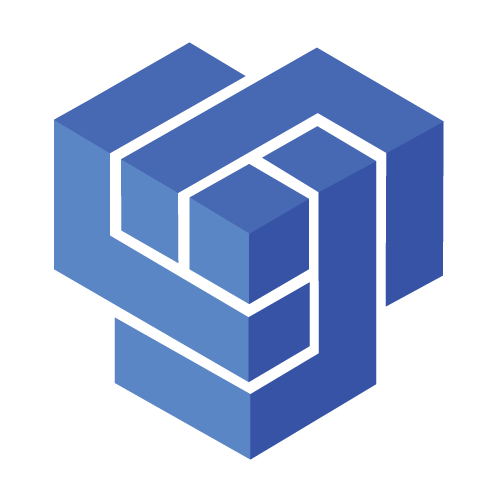
****

**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/таблици/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 функционалността за търсене.

**1.2 Основни класове/таблици/структури**

**1.2.1. Models (Модели):**

* SearchBook (Търсене на книга): Представлява книга в библиотеката със свойства като заглавие, автор, дата на публикуване, статус.
* SearchAuthor (Търсене на автор): Представлява 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): Представлява 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-релационно 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): Представляват 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 DataGrid 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 MyDbContext 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 променят в реално време спрямо действията на потребителя.

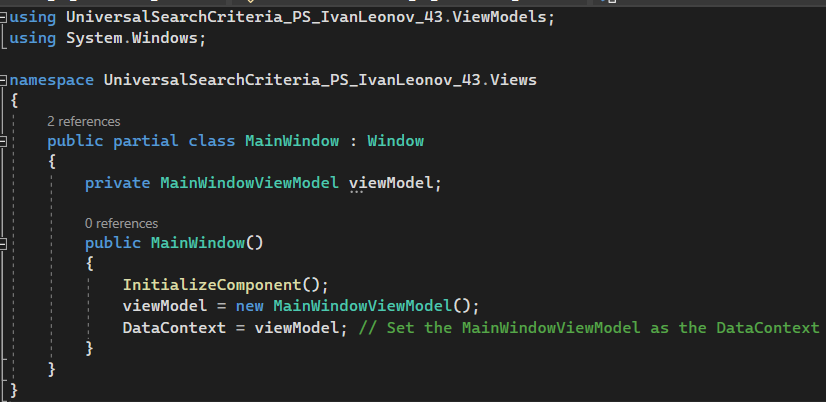
**2.3.4. Разширяемост и възможност за поддържане**

* 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, снипети 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.

****

**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.

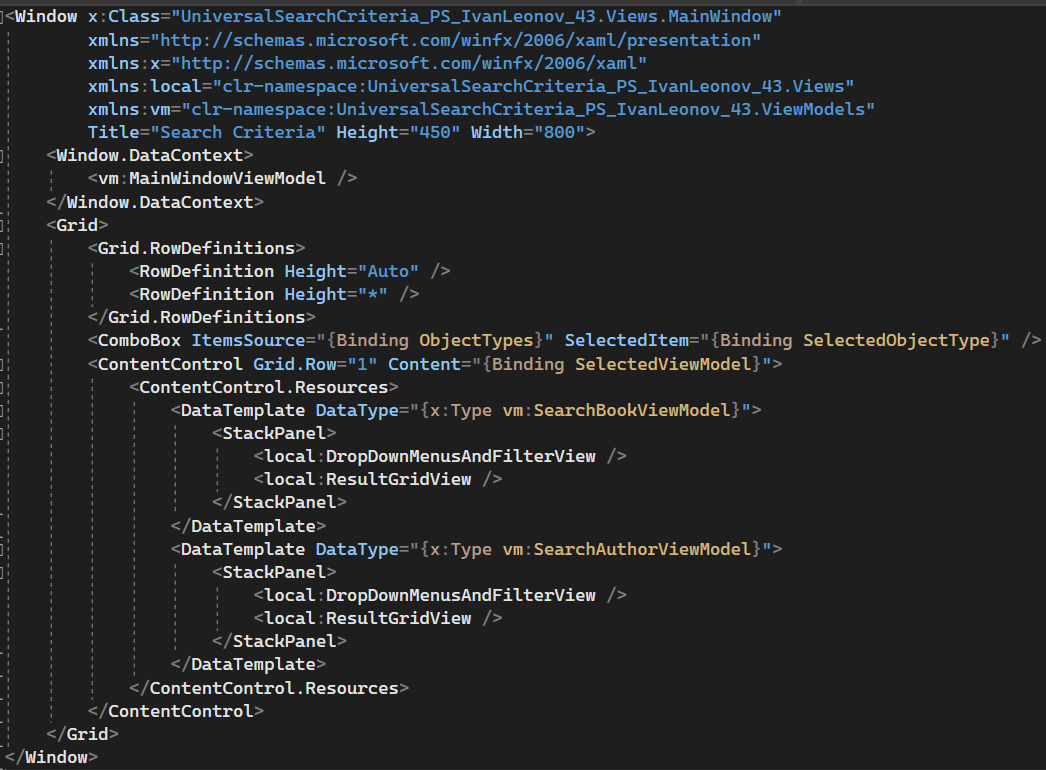
Основният прозорец е дефиниран със заглавие "Search Criteria" (Критерии за търсене) и е с определена височина и ширина. Свойството DataContext е зададено на инстанция на MainWindowViewModel, която служи като контекст на данните за прозореца.

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.

****

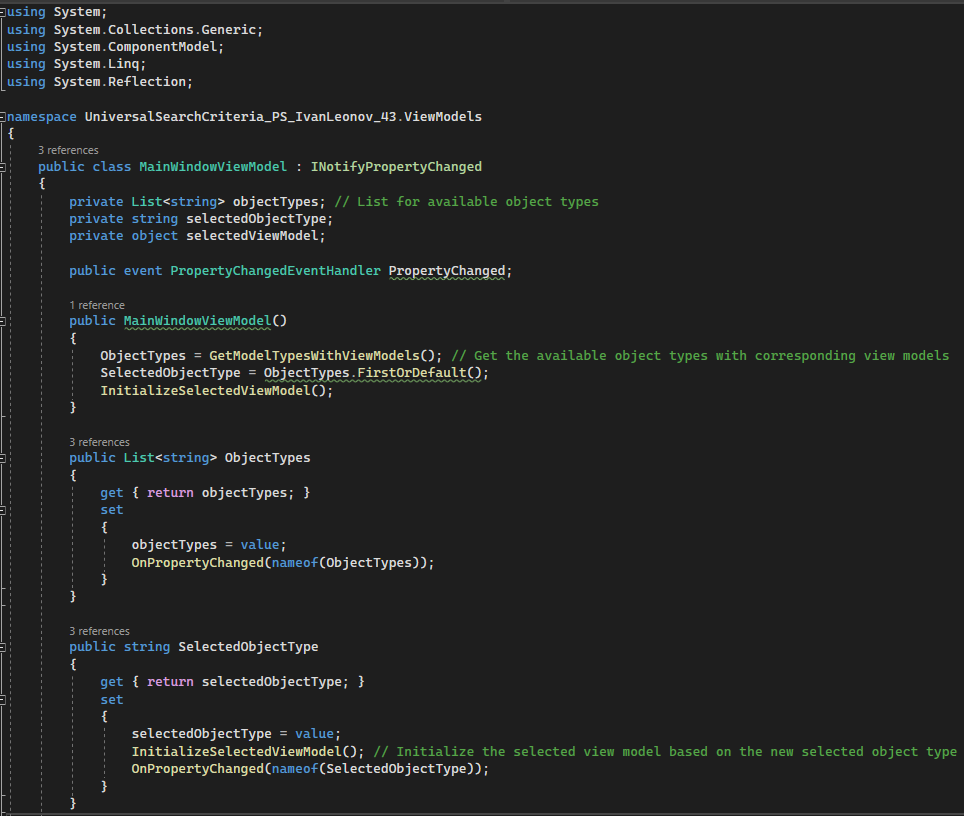
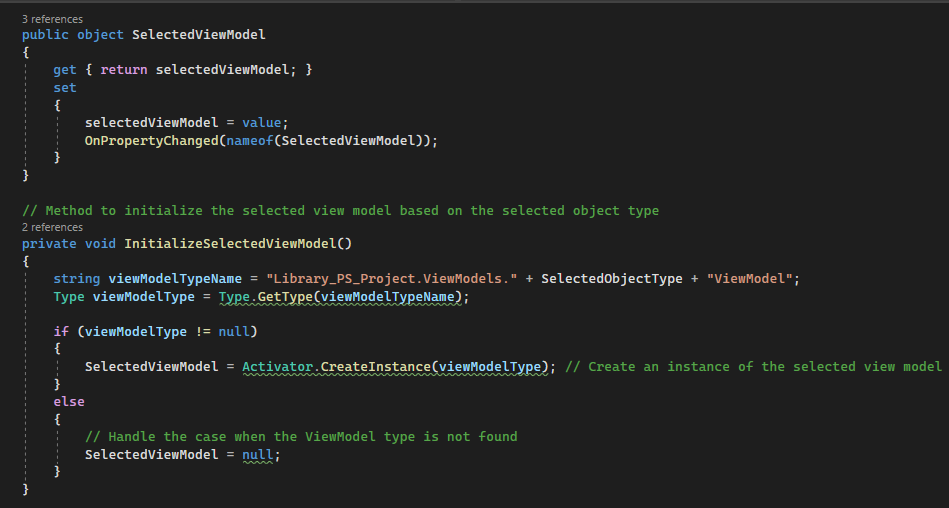
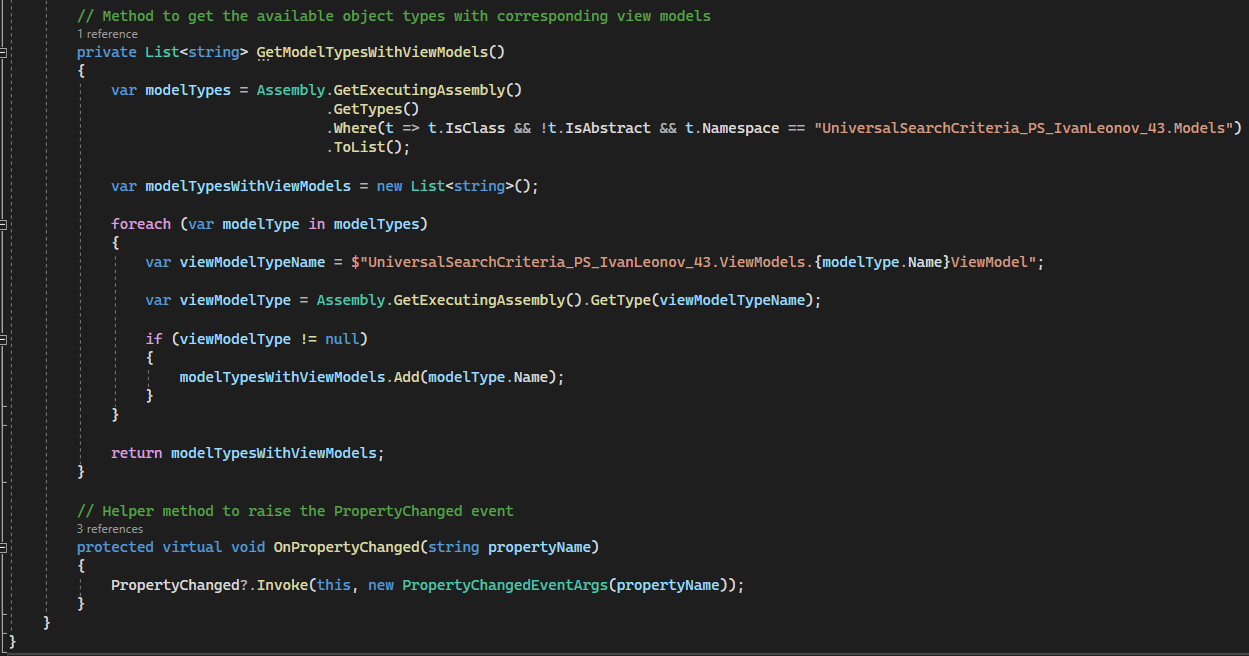
**3.2. MainWindowViewModel:**

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.

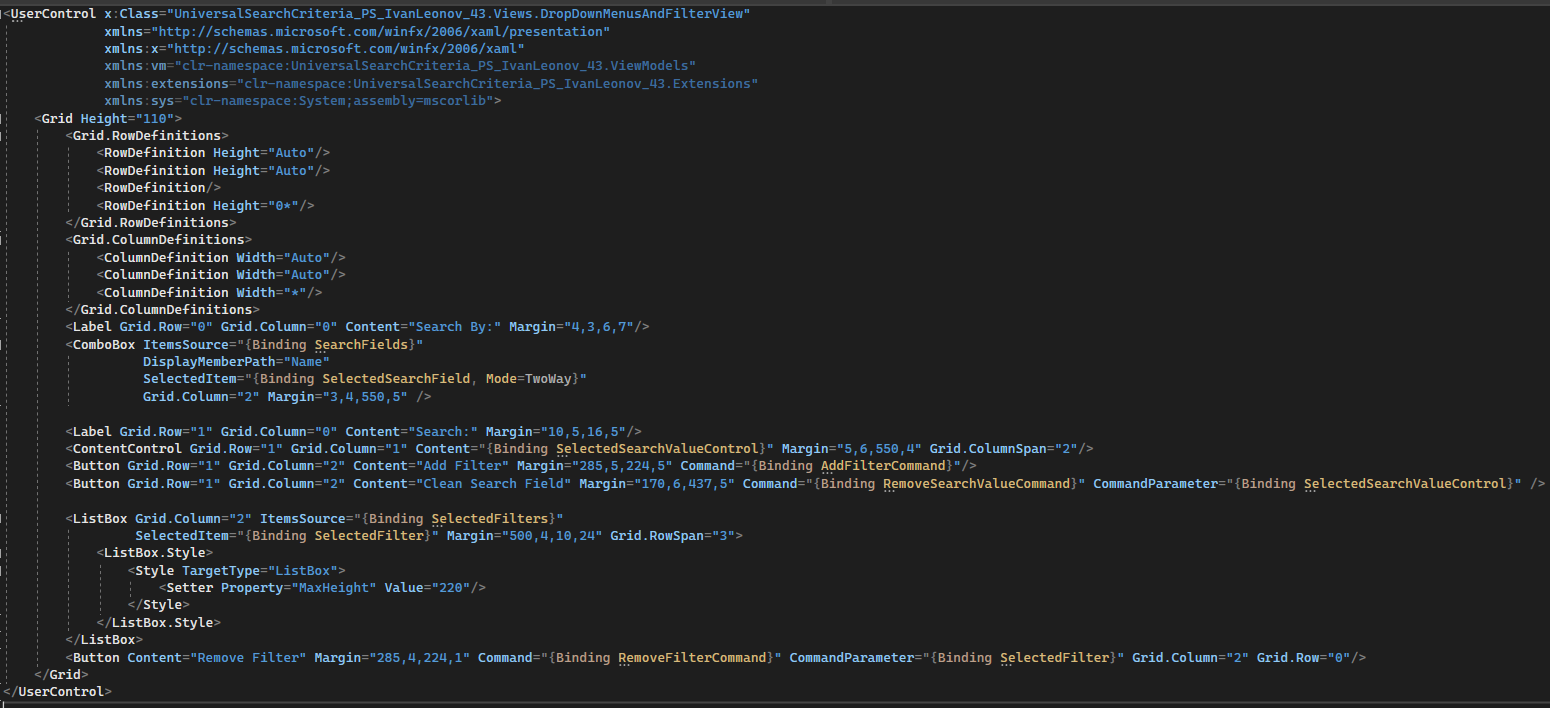
Методът GetModelTypesWithViewModels извлича наличните типове модели в проекта, които имат съответни модели и ViewModel-и. За всеки тип модел той конструира имената на свързаните типове модели на изгледи и проверява дали те съществуват. Ако моделите на изгледа(ViewModels) и модел (Models) съществуват, 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.

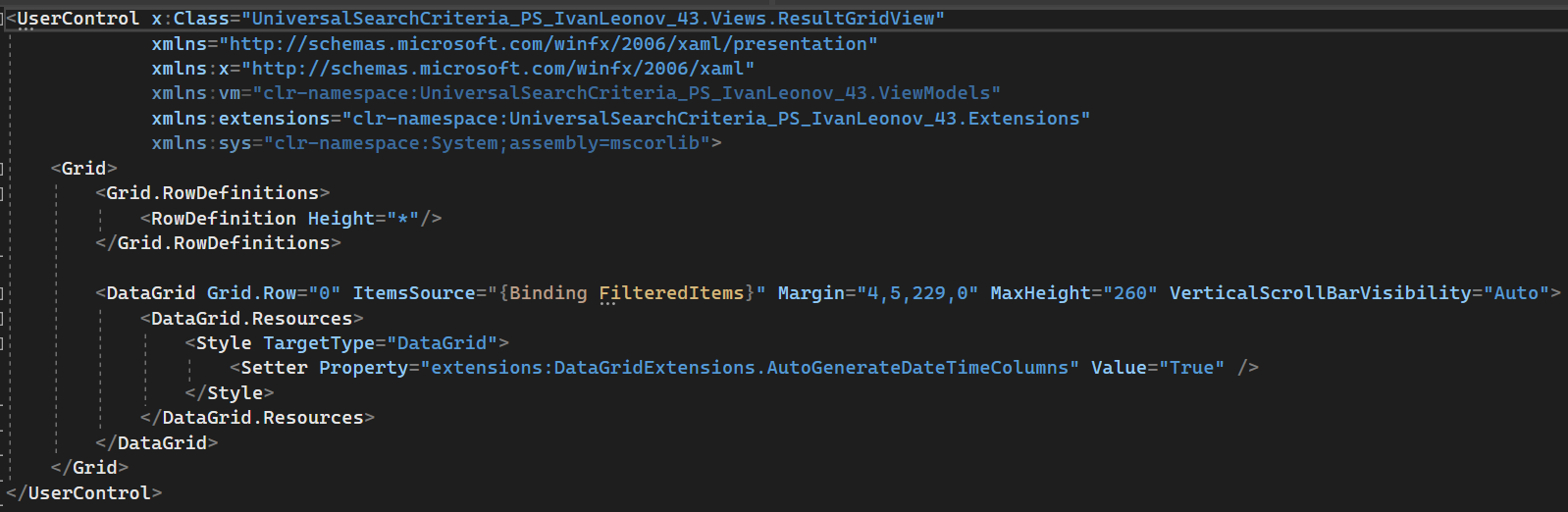
**3.3. DropDownMenusAndFilterView**

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. 

**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. 

**3.5. SearchViewModelBase<T>:** използва се катоабстрактен 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).

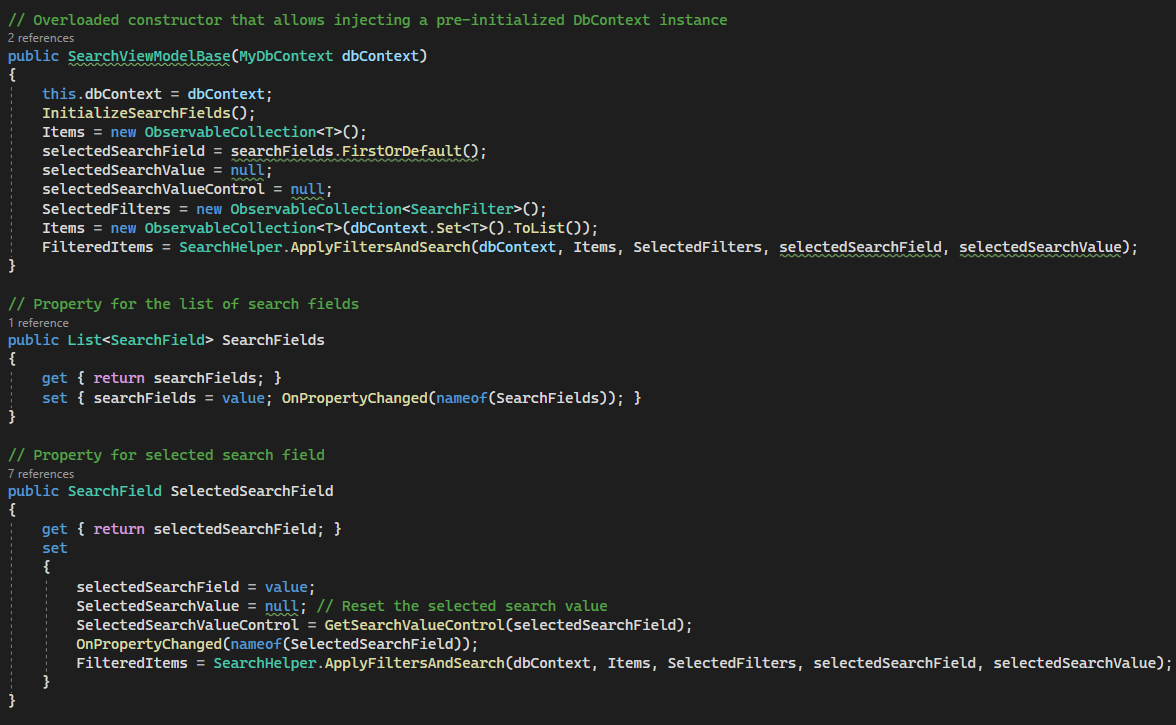
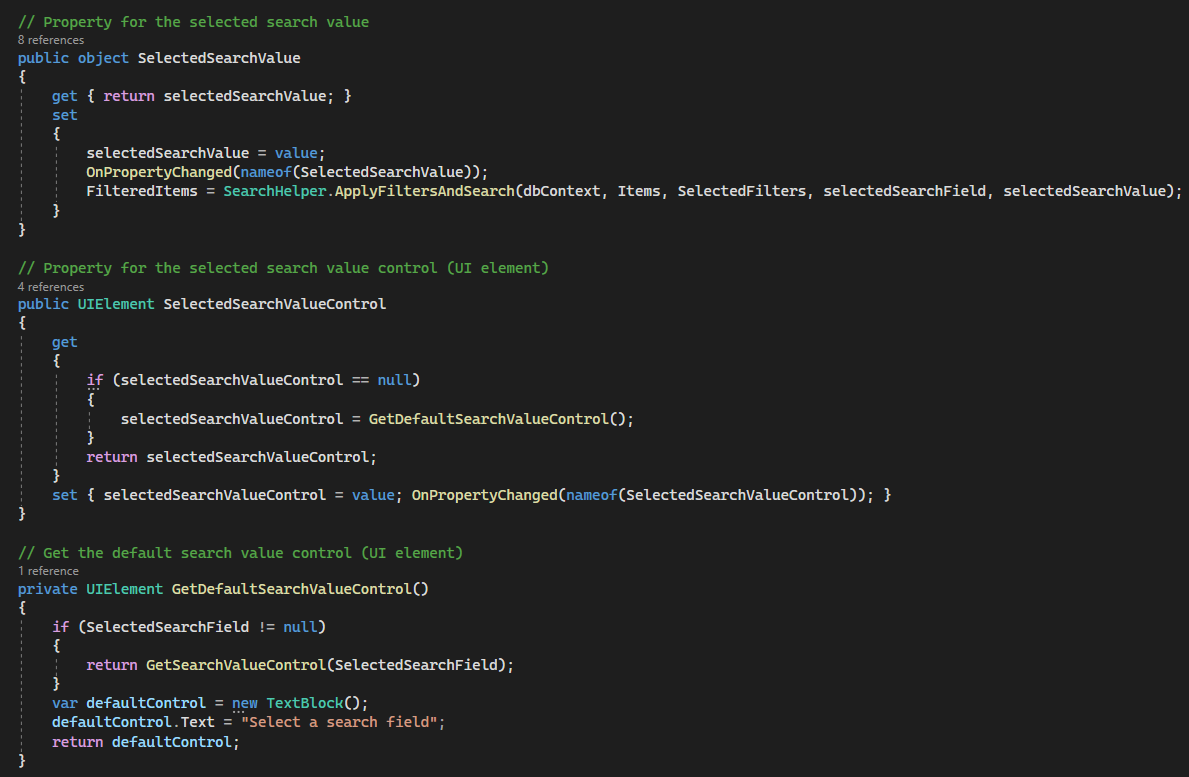
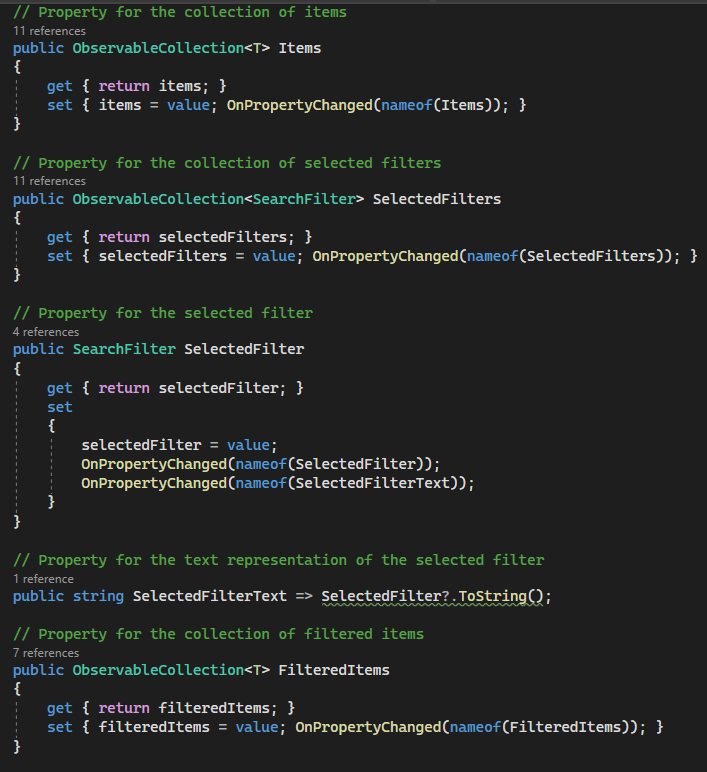
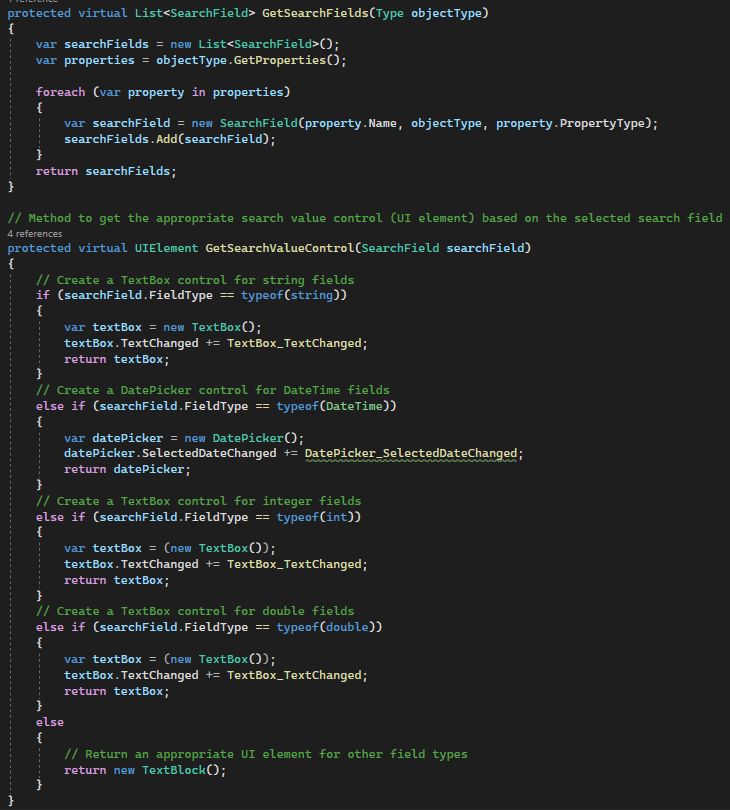
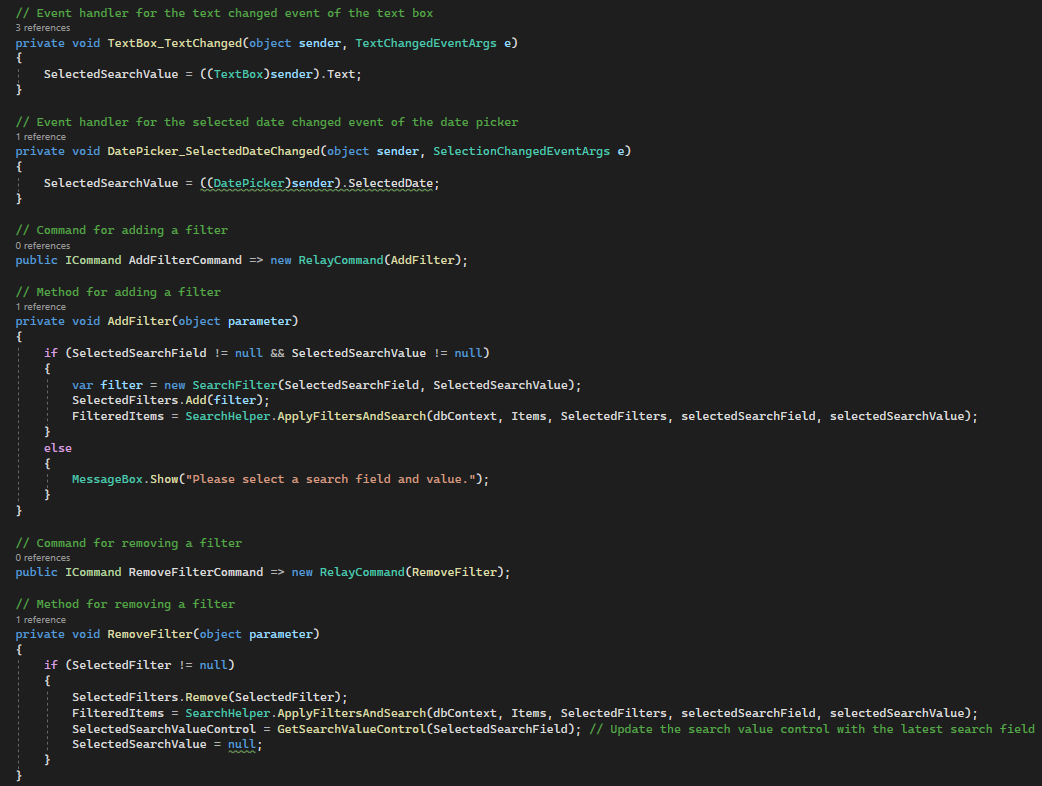
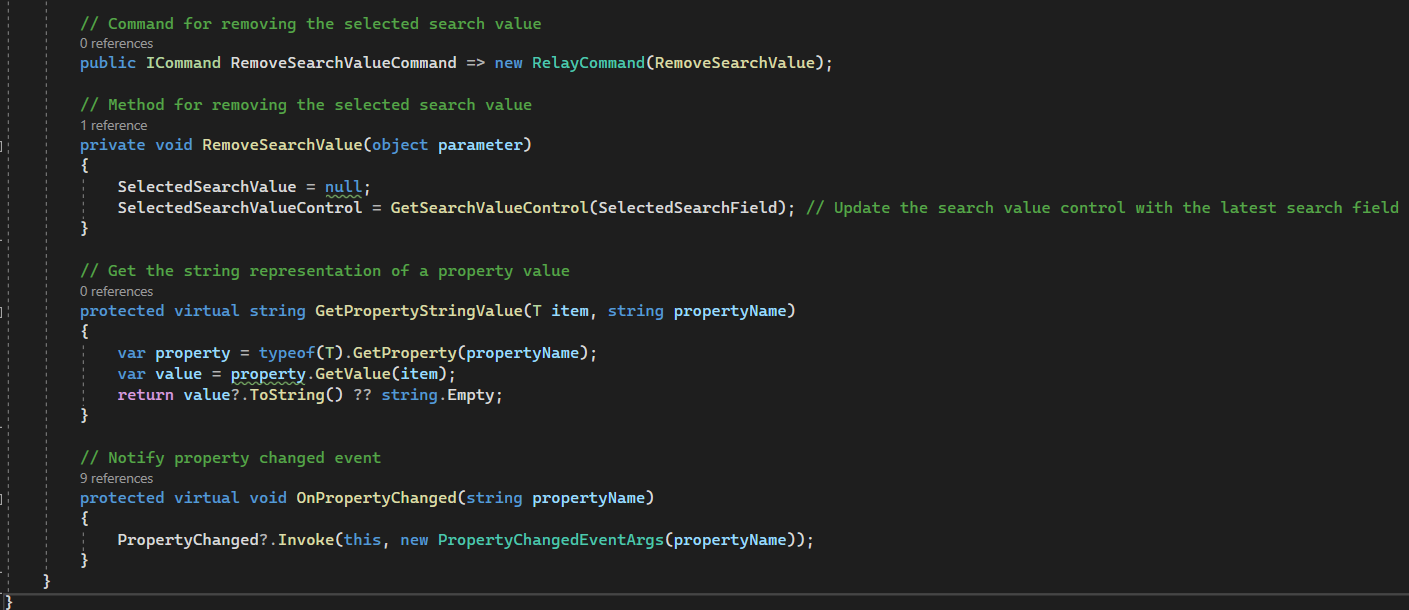
Той включва няколко полета и свойства, които се използват за управление на функционалността за търсене. Те включват полета за DbContext, полета за търсене, избрано поле за търсене, избрана стойност за търсене, контрол на избраната стойност за търсене (елемент на потребителския интерфейс), елементи, избрани 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.        

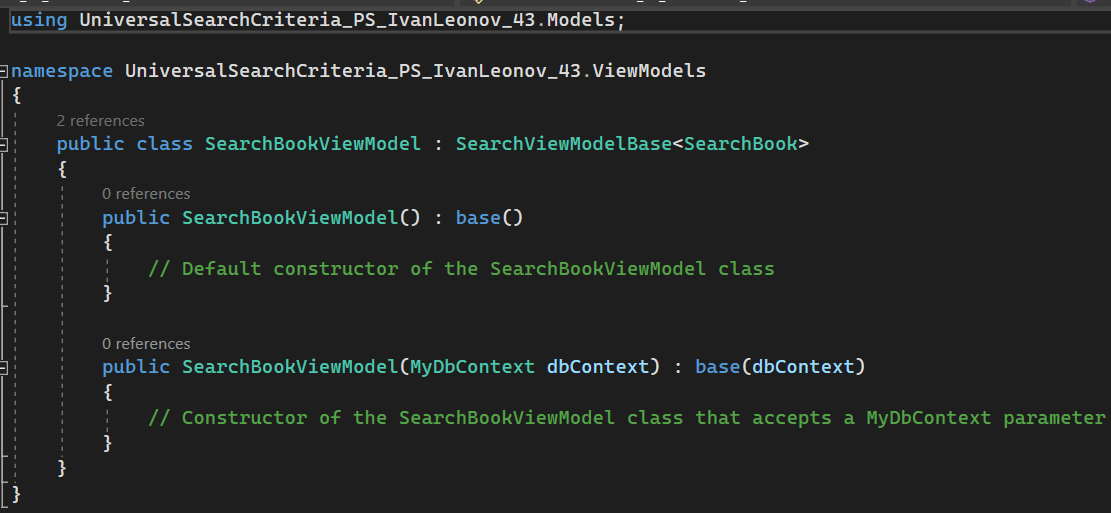
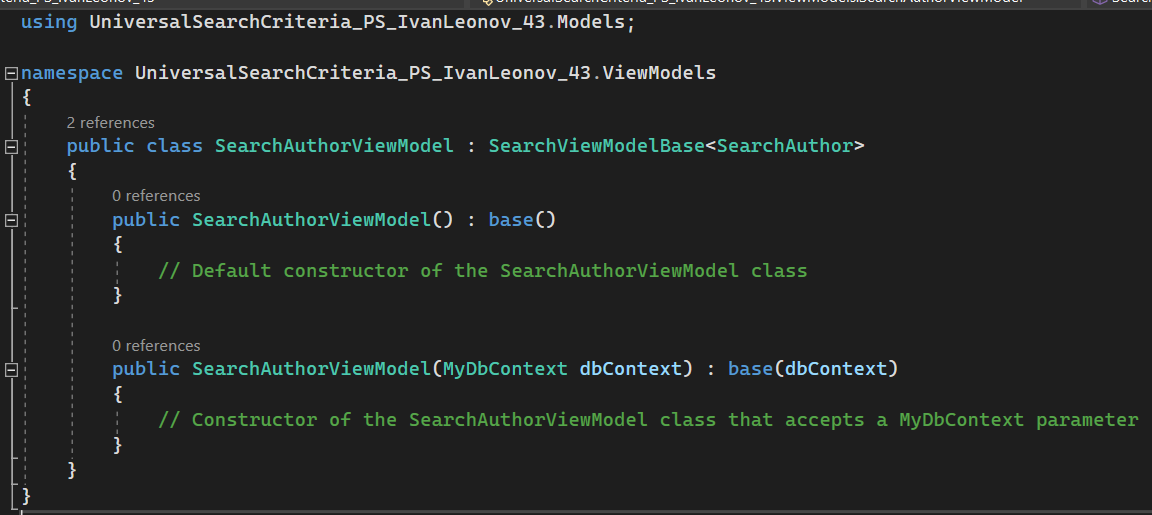
**3.6. SearchBookViewModel и SearchAuthorViewModel:**

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.

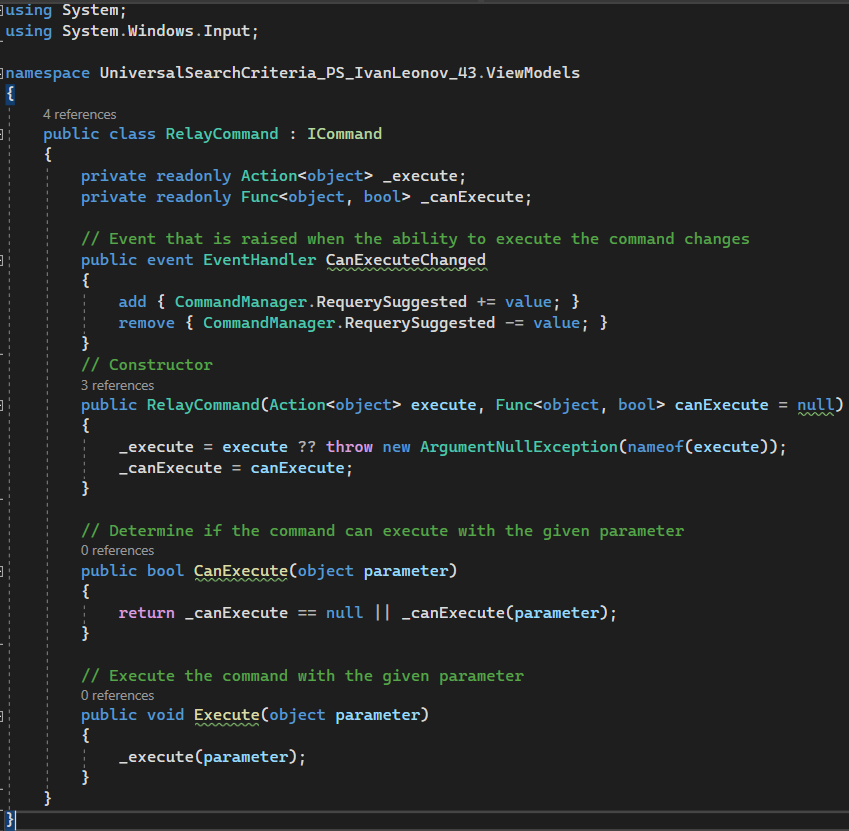
Съдържат два конструктора: конструктор по подразбиране и конструктор, който приема параметър MyDbContext. И двата конструктора извикват съответните конструктори на базовия клас, като използват ключовата дума base(), предавайки аргументите на конструктора на базовия клас. Конструкторът по подразбиране няма никаква допълнителна логика и просто извиква конструктора по подразбиране на базовия клас. Конструкторът, който приема параметър MyDbContext, позволява използването на предварително инициализирана инстанция на MyDbContext 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.



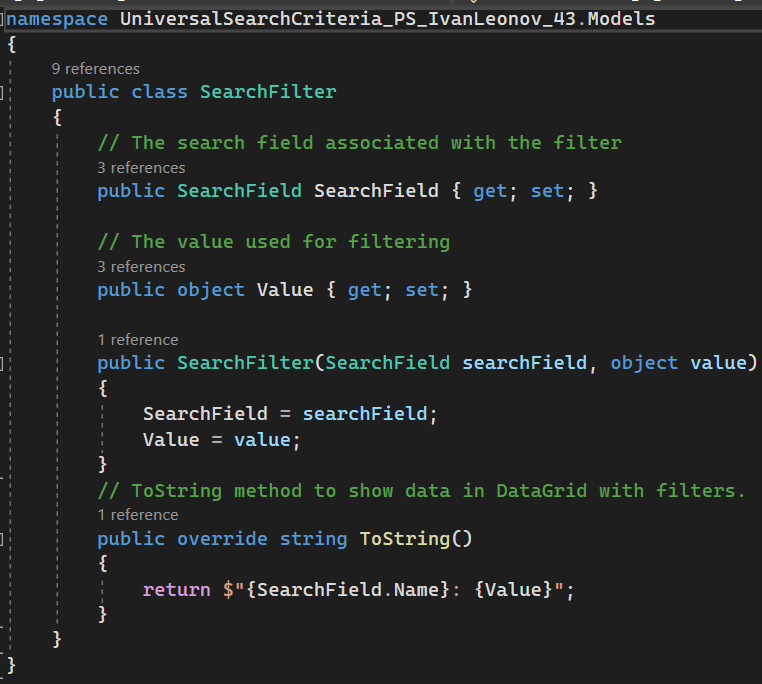
**3.7. RelayCommand**

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. 

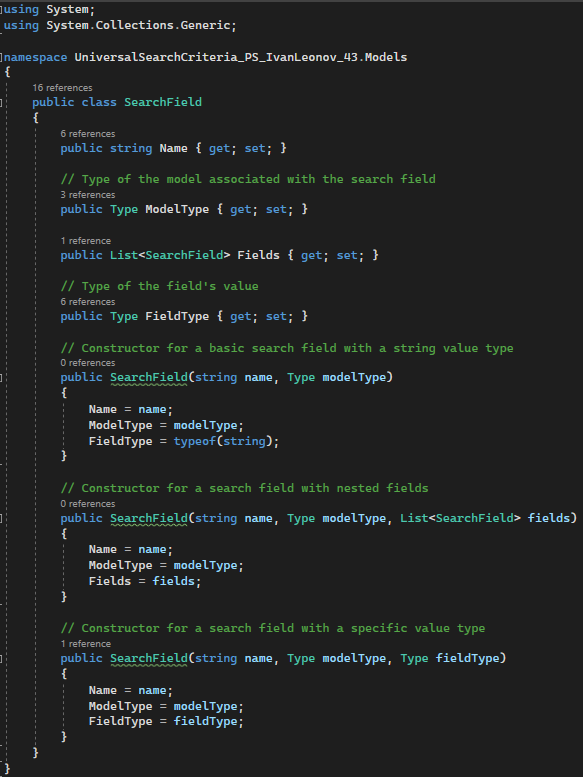
**3.8. SearchFilter**

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. 

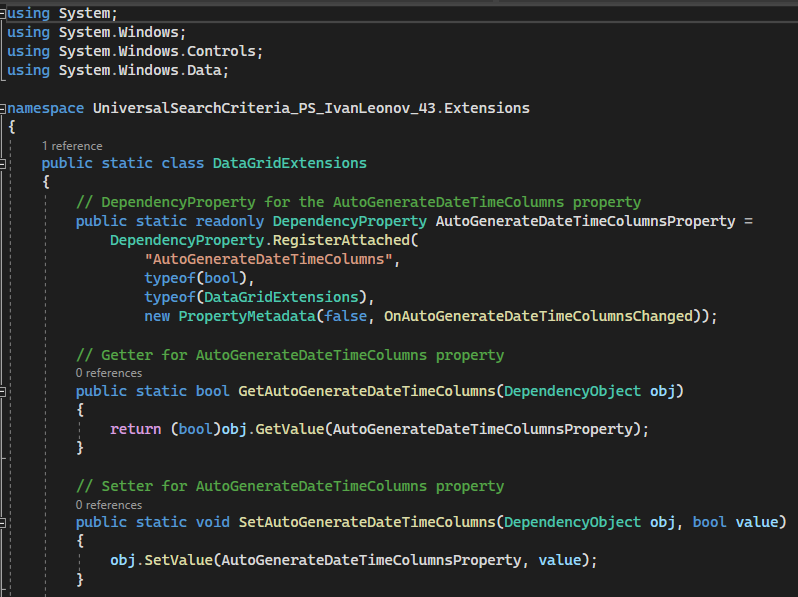
**3.9. SearchField**

Представлява 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. 

**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 DataGridTextColumn. Заглавието и обвързването на колоната се задават на съответните стойности от автоматично генерираната колона. Освен това към връзката се прилага StringFormat, за да се форматират стойностите на DateTime като "dd/MM/yyyy". Накрая автоматично генерираната колона се заменя с новосъздадената колона DataGridTextColumn, 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.

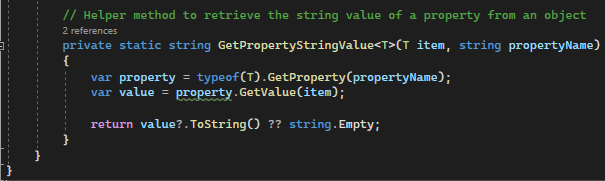
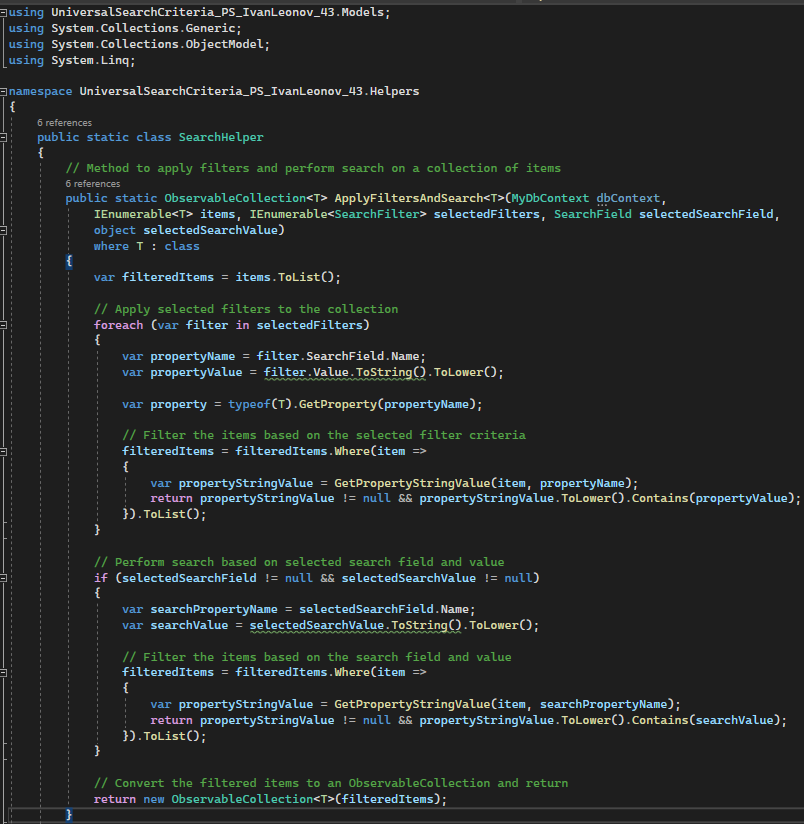


**3.11. Search Helper**

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.

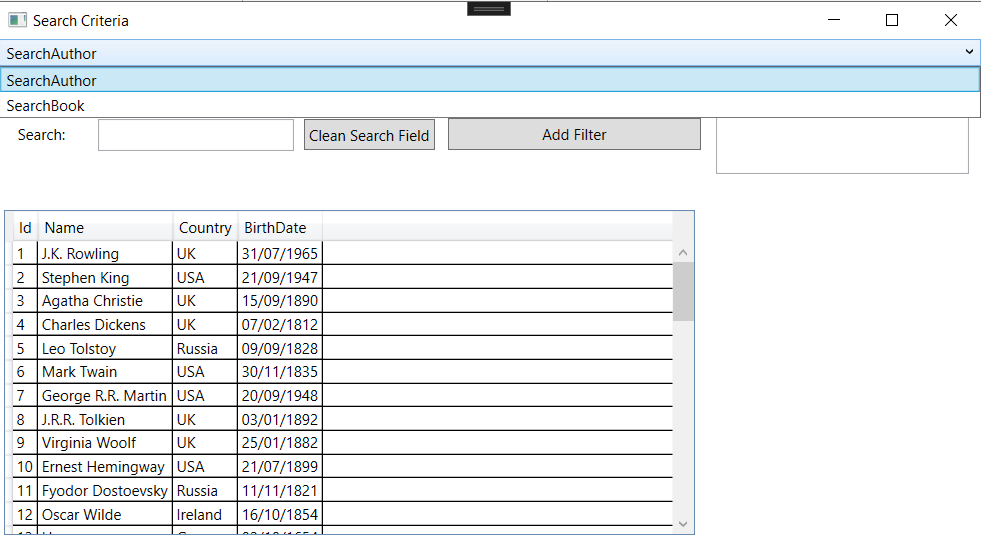


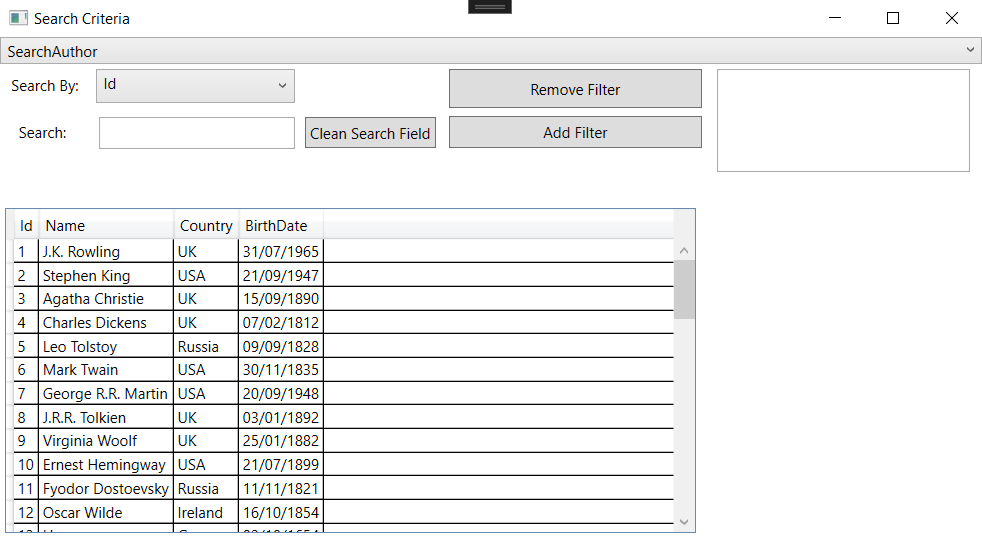
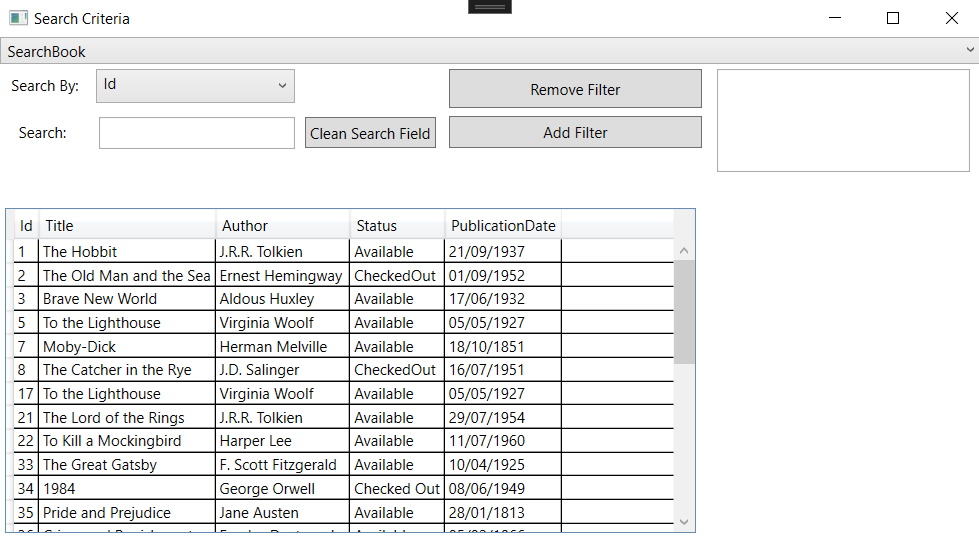
4. Ръководството за потребителя - така предложеното решение как се ползва през графичния интерфейс. Тук се демонстрират и функционалните изисквания от заданието

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

1.Стартирайте приложението, като стартирате изпълнимия файл.

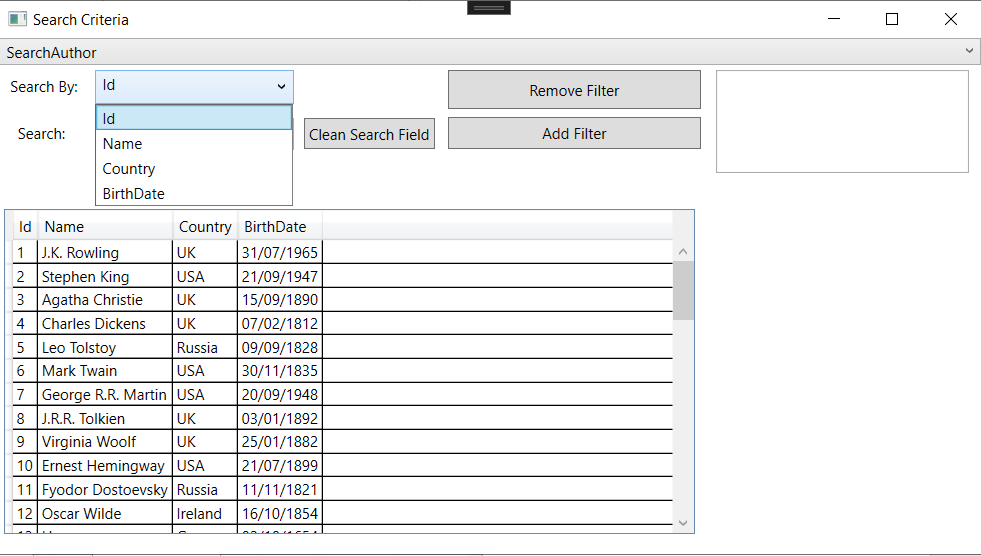
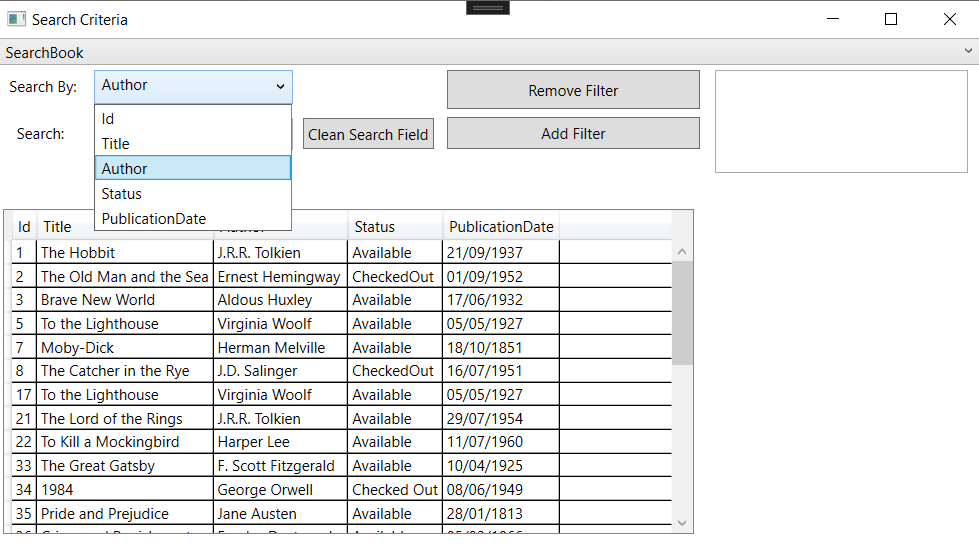
2.При отваряне на главния прозорец ще видите ComboBox (списък с модели), на който от падащото меню може да направите избор на типа на обекта (в случая книга или автор).



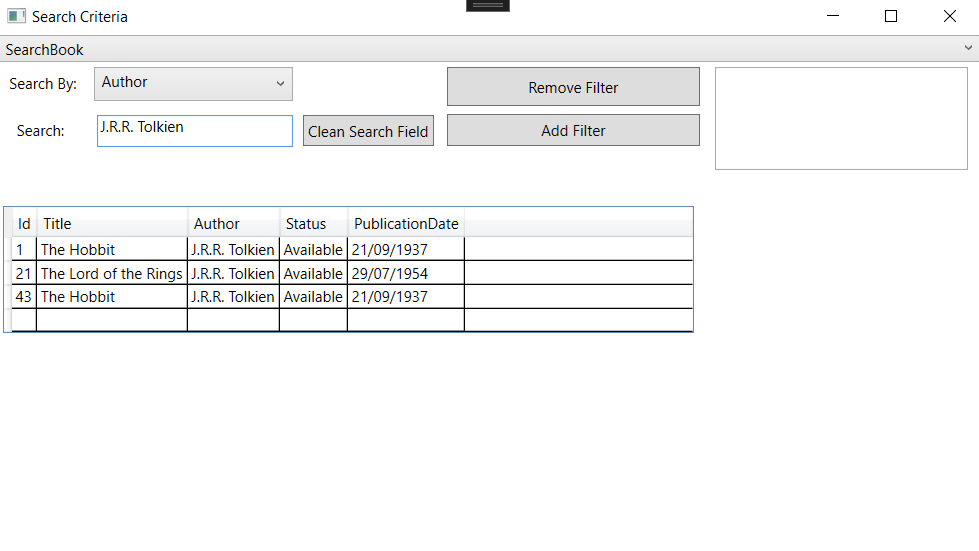
3.Изберете желания тип обект и съответният интерфейс за търсене ще се покаже по-долу. 

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 колони.

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.

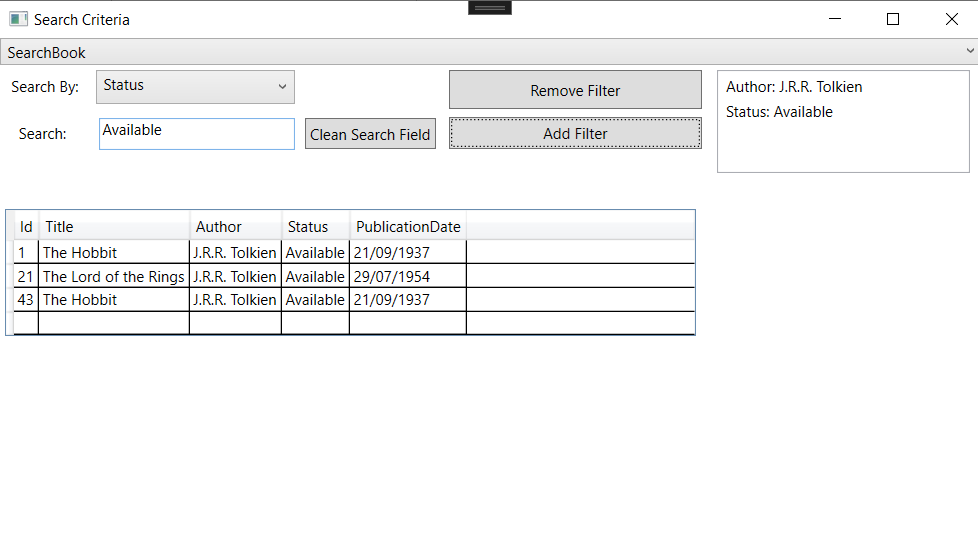
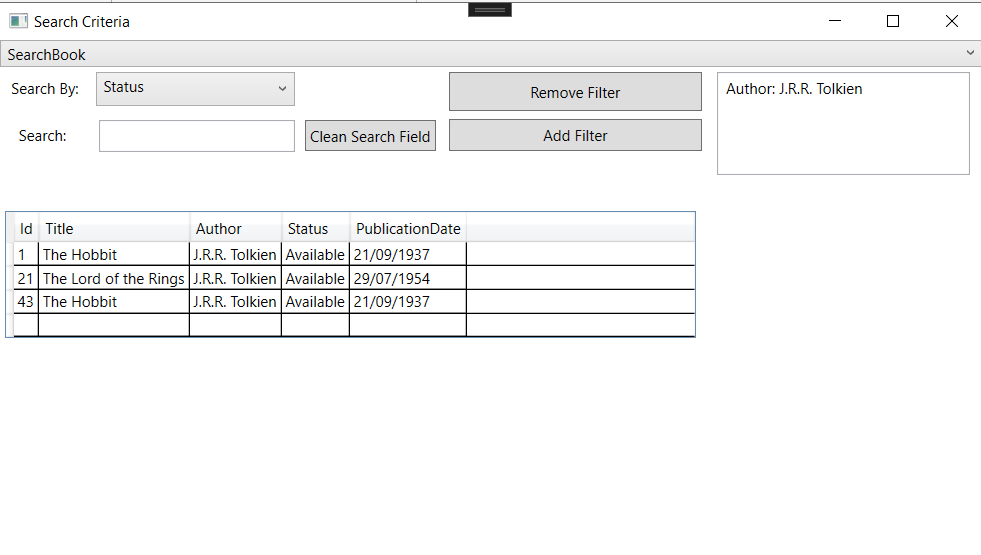


6. След като изберете даденото поле по което искате да филтрирате, ще виждате поле за търсене, в което може да въведете стойността, по която искате да търсите, като търсенето ще се случва в реално време. Real-time searching-а, както и прилагането на филтрите са направени non-case sensitive, което показва, 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.

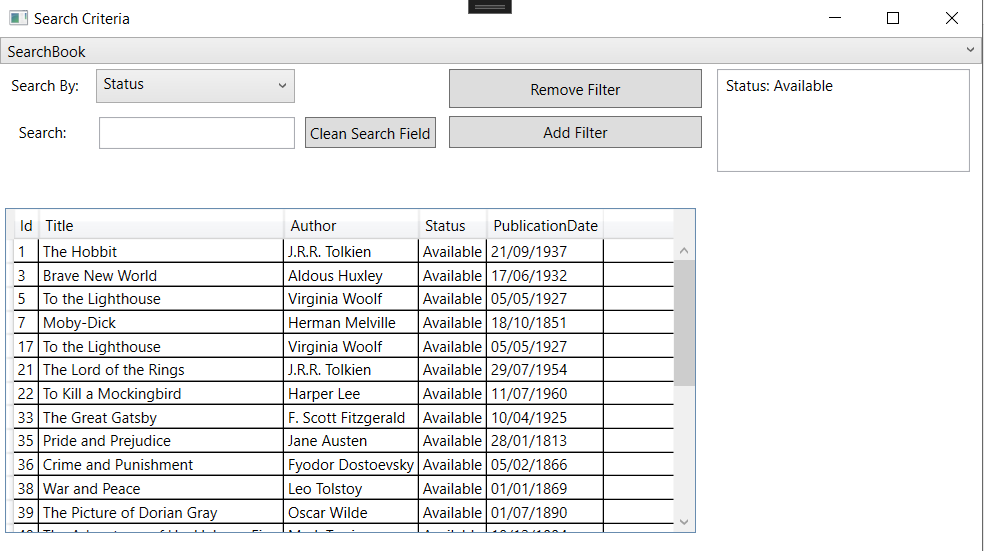


7. За да приложите филтъра, щракнете върху бутона "Add Filter" (Добавяне на филтър), като е възможно да добавяте няколко филтъра последователно по различни критерии.

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



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



5. Указания за преизползване/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.

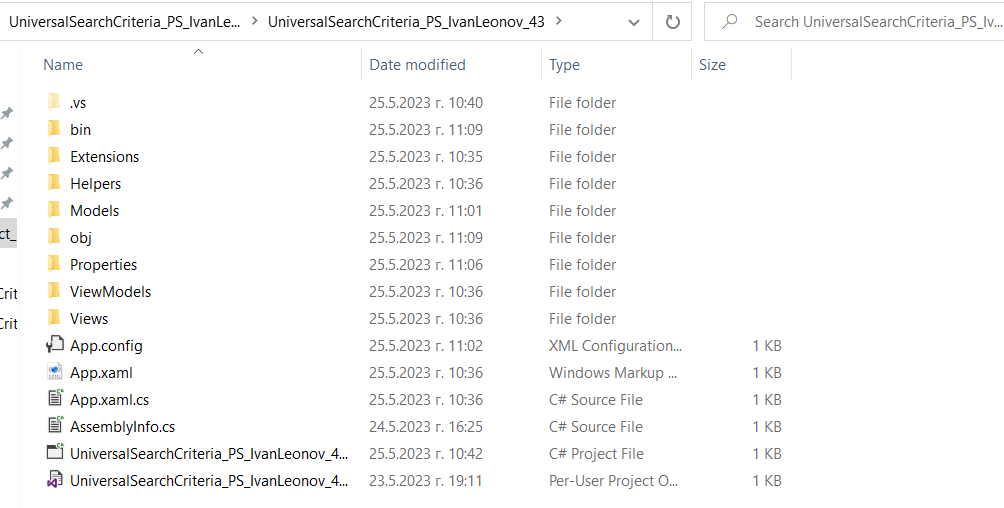
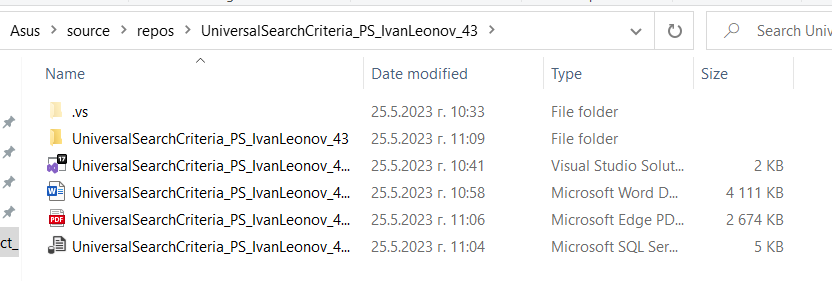
**Реализиране на графичния потребителски интерфейс:**

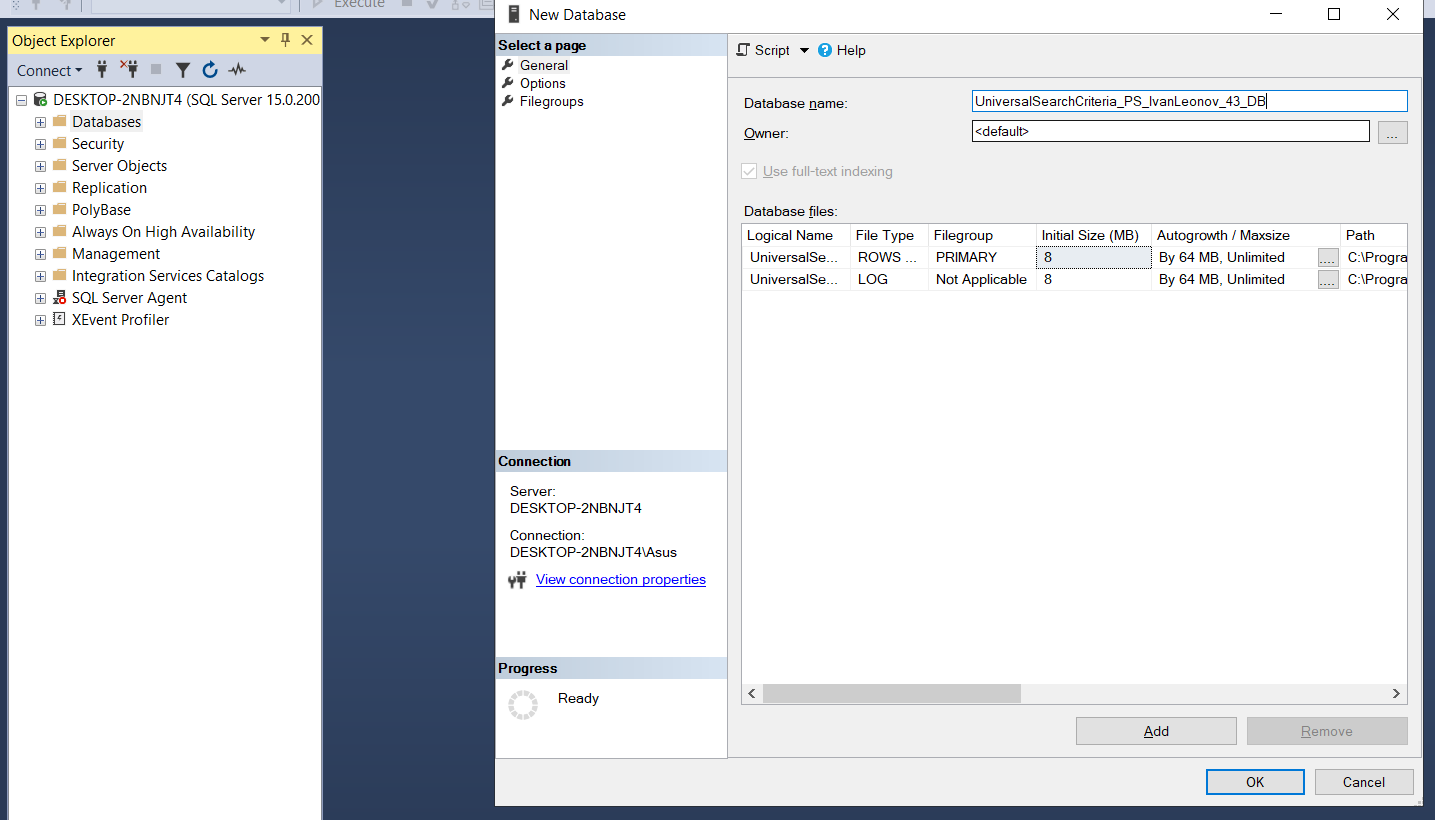
* Създайте изгледи в XAML (потребителски контроли, прозорци), съответстващи на създадените модели на изгледи.
* Проектирайте оформлението на графичния потребителски интерфейс, включително полетата за търсене, контролите за въвеждане, решетката с данни за показване на резултатите от търсенето и бутоните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

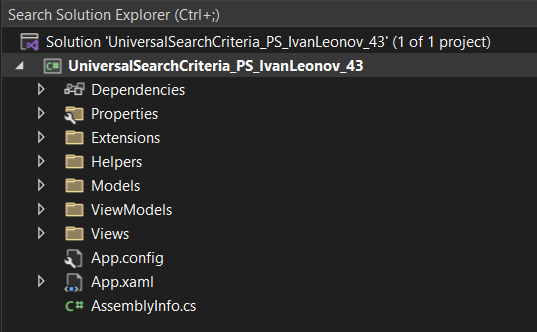
**Интегриране със системата за управление на библиотеката:**

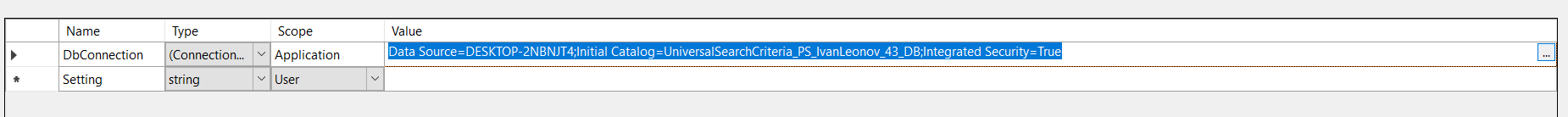
* Свържете се с подходящ източник на данни или база данни, като използвате Entity Framework или подобна рамка.
* Настройте връзката с базата данни, като промените connection string-а в настройките на проекта. (Properties/Settings.Settings).
* Може да използвате миграции, ако е необходимо, за да приложите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

***Демонстрация на свързването на проекта с Microsoft SQL Server Management:***

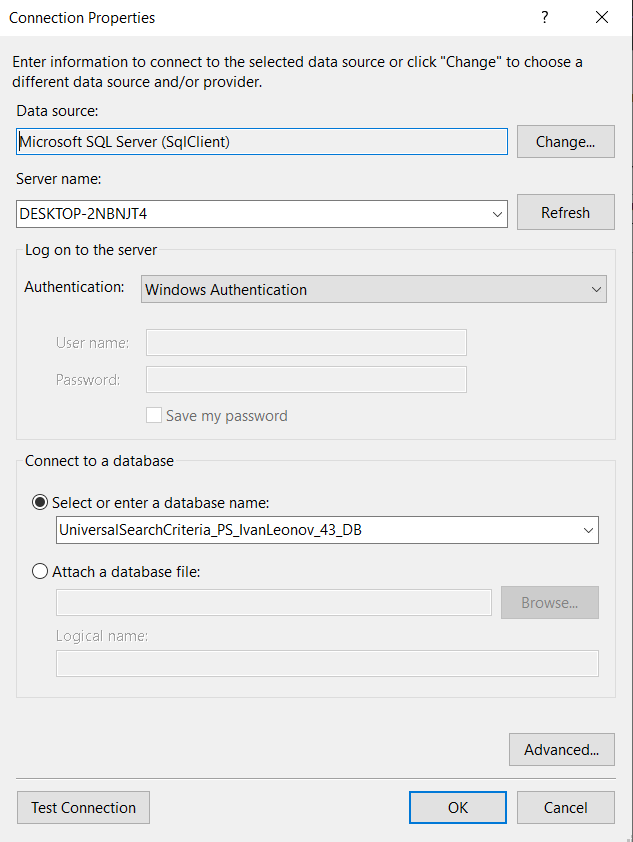
1. Изтегляме си проекта. Папката с проекта след изтеглянето: 

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:

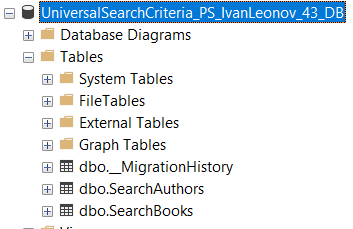
2. Сега отваряме проекта, като препоръчително е да отворите Solution File, след отварянето би трябвало да изглежда по този начин: 

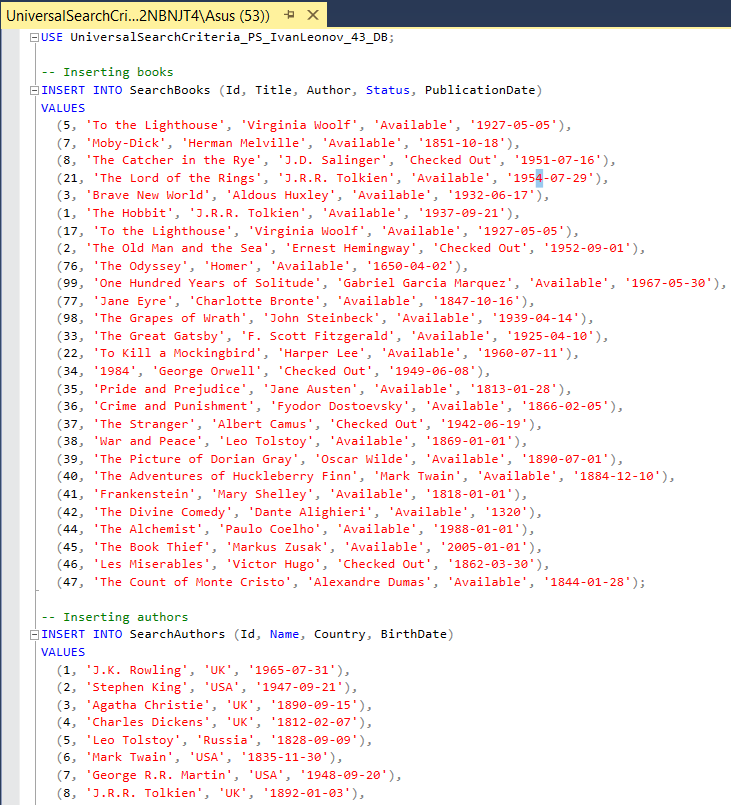
3. Отиваме в Properties/Settings.Settings и натискаме върху Value : 

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.



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: 

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 ни 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.