Технически университет - София

ДОКУМЕНТАЦИЯ

на проект

ПО ДИСЦИПЛИНА

ПРОГРАМНИ СИСТЕМИ

Тема: №2

Факултет: Компютърни системи и технологии
Специалност: Компютърно и софтуерно инженерство

Изготвил: Георги Александров Стефанов
Фак. номер: 121220015
Група: 43

Асистент: ас. Петър Маринов

София 2023 г.

Съдържание

Задание

Без използване на готова компилирана или лицензирана библиотека чрез използване или надграждане на включените в Windows Forms и WPF класове да се изгради:

Система за извеждане на контроли за потребителя, в които да се въвеждат критерии за търсене.

Изисквания:

- Трябва да може да се окаже за кои полета/свойства/ на обект ще бъдат въведени критерии за търсене.
- За всяко от посочените полета/свойства/, според типа на полето/свойството/ се визуализира подходяща контрола за въвеждане на стойност.
- За всяка от визуализираните контроли се визуализира и подходящо описание на контролата.
- След въвеждане на стойностите (в съответните контроли) те да послужат за формиране на критерий, по който да се намерят (например от БД) само обекти, отговарящи на този критерий.

Приложение за демонстрация:

- Системата трябва да се използва в .Net C# WPF приложение с връзка с БД чрез EF и с MVVM структура. (Приложението не е нужно да е богато реализирано за целите на заданието.)
- Обектите за визуализация и търсене по критерии идват от БД.
- Оказване на полетата/свойствата/, за които ще се въвеждат критерии, трябва да е по модулен начин, необвързан към конкретен тип обекти (но може да наследяват общ клас или интерфейс).
- Пространството за визуализиране на контролите за критерии да може да се указва за всеки отделен интерфейс.
- Да се демонстрира в рамките на поне 2 различни интерфейса (с различен view-model) указване на полета/свойства/ на обекти от 2 различни типа.
- Да се демонстрира посочване на различни полета/свойства/ за един и същи тип обекти.

Примери за използване (Application scenario):

1. Система за управление на Библиотека. Екран за търсене на книга. Съставящият интерфейс може да определи кои полета да са видими за потребителя, за да търси по
тях.

Въведение в приложението

Проектът представлява WPF приложение, което използва MVVM шаблона и реализира връзка с база данни посредством Entity Framework. Само по себе си предоставя на потребителите услуга за филтриране на обекти на базата на техните полета (критерии). Наличните обекти са: филми (Movies) и коли (Cars), като изборът между тях и самите им параметри става по динамичен начин в основния прозорец. При въвеждане на стойности за избраните параметри и натискане на бутона за филтриране, в нов прозорец в табличен вид се визуализират обектите, отговарящите на зададените критерии.

Спецификация на базата данни

На следната фигура е представена Entity Relationship (ER) диаграмата на базата данни, която показва наличните таблици в базата данни. Всяка една от таблиците представлява списък от обекти, които могат да бъдат подложени на филтриране от системата.

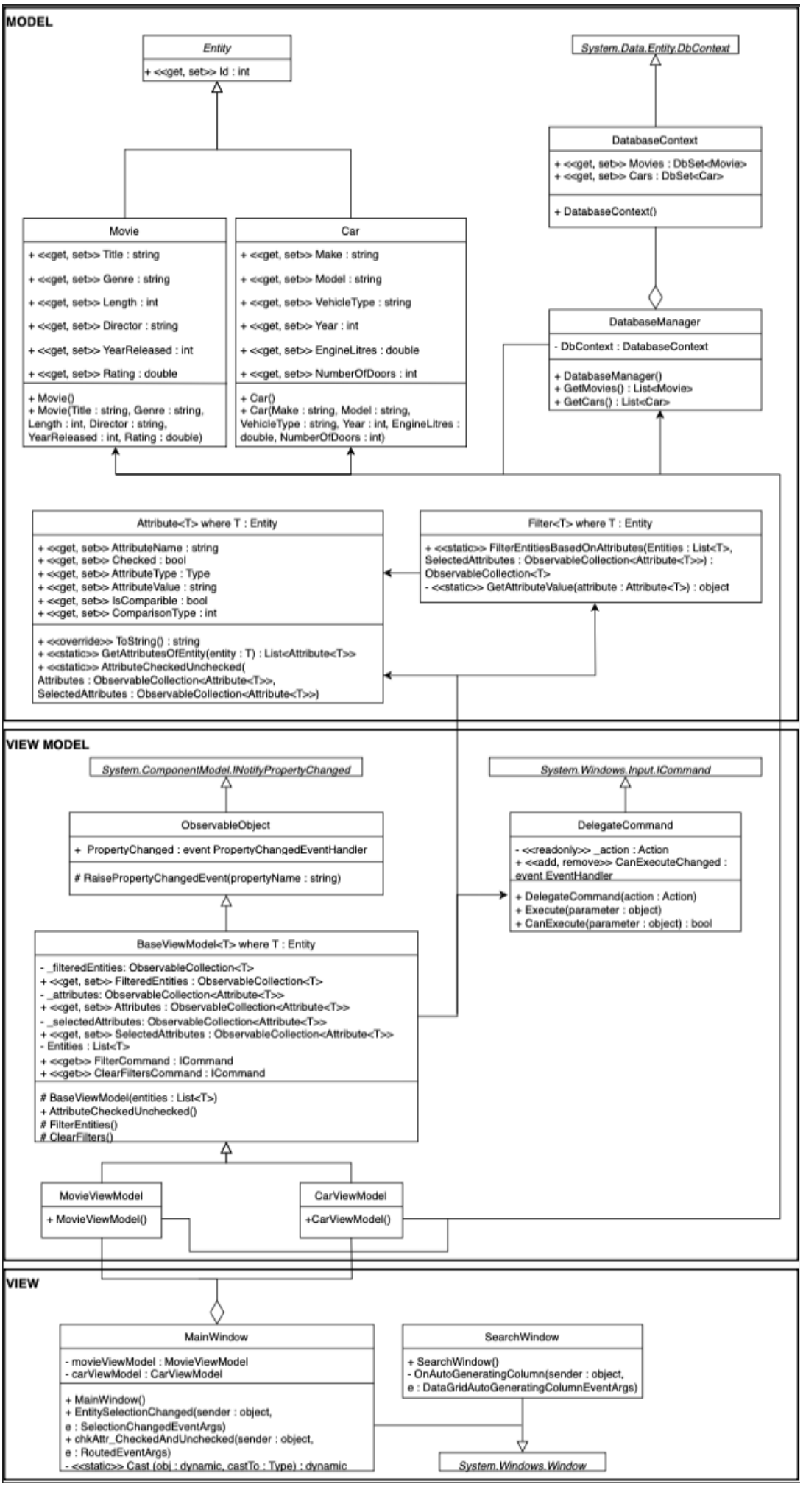
Movies	
PK	**id int NOT NULL**
	Title nvarchar(max)
	Genre nvarchar(max)
	Length int NOT NULL
	Director nvarchar(max)
	YearReleased int NOT NULL
	Rating float NOT NULL

Cars	
PK	**id int NOT NULL**
	Make nvarchar(max)
	Model nvarchar(max)
	VehicleType nvarchar(max)
	Year int NOT NULL
	EngineLitres float NOT NULL
	NumberOfDoors text NOT NULL

Entity Relationship диаграма

Спецификация на класовете

На следващата страница е представена клас диаграмата (UML Class Diagram), която визуализира класовете, изграждащи приложението, техните полета, методи, както и връзките между различните класове.


MODEL
Entity
+ <<get, set>> Id : int
System.Data.Entity.DbContext
DatabaseContext
+ <<get, set>> Movies : DbSet<Movie>
+ <<get, set>> Cars : DbSet<Car>
+ DatabaseContext()
Movie
+ <<get, set>> Title : string
+ <<get, set>> Genre : string
+ <<get, set>> Length : int
+ <<get, set>> Director : string
+ <<get, set>> YearReleased : int
+ <<get, set>> Rating : double
+ Movie()
+ Movie(Title : string, Genre : string, Length : int, Director : string, YearReleased : int, Rating : double)
Car
+ <<get, set>> Make : string
+ <<get, set>> Model : string
+ <<get, set>> VehicleType : string
+ <<get, set>> Year : int
+ <<get, set>> EngineLitres : double
+ <<get, set>> NumberOfDoors : int
+ Car()
+ Car(Make : string, Model : string, VehicleType : string, Year : int, EngineLitres : double, NumberOfDoors : int)
DatabaseManager
- DbContext : DatabaseContext
+ DatabaseManager()
+ GetMovies() : List<Movie>
+ GetCars() : List<Car>
Attribute<T> where T : Entity
+ <<get, set>> AttributeName : string
+ <<get, set>> Checked : bool
+ <<get, set>> AttributeType : Type
+ <<get, set>> AttributeValue : string
+ <<get, set>> IsComparible : bool
+ <<get, set>> ComparisonType : int
+ <<override>> ToString() : string
+ <<static>> GetAttributesOfEntity(entity : T) : List<Attribute<T>>
+ <<static>> AttributeCheckedUnchecked(Attributes : ObservableCollection<Attribute<T>>, SelectedAttributes : ObservableCollection<Attribute<T>>)
Filter<T> where T : Entity
+ <<static>> FilterEntitiesBasedOnAttributes(Entities : List<T>, SelectedAttributes : ObservableCollection<Attribute<T>>) : ObservableCollection<T>
- <<static>> GetAttributeValue(attribute : Attribute<T>) : object
VIEW MODEL
System.ComponentModel.INotifyPropertyChanged
ObservableObject
+ PropertyChanged : event PropertyChangedEventHandler
# RaisePropertyChangedEvent(propertyName : string)
System.Windows.Input.ICommand
DelegateCommand
- <<readonly>> _action : Action
+ <<add, remove>> CanExecuteChanged : event EventHandler
+ DelegateCommand(action : Action)
+ Execute(parameter : object)
+ CanExecute(parameter : object) : bool
BaseViewModel<T> where T : Entity
- _filteredEntities: ObservableCollection<T>
+ <<get, set>> FilteredEntities : ObservableCollection<T>
- _attributes: ObservableCollection<Attribute<T>>
+ <<get, set>> Attributes : ObservableCollection<Attribute<T>>
- _selectedAttributes: ObservableCollection<Attribute<T>>
+ <<get, set>> SelectedAttributes : ObservableCollection<Attribute<T>>
- Entities : List<T>
+ <<get>> FilterCommand : ICommand
+ <<get>> ClearFiltersCommand : ICommand
# BaseViewModel(entities : List<T>)
+ AttributeCheckedUnchecked()
# FilterEntities()
# ClearFilters()
MovieViewModel
+ MovieViewModel()
CarViewModel
+CarViewModel()
VIEW
MainWindow
- movieViewModel : MovieViewModel
- carViewModel : CarViewModel
+ MainWindow()
+ EntitySelectionChanged(sender : object, e : SelectionChangedEventArgs)
+ chkAttr_CheckedAndUnchecked(sender : object, e : RoutedEventArgs)
- <<static>> Cast (obi : dynamic, castTo : Type) : dynamic
SearchWindow
+ SearchWindow()
- OnAutoGeneratingColumn(sender : object, e : DataGridAutoGeneratingColumnEventArgs)
System.Windows.Window

Model

Това е основният модел на приложението, който включва в себе си модела на данните, както и бизнес и валидационната логика. Класовете тук са следните:

- Entity - основен клас за обозначаване на обекти като таблици в базата данни. Всички обекти за филтриране трябва да наследяват този клас. Съдържа уникалния идентификатор (Id) на съответния обект в базата;
- Movie - модел на филм, съдържащ необходимата информация. Полетата на един филм са следните и те са представени като свойства (properties):
  - Title : string - заглавие на филма;
  - Genre : string - жанр на филма;
  - Length : string - дължина на филма в минути;
  - Director : string - режисьор на филма;
  - YearReleased : int - година на излизане на филма;
  - Rating : double - рейтинг на филма, взет спрямо IMDb.
- Car - модел на кола, съдържащ необходимата информация. Полетата на една кола са следните и те са представени като свойства (properties):
  - Make : string - марка на автомобила;
  - Model : string - модел на автомобила;
  - VehicleType : string - вид на автомобила (напр. SUV, Sedan, Hatchback)
  - Year : int - година на производство на автомобила;
  - EngineLitres : double - големина на двигателя на автомобила в литри;
  - NumberOfDoors : int - брой врати на автомобила според каталожна информация.
- DatabaseContext - контекст на базата данни, който наследява от System.Data.Entity.DbContext. Съдържа свойства от тип DbSet за всеки един от обектите в базата данни. Връзката към базата данни посредством Connection String се извършва в конструктора на класа.
- DatabaseManager - служи за работа с базата данни и обектите в нея, използвайки контекста на базата данни. Понастоящем съдържа два публични метода:
  - public List<Movie> GetMovies() - връща всички налични филми в базата данни под формата на Collection;
  - public List<Car> GetCars() - връща всички налични коли в базата данни под формата на Collection;
- Attribute<T> - генеричен (generic) клас, който представлява атрибут (поле) на обект, подлежащ на филтриране. Въведено е ограничение на типа, от който може да бъде инстанциран генеричният клас и то е само за “Entity” обекти (т.е. всички наследници на Entity класа). Обектите от този клас служат за представяне и работа с всеки един от атрибутите на избрания за филтриране обект, като предоставя следните свойства:
  - AttributeName : string - името на атрибута (полета), напр. Make на класа Car;

- ➢ Checked : bool - булева променлива, която отбелязва дали даденото поле е избрано като критерии за въвеждане на стойност. Чрез Binding това поле се свързва с тикче;
- ➢ AttributeType : Type - тип на полето, което се взима чрез Reflection за всяко едно от полетата на обекта, който ще се филтрира;
- ➢ AttributeValue : String - въведената стойност за атрибута при неговото избиране. Тъй като полето, към което е свързано в XAML частта е TextBox, стойността се взима под формата на низ, който с помощта на AttributeType динамично се каства към правилния тип;
- ➢ IsComparible : bool - булева променлива, която указва дали даденото поле може да бъде сравнявано (т.е. по-голямо и по-малко) с въведената стойност. Валидно само при полета от числов тип, но не и за низове.
- ➢ ComparisonType : int - идентификатор на вида сравнение между полето и зададената стойност. Валидни са следните опции:
  - 0 -> по-малко от (само числови типове);
  - 1 -> равно на (за всички типове, стойност по подразбиране);
  - 2 -> по-голямо от (само числови типове);
  - 3 -> съдържа се в (само за низове).

Методите на класа Attribute са следните:

- ➢ public override string ToString() - презаписване на базовия ToString() метод с цел връщане на името на атрибута, когато обект от тип Attribute се визуализира;
- ➢ public static List<Attribute<T>> GetAttributesOfEntity(T entity) - статичен метод, който приема генеричен обект за филтриране и като резултат връща колекция от атрибутите му, представени под формата на обекти от клас Attribute. Полетата на Attribute обектите се попълват на базата на полетата на подадения обект чрез изциклянето им едно по едно. Checked по подразбиране се поставя на “false”, AttributeValue на празен низ (тъй като ще се въвежда от потребителя), а ComparisonType на 1 (равно на);
- ➢ public static void AttributeCheckedUnchecked(...) - статичен метод, който получава колекция от атрибути, както и колекция от избрани за филтриране атрибути. За всеки един атрибут се проверява дали е бил избран от потребителя (т.е. дали Checked == true), при което се добавя в колекцията за избрани атрибути. Тъй като и двете колекции се подават по референция, това позволява динамично опресняване на екрана при добавяне или премахване на дадено поле за филтриране.

❖ Filter<T> - генеричен (generic) клас, който предоставя функционалност за филтриране на обекти. Въведено е ограничение на типа, от който може да бъде инстанциран генеричният клас и то е само за “Entity” обекти (т.е. всички наследници на Entity класа). Класът предоставя само един публичен метод, който може да бъде извикван за филтриране на избрания тип обекти:

- public static ObservableCollection<T> FilterEntitiesBasedOnAttributes(...) - филтрира подадената колекция от обекти на базата на подадените атрибути. Първоначално се извършва проверка дали всички подадени атрибути за филтриране имат въведена стойност. Ако това не е така, методът повдига ArgumentException, който дава информация кои полета трябва да бъдат въведени. Ако всички полета са въведени, се преминава към филтрирането на колекцията от обекти. За всеки един обект се проверява дали изпълнява зададените критерии, използвайки стойността и вида на сравнение на всеки един от избраните атрибути. Ако обектът отговаря на критериите, се добавя в колекцията, която се връща като резултат на функцията;
- private static object GetAttributeValue(Attribute<T> attribute) - вътрешен метод, който връща стойността на дадения атрибут под формата на променлива от обект с правилния тип. За целта се използва TypeDescriptor, който конвертира стойността на атрибута от низ (идващ от TextBox полето) в правилния тип на базата на AttributeType (Type).

ViewModel

ViewModel е средният слой в архитектурата на приложението и служи за връзката между другите два - Model и View. Този слой използва логиката на Model, след което предоставя необходимите данни в желания формат на View, където те се визуализират. Класовете на този слой са следните:

- DelegateCommand - клас, който представлява делегат на команда, наследяващ от System.Windows.Input.ICommand. Съдържа вътрешно поле от тип Action, което се подава на конструктора. Имплементира функциите Execute(), извиквайки обекта от тип Action, както и булевата функция CanExecute(), връщайки true.
- ObservableObject - клас, наследник на , който представлява наблюдаваем обект. За целта съдържа поле от тип събитие (event PropertyChangedEventHandler), както и метод, който повдига това събитие при промяна на дадено свойство, чието име се подава като аргумент на функцията;
- BaseViewModel - базов ViewModel, който имплементира функционалността по подразбиране. Този клас е родител на всеки друг ViewModel. За тази цел е генеричен, като отново е въведено ограничение на типа, от който може да бъде инстанциран и то е само за “Entity” обекти (т.е. всички наследници на Entity класа). Съдържа следните полета и методи:
  - private ObservableCollection<T> _filteredEntities - вътрешна променлива от тип ObservableCollection, която ще съдържа списъка с вече филтрираните обекти;
  - public ObservableCollection<T> FilteredEntities - публично стойство (property), чийто get{} метод връща _filteredEntities. В set {} метода, освен присвояване на подадената стойност, се повдига събитие

RaisePropertyChangedEvent с името на променената свойство. Чрез Binding това свойство е свързано с таблицата за визуализиране на вече филтрираните обекти;

- private ObservableCollection<Attribute<T>> _attributes - вътрешна променлива от тип ObservableCollection, която представлява атрибутите на избрания обект;
- public ObservableCollection<Attribute<T>> Attributes - публично стойство (property), чийто get{} метод връща _attributes. В set {} метода, освен присвояване на подадената стойност, се повдига събитие RaisePropertyChangedEvent с името на променената свойство. Чрез Binding това свойство е свързано с падащото меню, предоставящо възможност за избор на атрибути за филтриране измежду всичките атрибути;
- private ObservableCollection<Attribute<T>> _selectedAttributes - вътрешна променлива от тип ObservableCollection, която представлява избраните от потребителя атрибути за филтриране на избрания обект;
- public ObservableCollection<Attribute<T>> SelectedAttributes - публично стойство (property), чийто get{} метод връща _selectedAttributes. В set {} метода, освен присвояване на подадената стойност, се повдига събитие RaisePropertyChangedEvent с името на променената свойство. Чрез Binding това свойство е свързано с контролата за визуализиране и въвеждане на стойности за избраните атрибути, по които ще се извършва филтрирането;
- private List<T> Entities - вътрешна колекция, която представлява списъка от обекти, върху които ще се извършва филтрирането. Подава се на конструктора на класа;
- protected BaseViewModel(List<T> entities) - protected конструктор, който приема списъка с обекти за филтриране. Инициализирането на променливите и взимането на необходимите начални се свойства се извършва в тялото му;
- public void AttrCheckedUnchecked() - публична функция, която извиква метода AttributeCheckedUnchecked() на Attribute класа. Извиква се в code-behind частта на главния прозорец, където е свързан с тикчетата за избор на атрибут като критерий за филтриране;
- public ICommand FilterCommand - свойство от тип ICommand, което връща нова инстанция на DelegateCommand. Действието, което се извършва е FilterEntities;
- protected void FilterEntities() - действие за филтриране на подадената колекция от обекти на базата на въведените атрибути. Изивква метода FilterEntitiesBasedOnAttributes() на Filter в тялото си, както и отваря нов прозорец от тип SearchWindow, в който ще се визуализират филтрираните данни. За целта, стойността на DataContext на отворения прозорец се слага на текущия ViewModel;

- ➢ public ICommand ClearFiltersCommand - свойство от тип ICommand, което връща нова инстанция на DelegateCommand. Действието, което се извършва е ClearFilters;
- ➢ public void ClearFilters() -> служи за изчистване на текущо-избраните филтри посредством LINQ заявка.

- ❖ MovieViewModel - ViewModel за работа с обекти от тип Movie. Наследява BaseViewModel и го имплементира за клас Movie. Съдържа само публичен конструктор, в който се извиква бащиния конструктор, като посредством DatabaseManager.GetMovies() се подава списъка с филми за филтриране;
- ❖ CarViewModel - ViewModel за работа с обекти от тип Car. Наследява BaseViewModel и го имплементира за клас Car. Съдържа само публичен конструктор, в който се извиква бащиния конструктор, като посредством DatabaseManager.GetCars() се подава списъка с коли за филтриране;

View

Този слой служи за визуализирането на данните, предоставени от ViewModel, пред потребителя. Дефинира се с XAML и ограничен code-behind. Класовете тук са следните:

- ❖ MainWindow - code-behind на главния прозорец MainWindow. Съдържа полета за всеки един тип ViewModel с цел динамична едностраницова смяна между обектите за филтриране чрез смяна на ViewModel. Освен конструктора си, съдържа следните методи:
  - ➢ public void EntitySelectionChanged(...) - публичен метод, който е свързан с промяна на избрания елемент в падащото меню за избор на списък от обекти за филтриране. В зависимост от избраното поле в контролата, се извършва динамично превключване на DataContext към правилния ViewModel. Тъй като ViewModel инстанциите се съхраняват под формата на инициализирани в конструктора обекти, се предоставя запомняне на въведените данни за всеки един списък от обекти за филтриране при промяна на избора на списък;
  - ➢ public void chkAttr_CheckedAndUnchecked(...) - публичен метод, който се свързва с тикчетата в контролата за избор на атрибути. За целта се извиква методът AttrCheckedUnchecked() на съответния ViewModel;
  - ➢ private static dynamic Cast(dynamic obj, Type castTo) - вътрешна функция, която служи за динамично кастване към желан тип. Използва се за динамично кастване към типа на желания ViewModel с цел едностраницово опресняване.
- ❖ SearchWindow - code-behind на прозореца, в който се визуализират вече филтрираните обекти. Съдържа метод OnAutoGeneratingColumn(), който се извиква при генерирането на колоните на таблицата за визуализиране на обектите. Служи за изключване на създаване на колона от тип Id.

Приложението съдържа два XAML изгледа:

- MainWindow.xaml - изгледът е структуриран под формата на Grid с редове и колони, като контролите са разположени в тях.
  - **ред 0, колона 0** - StackPanel контрола, която отговаря за избора на списък от обекти за филтриране. Съдържа Label с указващ текст и ComboBox падащо меню, от което потребителят може да избере обект за филтриране от всички възможни. Както бе споменато по-горе, този SelectionChanged събитието на тази ComboBox контрола е свързана със съответния метод;
  - **ред 1, колона 0** - ComboBox контрола, която служи за избор измежду всички атрибути на избрания обект. Чрез DataTemplate се създава CheckBox контрола за всеки един от атрибутите, които са свързани чрез Binding към ItemsSource на главната контрола. IsChecked е свързано към Checked полето на атрибута, а Content - към AttributeName. Събитията Checked/Unchecked са свързани към функцията от code-behind частта, която извика съответната функция от ViewModel;
  - **ред 1, колона 1** - Grid контрола, която съдържа двата бутона “Search” и “Clear Filters”. Първият бутон е свързан чрез Binding към FilterCommand командата на ViewModel, а вторият - към ClearFiltersCommand;
  - **ред 2, колона 1** - Grid контрола, която съдържа в себе си Label контроли, които служат за указване на имената на колоните при въвеждане на атрибутите;
  - **ред 3, колона 1** - ListView контрола, която чрез DataTemplate създава необходимите контроли за въвеждане на данни за избраните атрибути. Всеки ред съдържа Label (Binding към AttributeName), ComboBox, който се използва за избор на вида сравнение на критерия, както и TextBox, в който потребителят въвежда стойността за избрания атрибут. Тъй като някои от опциите на ComboBox контролата са валидни само за определен тип данни, те се активират/деактивират чрез DataTrigger елементи, които чрез Binding са свързани към IsComparible полето на всеки атрибут.

  Важно е да се отбележи, че изгледът е структуриран по начин, позволяващ работа с всички контроли дори при смаляване на прозореца до минималната му големи, която е 250x500. Това се постига чрез Height и Width атрибутите на колоните и редовете в Grid контролата.

- SearchWindow.xaml - изгледът се използва за визуализиране на вече филтрираните избрани обекти. Реализацията е постигната чрез DataGrid контрола, чийто ItemsSource чрез Binding е свързан с FilteredEntities свойството на съответния ViewModel. При автоматичното генериране на колоните е указана функция в code-behind частта с цел избягване на визуализирането на колона за Id. Всеки втори ред има алтернативен цвят “LightSalmon”.

Ръководство за програмиста

Обща част

Двата сценария за по-нататъшна разработка, които ще бъдат покрити, са следните: **добавяне на нов обект за филтриране към текущата структура на изгледите** и **добавяне на нов обект за филтриране с нов произволен изглед**. За целта е необходима среда за разработване с инсталирани рамките Microsoft.NETCore.App 3.1.0, Microsoft.WindowsDesktop.App.WPF 3.1.0, както и nugget пакета Entity Framework 6.4.4.

Стъпките за реализация на сценариите използват Code-First подхода при създаване на таблицата с обекти в базата данни. Независимо от избора на сценарий, следните стъпки са еднакви и се изпълняват и в двата случая:

1. Добавяне на клас, описващ новия обект в директория “Model/”. Класът трябва да наследява клас “Entity” и да има добавена анотация [Table(Table_name))], която да указва името на таблицата, която ще се създаде. **ВАЖНО**: За улеснение, от тук нататък създаденият клас ще се назовава като “NewEntity”;
2. Добавяне на публично свойство от тип DbSet<NewEntity> в DatabaseContext класа, намиращ се под “Model/Database/”. **ВАЖНО**: В конструктора на DatabaseContext, който извиква базовия, трябва да бъде подаден низ за свързване (connection string) към локалната база, която ще се използва за разработка;
3. Добавяне на метод, който връща колекция от тип List<NewEntity>, съдържаща записите от базата, върху които ще се извършва филтрирането. Методът се добавя в DatabaseManager класа, намиращ се под “Model/Database/”;
4. Създаване на миграция и опресняване на базата данни. В конзолата на проекта (Package Manager Console, Package source: nuget.org) се изпълняват следните команди:
    a. <Опционално> “Enable-Migrations” - в случай, че миграциите не са включени;
    b. “Add-Migration <MigrationName>” - създаване на клас за миграция с подходящо дескриптивно име;
    c. “Update-Database” - изпълнение на последния миграционен файл, създаден от предходната команда. Тази команда прави промяна в схемата на базата данни.

При правилна работа и конфигурация на свързаността с базата данни, дотук трябва да бъдат налични следните точки:

- създадена таблица в базата данни, съобразно описания в класа модел;
- връзка между приложението и базата данни;
- метод в DatabaseManager, връщаш колекция от новия обект, взета от базата данни.

5. Създаване на ViewModel клас за новия модел под директория “ViewModel/” по подобие на MovieViewModel или CarViewModel. Името на новия модел е препоръчително да присъства в името на ViewModel класа (напр. - NewEntityViewModel). Новият ViewModel трябва да наследява генеричния BaseViewModel клас и да го имплементира за новия модел (NewEntity), напр.
   public class NewEntityViewModel : BaseViewModel<NewEntity>

   При извикването на базовия конструкт е необходимо да бъде подадена колекция от тип List<NewEntity>, която ще бъде филтрирана (напр. тази, връщана от метода от т. 3).

   Забележка: Точките в тази стъпка са достатъчни за получаване на функционалността за филтриране на приложението по подразбиране. При нужда от допълнителна функционалност, новият ViewModel клас може да презапише функциите на родителския клас.

<u>В зависимост от избрания сценарии, следващите точки в изпълнението се различават!</u>

Добавяне на нов обект за филтриране към текущата структура на изгледи

6. В изгледа MainWindow.xaml в StackPanel контролата с име “EntitiesSP”, под ComboBox контролата с име “EntityCB”, се добавя нов <ComboBoxItem> с името на новия обект за филтриране (напр. <ComboBoxItem>NewEntity<ComboBoxItem>;
7. В code-behind класа MainWindow (MainWindow.xaml.cs) под директория “View/” се добавя вътрешно поле от тип NewEntityViewModel, което се инициализира в конструктора;
8. В метода EntitySelectionChanged(...) в MainWindow (който е свързан със SelectionChanged събитието на “EntityCB” ComboBox) се добавя нова опция (case) в “switch-case” оператора по подобие на вече съществуващите. Важно е да се отбележи, че стойността на case **ТРЯБВА** да отговаря на името, зададено в <ComboBoxItem> в т.7. В тялото на новия case, DataContext на прозореца трябва да бъде сменен на новия ViewModel (т.е. DataContext = NewEntityViewModel).

Добавяне на нов обект за филтриране с нов произволен изглед

6. Създаване на произволен изглед под директория “View/”. Визуализацията и подредбата на елементите е по преценка на разработчика. Тук ще бъдат описани главните съображения при извършване на Binding връзки с цел реализиране на пълната функционалност:
    a. <Опционално> При наличие на контрола, служеща за избор за филтриране между различни елементи, съществуват два варианта:
        i. Динамична смяна на DataContext към съответния ViewModel на базата на избран обект (едностраницово приложение; текуща реализация);
        ii. Връзка от тип “1:1” между ViewModel и View - операции само върху един обект в една страница.

        И в двата случая, изгледът трябва да използва като DataContext създадения ViewModel (NewEntityViewModel);

    b. Контрола, служеща за избор на атрибути на обекта, спрямо които ще се филтрира - в ItemsSource чрез Binding трябва да бъде направена връзка с Attributes свойството на ViewModel (от тип ObservableCollection<Attribute<NewEntity>>). За целта в контролата трябва да бъде използван DataTemplate, което ще позволи динамично зареждане и създаване на контролата на базата на подадената колекция;
        i. **ВАЖНО**: За избиране на даден атрибут (използване на контрола CheckBox), се извършва следната връзка: IsChecked=”{Binding Checked}”, където Checked е свойство на Attribute<NewEntity>. Checked и Unchecked събитията ТРЯБВА да бъдат свързани през code-behind към метода AttrCheckedUnchecked() на съответния ViewModel (който от своя страна извиква метода на Attribute) ;
        ii. За визуализация на името на атрибута, който ще се попълва, Binding връзката е към AttributeName.

    c. Контрола, служеща за въвеждане на стойности на избраните атрибути - в ItemsSource чрез Binding трябва да бъде направена връзка със SelectedAttributes свойството на ViewModel (от тип ObservableCollection<Attribute<NewEntity>>). Това е динамично-променящо се свойство, което във всеки момент съдържа избраните за филтриране атрибути. За целта в контролата трябва да бъде използван DataTemplate, което ще позволи динамично зареждане и създаване на контролата на базата на подадената колекция;
        i. При използване на различните видове сравнение трябва да бъдат създадени контроли с множествен избор (ComboBox), чийто избран индекс (SelectedIndex) трябва посредством Binding да бъде свързан с ComparisonType свойството на избрания атрибут от колекцията;

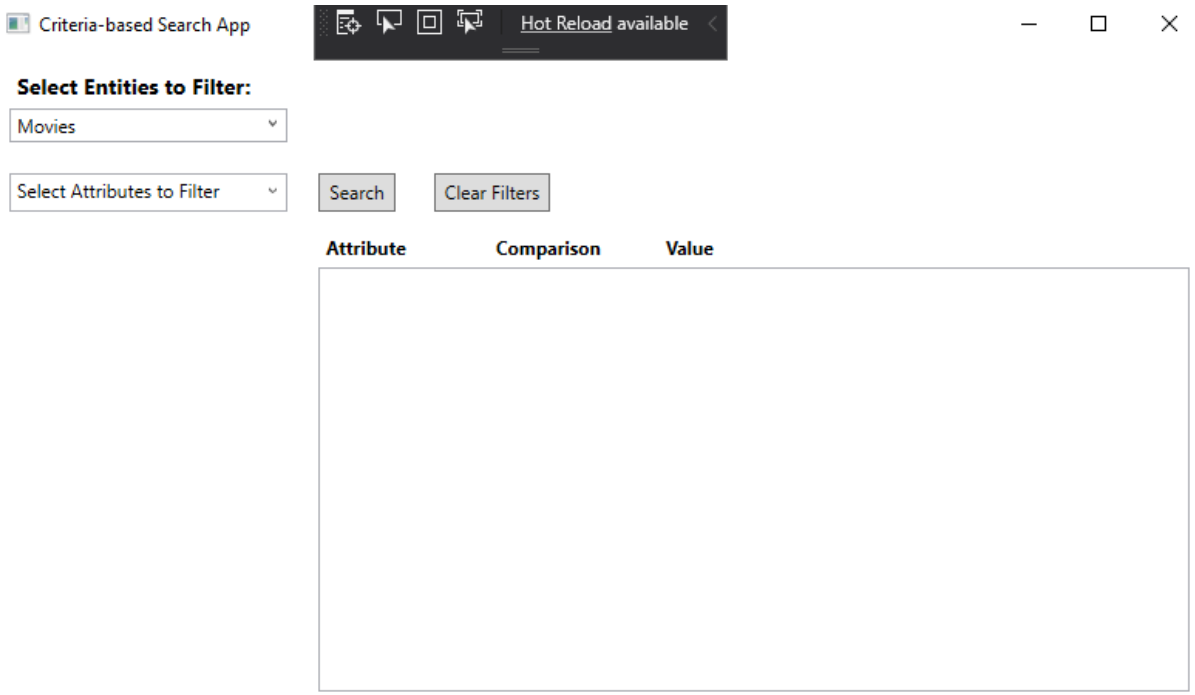
ii. За лимитация кои видове сравнение да се представят за кои атрибути - използване на DataTrigger, който променя IsEnabled свойството на съответната контрола за избор на сравнение на базата на свойството IsComparible на атрибута (отново чрез Binding);
iii. Използване на TextBox за въвеждане на стойност на избраните атрибути - Text полето трябва да е свързано чрез Binding с AttributeValue свойството на атрибута.

d. Бутони за филтриране и изчистване на филтрите
   i. Бутон за филтриране - Command полето му трябва да бъде свързано чрез Binding с командата за филтриране във NewEntityViewModel (FilterCommand по подразбиране). Тази команда трябва да извиква метода за филтриране в модела (по подразбиране FilterEntitiesBasedOnAttributes(...) във Filter.cs);
   ii. Бутон за изчистване на филтрите - Command полето му трябва да бъде свързано чрез Binding с командата за изчистване на филтрите във NewEntityViewModel (ClearFiltersCommand по подразбиране). Тази команда трябва да извиква метода за изчистване на филтрите (по подразбиране ClearFilters(...) в BaseViewModel.cs).

e. При наличие на контрола, служеща за визуализация на вече филтрираните обекти, Binding връзката за ItemsSource трябва да бъде направена към FilteredEntities свойството на ViewModel.

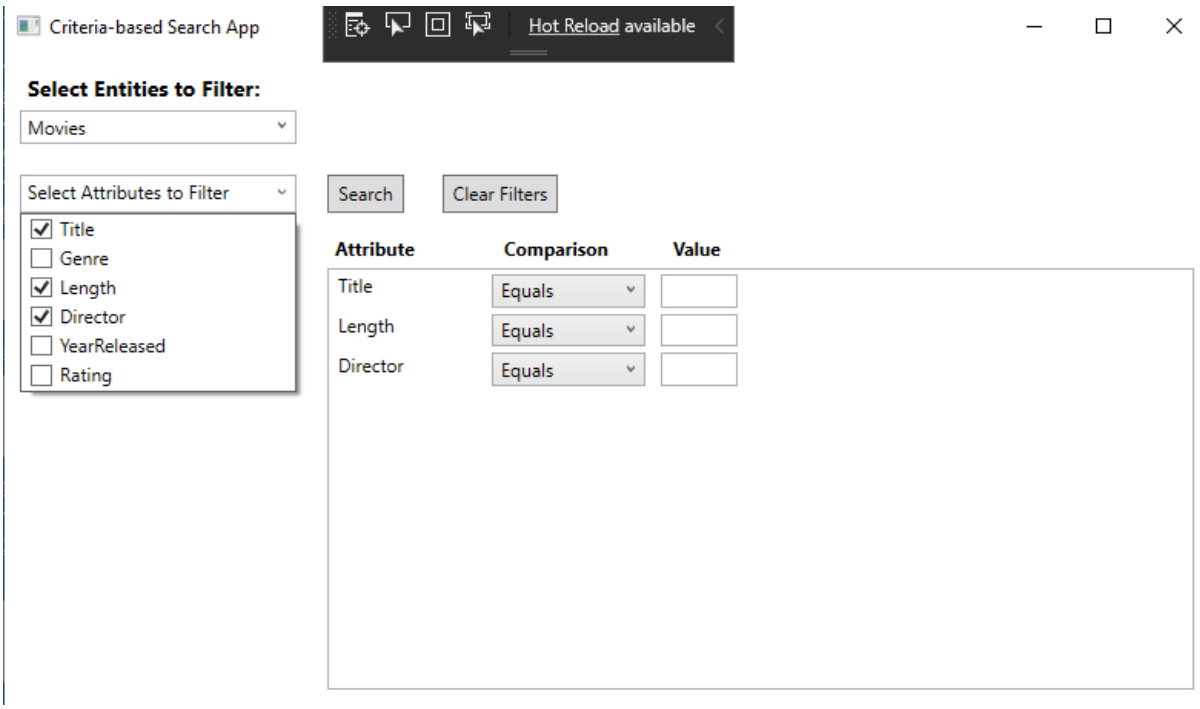
Демонстрация на работоспособността

На фигурите по-долу е демонстрирана работата на проекта. На *фиг. 1* е показано най-първоначалното стартиране на програмата, при което се визуализира главният изглед, съдържащ необходимите контроли - избор на обект за филтриране, избор на атрибути за филтриране, бутони за филтриране и изчистване на филтрирането и място за попълване на стойността на критериите.



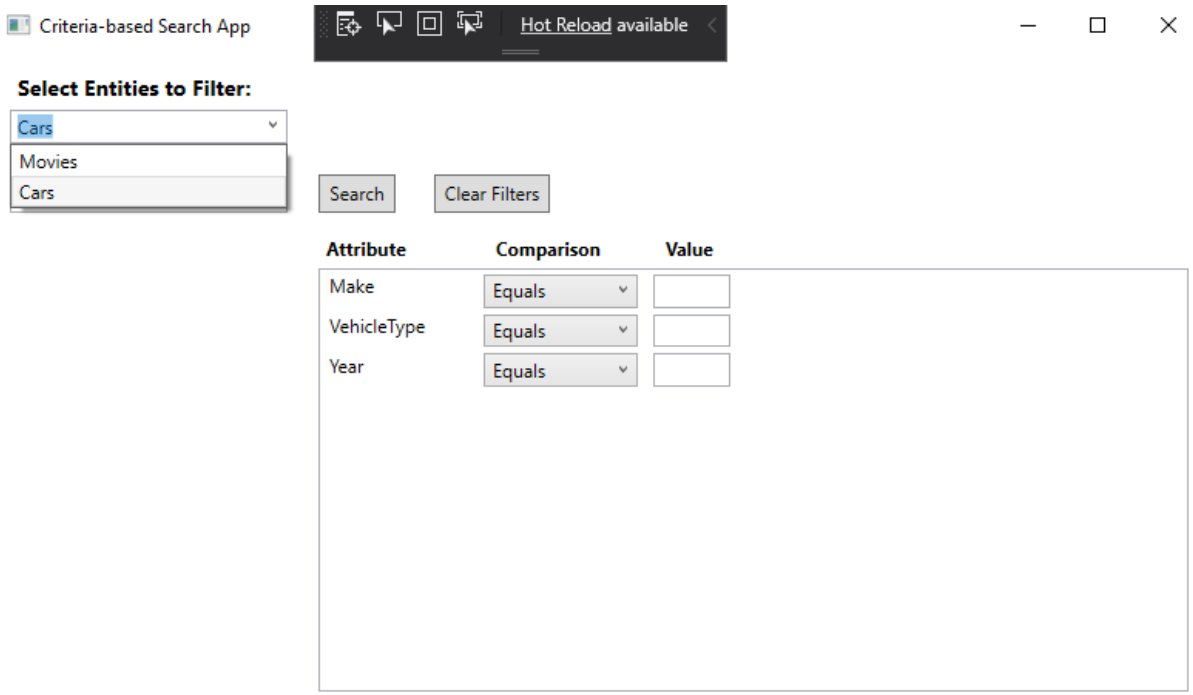
Фиг. 1. Главен изглед на програмата

На *фиг. 2* е показан изборът на атрибути за един от обектите (филми). Както се вижда, всички атрибути на обекта се визуализират, заедно с поле за тикче до тях. При отбелязване/премахване на тикчето до името на избрания атрибут, той се добавя/премахва като критерий за филтриране в контролата вдясно.



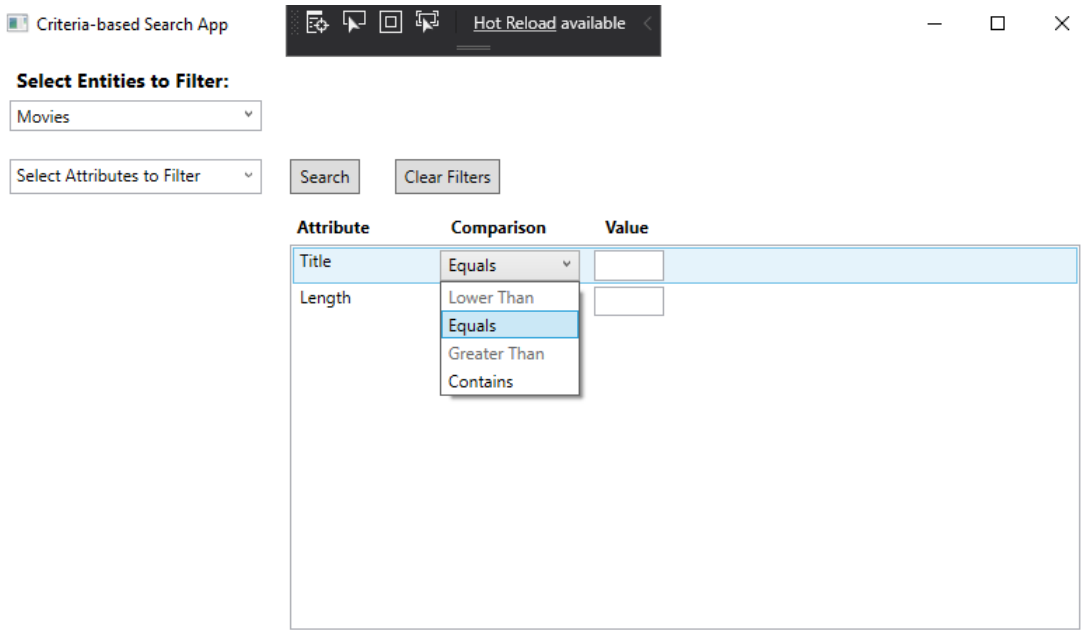
Фиг. 2. Избор на атрибути за филтриране

На *фиг. 3* показва възможността за избор между различните обекти, които имат различни атрибути за филтриране.



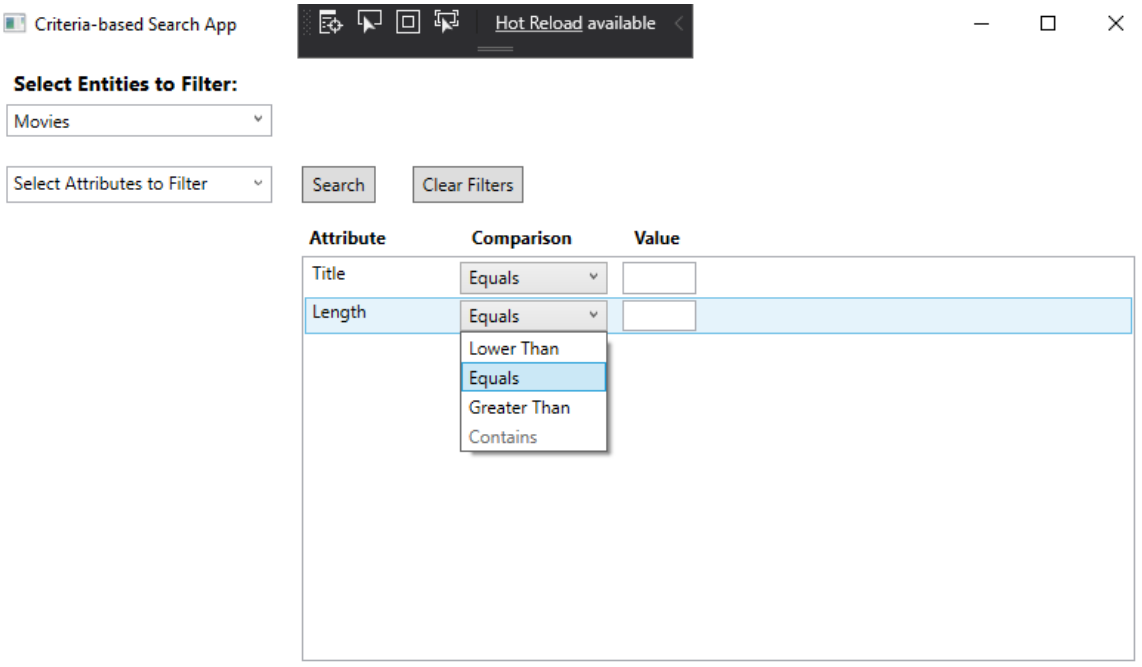
Фиг. 3. Избор между различни обекти за филтриране

На *фиг. 4* са показани различните опции за филтриране на поле от тип низ (string). По подразбиране типа сравнение е “равно на”, при което низовете трябва да съвпадат напълно. Другата опция за низове е “съдържа се в”, при която се проверява дали въведената стойност в контролата се съдържа в желания атрибут.



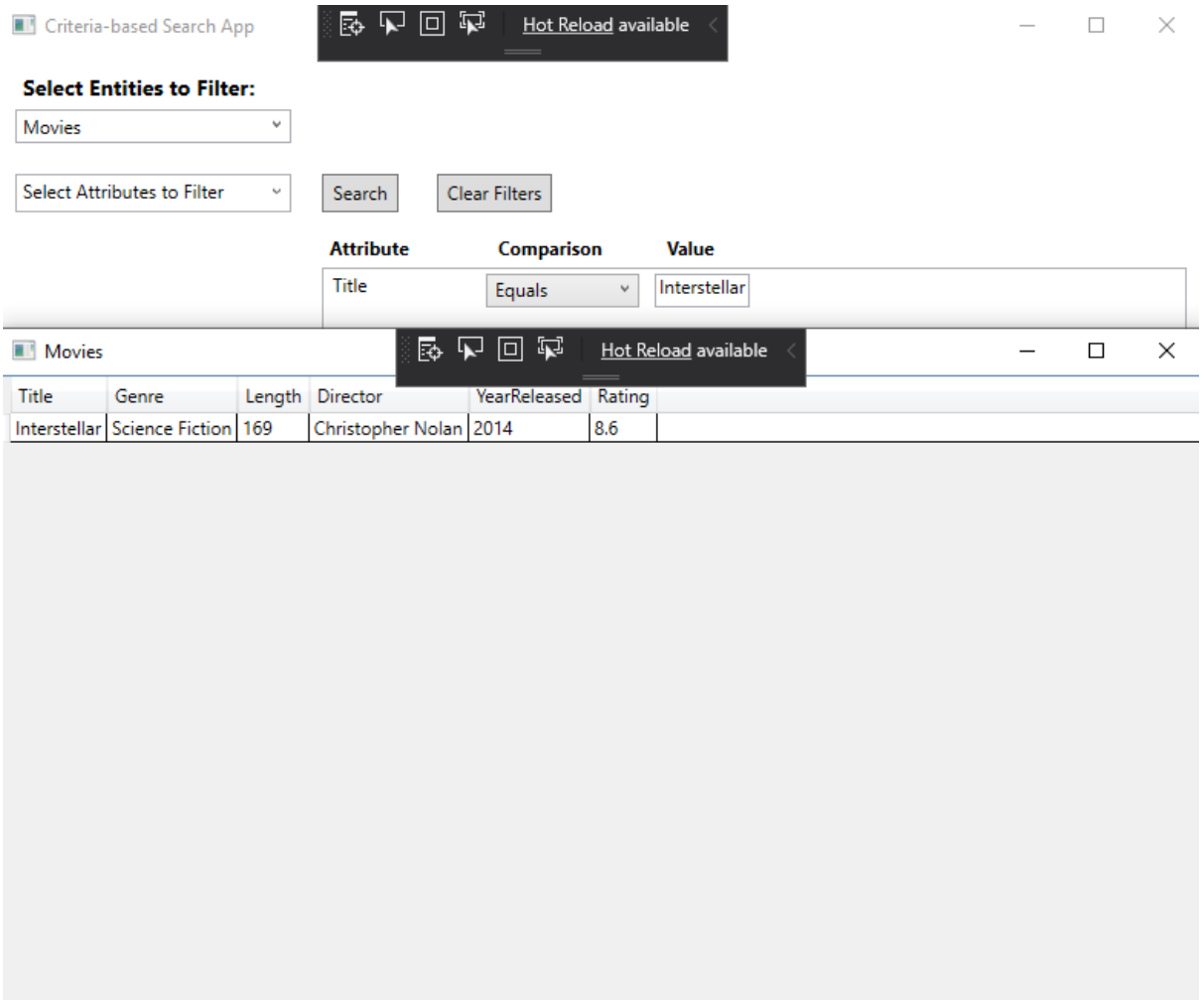
Фиг. 4. Опции за филтриране на поле от тип низ (string)

Фиг. 5 показва различните опции за филтриране на числови полета (double, int).



Фиг. 5. Опции за филтриране на поле от числов тип (double, int)

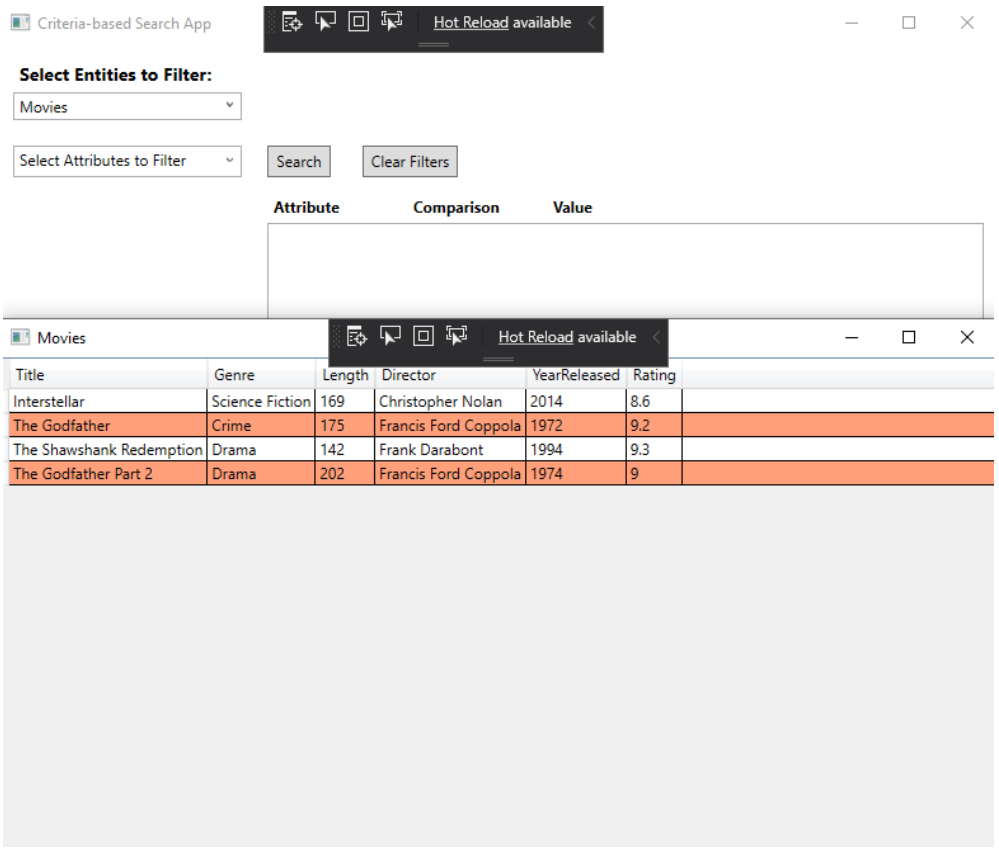
На *фиг. 6* е показан резултатът от филтриране на филмите на базата на поле “Title” (заглавие), което трябва да бъде “Interstellar”. Както се вижда, резултатът е единствен филм с желаното име. Визуализацията на филтрираните обекти се извършва в отделен прозорец, който се отваря при натискане на бутона “Search”.



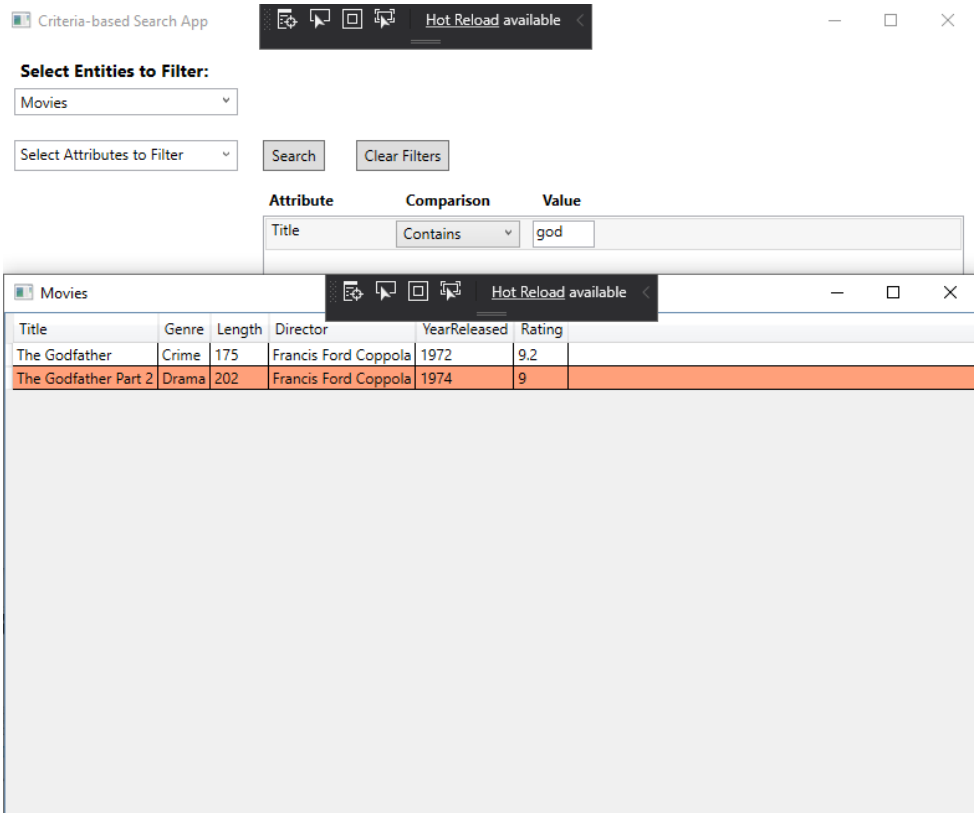
Фиг. 6. Филтриране на филми

На *фиг. 7* е показан резултатът от филтриране на филмите, когато не е избрано нито едно поле за критерий. Резултатът е визуализиране на всички налични филми.

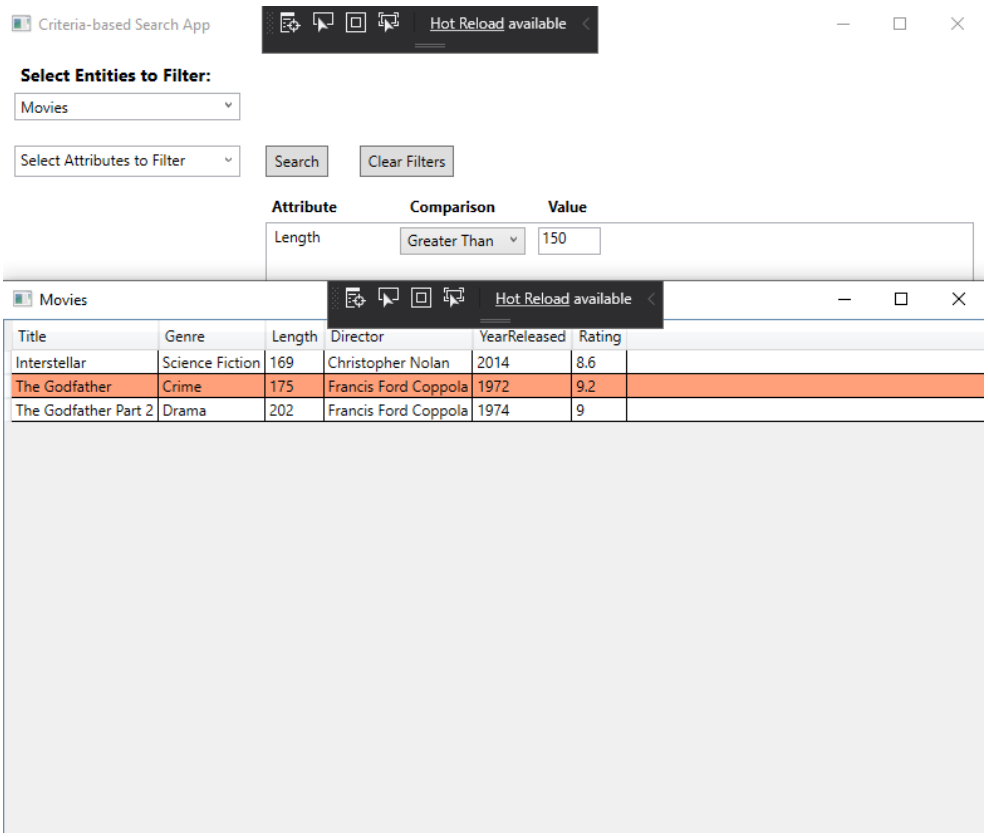
Ф*иг. 8* показва филтрирането чрез опция “contains”, което връща като резултат филмите, които съдържат в името си низа “god”. Важно е да се отбележи, че формата е case-insensitive. Ф*иг. 9* показва филтрирането чрез опция “greater than”, което връща като резултат филмите, които имат времетраене, по-голямо от посоченото.



Фиг. 7. Филтриране на филми без избор на критерии

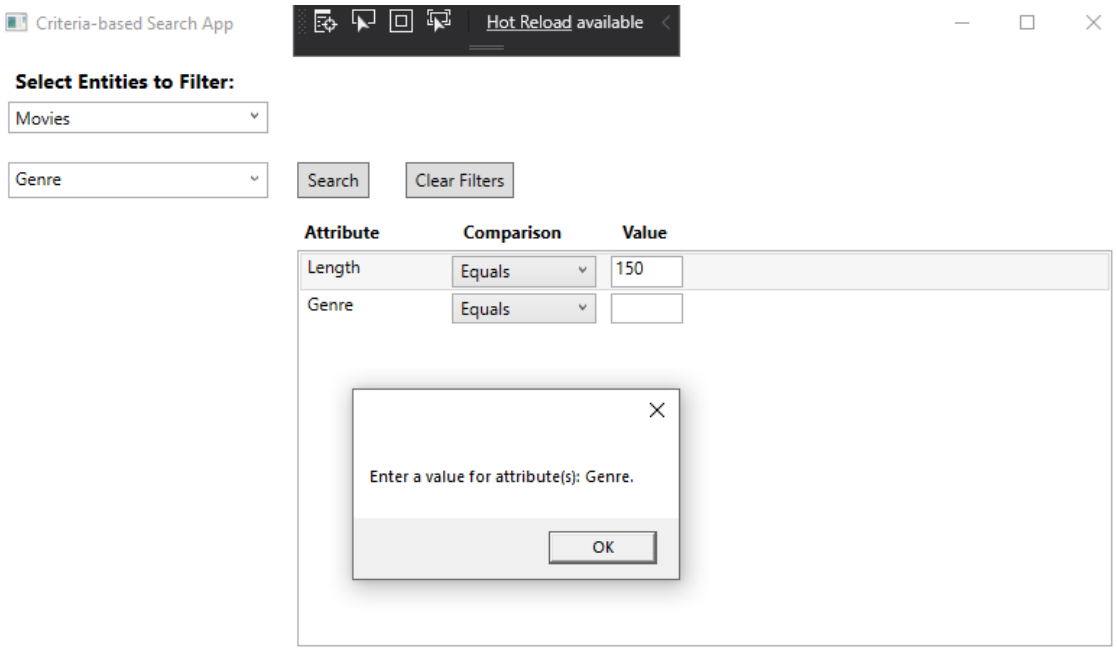


Фиг. 8. Филтриране на филми чрез "contains"



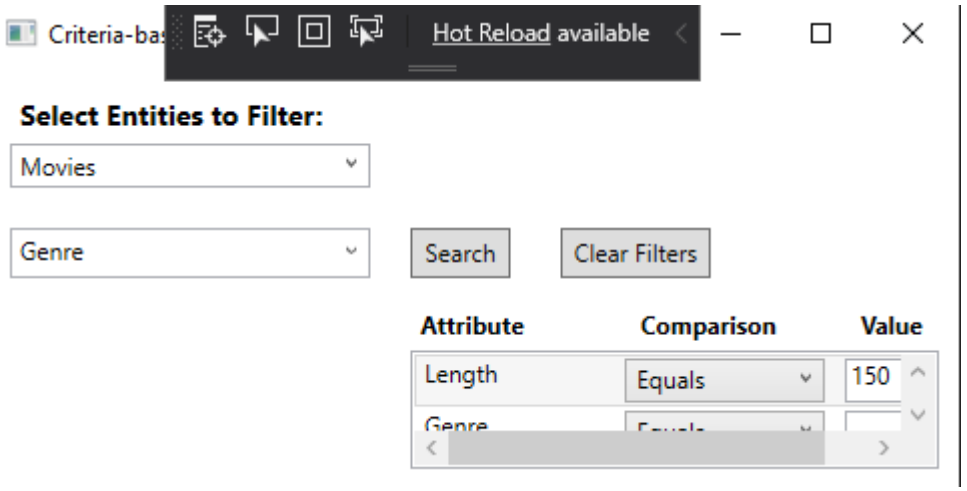
Фиг. 9. Филтриране на филми чрез "greater than"

На *фиг. 10* е представена грешката, която се поражда, когато някое от избраните полета за филтриране не е попълнено. При повече от едно липсващо поле, програмата информира потребителя за всички липсващи атрибути.



Фиг. 10. Грешка при невъведени полета за филтриране

Ф*иг. 11* представя минималната големина на прозореца за филтриране. Както се вижда, всички контроли са достъпни и цялата функционалност на приложението е използваема, благодарение на слайдерите за скролиране.



Фиг. 11. Минимална големина на прозореца за филтриране