



Програмни среди

Документация на Game Finder App

Проект изготвен по тема 2

Пламен Цветанов
121220013, 43 "а"
24.4.2023 г.

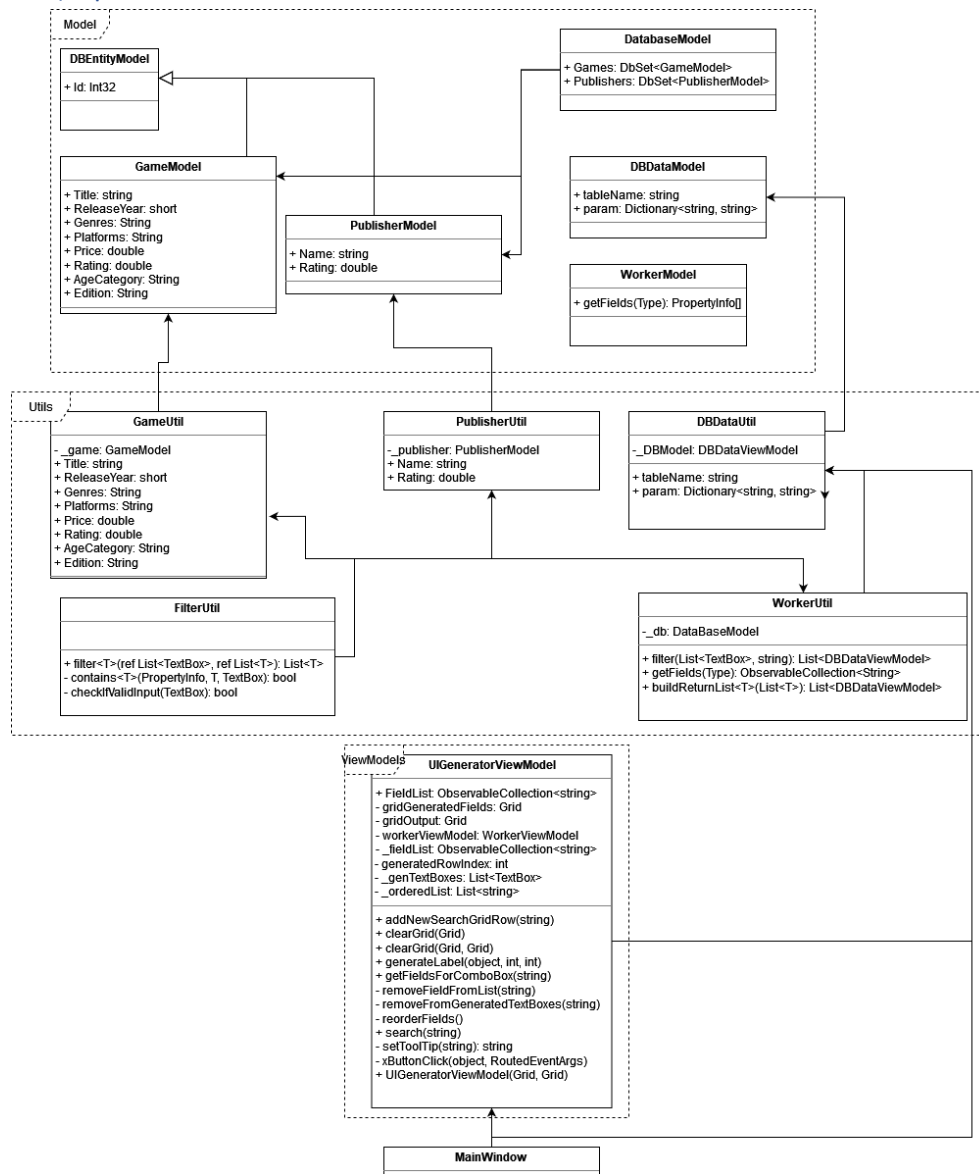
Съдържание

1. Въведение в системата	2
2. Специфика на системата	2
3. Специфика на базите данни.....	5
4. Начин на работа на системата	6
5. Наръчник за бъдеща разработка:.....	9

1. Въведение в системата

Системата представлява приложение за търсене на видео игри и издатели на видео игри. Спрямо потребителския избор за това какъв тип обекти да се филтрират, се генерират полета за попълване, по които се осъществява филтрирането. Полученият резултат от операцията се извежда в таблица до контролите за филтриране. Спрямо избрания тип обекти за филтриране се генерират различни таблици.

2. Специфика на системата



На дадената UML диаграма се вижда разделението на системата на Model-View-View елементи, като в Model категорията попадат:

- DatabaseModel - съдържа информация за създаване на конекция с базата данни, както и всички необходими DB сетове.
- DBDataModel - съдържа информация за дадено поле и неговата стойност.
- DBEntityModel - съдържа само клас променлива, която е идентификатор на обект в базата. Служи за обозначаване на обекти като таблици в базата.
- GameModel - съдържа информация за видео игра, наследява DBModel и е връзка с таблица GameModels в базата данни:
 - Title - string - заглавието на играта.
 - ReleaseYear - short - година на издаване на играта.
 - Genres - String - жанрове, в които попада играта - елементи, разделени от запетайки.
 - Platforms - String - платформи, на които играта е достъпна.
 - Price - double - цена на играта (служи за търсене по конкретна цена)
 - Rating - double - оценка на играта.
 - AgeCategory - String - възрастова категория на играта.
 - Edition - String - версия на играта.
- PublisherModel - съдържа информация за издател, наследява DBEntityModel и е връзка с таблица PublisherModels в базата данни:
 - Name - string - наименование на издателя.
 - Rating - double - оценка на издателя.
- WorkerModel - съдържа метод, който връща информация за всички полета на даден клас под формата на масив.
- GameUtil - Притежава всички полета, които притежава моделът. По този начин се разкача логиката и няма наличие на модел обекти във view-то. Съдържа метод filter, който извършва филтрирането за този тип обекти.
- PublisherUtil - Притежава всички полета, които притежава моделът. По този начин се разкача логиката и няма наличие на модел обекти във view-то. Съдържа метод filter, който извършва филтрирането за този тип обекти.
- DBDataUtil- View-Model, който свързва модела с view-то. Притежава всички полета, които притежава моделът. По този начин се разкача логиката и няма наличие на модел обекти във view-то.
- FilterUtil - съдържа основната логика за филтриране и се използва от всеки View-Model, който извършва филтриране.
- UIGeneratorViewModel - съдържа цялата логика за генериране и манипулация на user interface модули.

- WorkerUtil - съдържа метод filter, който е връзка между View и другите Utils, които са обекти на филтриране.
- MainWindow - графичен потребителски интерфейс. Съдържа контроли, с които потребителят може да избира обекти и атрибутите, по които да ги филтрира, като резултатът ясно се вижда в таблица до контролите за филтър.
 - Code-Behind част:
 - cbFields_SelectionChanged(object, SelectionChangedEventArgs) - методът описва действие при промяна на избран елемент от comboBox Fields.
 - cbSearchOptions_SelectionChanged(object, SelectionChangedEventArgs) - методът описва действие при промяна на избран елемент от comboBox Options.
 - btnSearch_Click (object sender, RoutedEventArgs e) - методът задейства основната логика на системата. Чрез натискане на бутон „Search“ генерираме таблица с изходни данни.
 - btnClear_Click(object sender, RoutedEventArgs e) - методът изчиства генерирана таблица с изходни данни.
 - btnReturn_Click(object sender, RoutedEventArgs e) - методът изчиства всички генерирани таблици и връща системата в стартовата ѝ точка.
 - XAML: съдържа comboBox с етикет в горния ляв ъгъл, който позволява достъп до обектите и параметрите за филтриране. Под тях се генерира таблица с входни данни за филтриране. Под тях се намират три бутона за контрол на дейността на програмата - return, search, clear. Цялата дясна половина е отредена за генерираната част.

3. Специфика на базите данни



Диаграмата показва таблиците GameModels и PublisherModels в базата данни. Наличните полета, които вече са споменати в точка 2.

4. Начин на работа на системата

При стартиране на приложението се визуализира следният екран:

The screenshot shows a web application interface. At the top left, there is a label "Search:" followed by a dropdown menu. Below this, there is a large, empty rectangular area, likely a placeholder for search results. At the bottom of the interface, there are three buttons: "Return", "Search", and "Clear". A vertical scrollbar is visible on the right side of the page.

При натискане върху полето до етикет “Search”, потребителят бива подканен да избере типа информация, за която ще филтрира данни:

This screenshot shows the "Search:" label and the dropdown menu that appears when it is clicked. The dropdown menu is open, displaying two options: "Games" and "Publishers".

При направен избор се променя стойността на етикета и се визуализира ново падащо меню:

This screenshot shows the "Fields:" label and the dropdown menu that appears when it is clicked. The dropdown menu is open, displaying a list of fields: "Title", "ReleaseYear", "Genres", "Platforms", "Price", "Rating", "AgeCategory", and "Edition". The "Title" option is currently selected and highlighted.

При избор на елемент от падащото меню се генерират етикет и текстово поле:

Title:

ReleaseYear:

Genres:

Бутонът до всяко текстово поле го скрива, като повторната му визуализация е възможна чрез селектиране в падащото меню. При задържане на мишката върху текстово поле се визуализира подсказка за потребителя:

Field requires a string value for title attribute.

Чрез натискане на бутона:

Се отправя заявка, която генерира таблица с всички записи, които отговарят на введените параметри от потребителя:

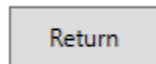
Title	ReleaseYear	Genres	Platforms
Red Dead Red	2018	Action, Advent	PS4, Xbox One
God of War	2018	Action, Advent	PS4
Marvel's Spide	2018	Action, Advent	PS4
Assassin's Cre	2018	Action, RPG	PS4, Xbox One

За удобство на потребителя таблицата е в умален вариант и е включена възможността за хоризонтално и вертикално скролиране.

Чрез бутонът

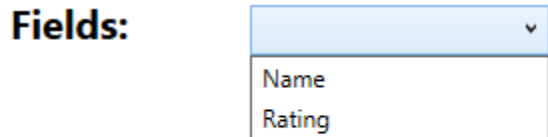
Се изчиства генерираната таблица.

Чрез бутонът



Потребителят се връща в началната страница на приложението и може да избере кой тип данни да филтрира.

Ако потребителят избере Publishers се визуализира ново падащо меню:



При избор на елемент от падащото меню се генерират етикет и текстово поле:

Name:

Rating:

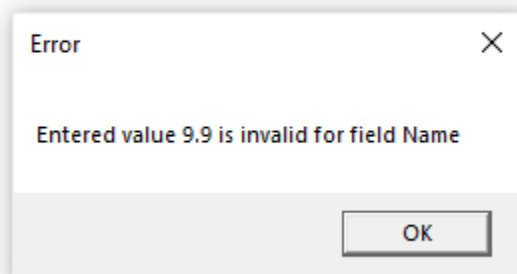
При неспазване на зададените ограничения от подсказката, която се появява при задържане на мишката върху дадено поле:

Name:

Field requires a string value for name attribute.

Се визуализира диалогов прозорец за грешка:

Name:



5. Наръчник за бъдеща разработка:

Стъпки за добавяне на нов тип обект за филтриране:

- Добавяне на нов модел, който наследява DBEntity - по този начин се генерира code first таблица в базата.

```
namespace GameFinderAppV2.Models
{
    6 references
    public class RegionModel : DBEntityModel
    {
        1 reference
        public string name { get; set; }
        1 reference
        public long population { get; set; }
        1 reference
        public string timezone { get; set; }
    }
}
```

Забележка: не забравяйте да направите класа публичен

- Добавяне на поле от тип DbSet<модел> в DatabaseModel

```
namespace GameFinderAppV2.Models
{
    9 references
    public class DatabaseModel : DbContext
    {
        private static string _connString = "Server=localhost\\SQLEXPRESS;Database=Game_Database_V2;Trusted_Connection=True;";
        3 references
        public DatabaseModel() : base("Server=localhost\\SQLEXPRESS;Database=Game_Database_V2;Trusted_Connection=True;") { }
        1 reference
        public DbSet<GameModel> Games { get; set; }
        1 reference
        public DbSet<PublisherModel> Publishers { get; set; }
        1 reference
        public DbSet<RegionModel> Regions { get; set; }
    }
}
```

- Добавяне на ViewModel с метод за филтриране по зададен шаблон

```
namespace GameFinderAppV2.ViewModels
{
    6 references
    public class RegionViewModel
    {
        private static DatabaseModel _db = new DatabaseModel();
        private RegionModel region;
        0 references
        public string name { get { return region.name; } }
        0 references
        public long population { get { return region.population; } }
        0 references
        public string timezone { get { return region.timezone; } }

        1 reference
        public RegionViewModel(RegionModel r) { region = r; }

        1 reference
        public static List<RegionViewModel> filter(ref List<TextBox> generatedTextBoxes)
        {
            List<RegionModel> regionModel = _db.Regions.ToList();
            List<RegionModel> filtered = FilterViewModel.filter(ref generatedTextBoxes, ref regionModel);

            List<RegionViewModel> ret = new List<RegionViewModel>();
            foreach (RegionModel region in filtered)
            {
                ret.Add(new RegionViewModel(region));
            }

            return ret;
        }
    }
}
```

- Добавяне на if statement във filter метод на WorkerViewModel, в който извикваме филтрирането на новосъздадения елемент

```

1 reference
public List<DBDataViewModel> filter(List<TextBox> generatedTextBoxes, string table)
{
    List<DBDataViewModel> result = new List<DBDataViewModel>();
    if (table.Equals("GameModel"))
    {
        result = buildReturnList(GameViewModel.filter(ref generatedTextBoxes));
    } else if (table.Equals("PublisherModel"))
    {
        result = buildReturnList(PublisherViewModel.filter(ref generatedTextBoxes));
    }
    else if (table.Equals("RegionModel"))
    {
        result = buildReturnList(RegionViewModel.filter(ref generatedTextBoxes));
    }

    return result;
}

```

- Добавяне на запис в search полета като името трябва да бъде името на модела без „Model“ и с добавен символ ,s‘ накрая (RegionModels -> Regions):

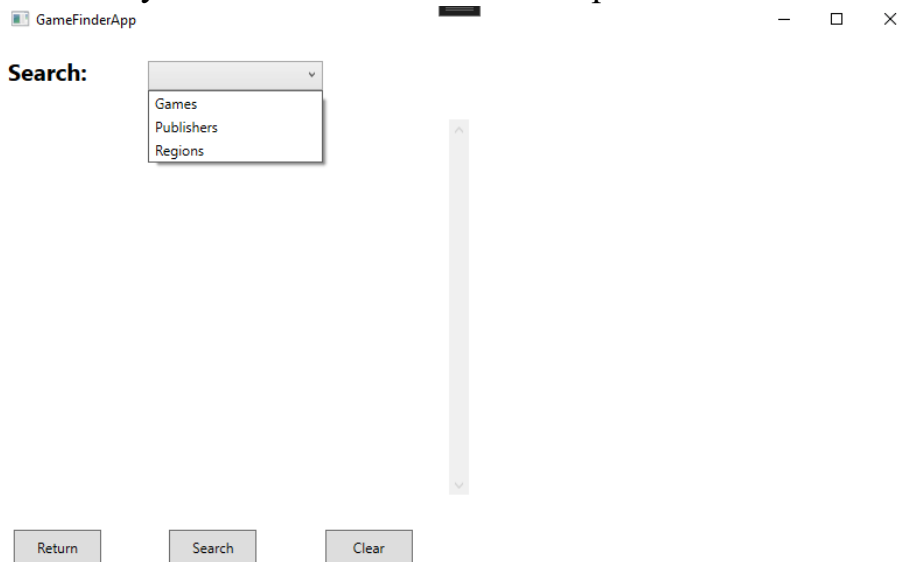
```

public MainWindow()
{
    InitializeComponent();
    DataContext = this;
    SearchModels = new ObservableCollection<string> { "Games", "Publishers", "Regions" };
    uiGenerator = new UIGeneratorViewModel(ref gridGeneratedFields, ref gridOutput);
}


```

Забележка: Трябва моделите да бъдат в един namespace. Не нарушавайте конвенцията на наименоване.

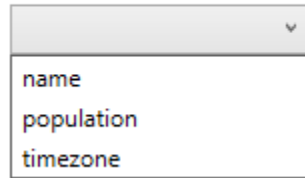
Така получаваме следния изглед на приложението:



При избор на новосъздадения елемент:

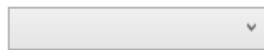
 GameFinderApp

Fields:



При въвеждане на даден критерий за филтриране имаме:

Fields:



timezone:



name	population	timezone
Oceania	42000000	UTC+10:00
Australia	25600000	UTC+10:00

И виждаме, че създаването на нов елемент и филтрирането му работят успешно.

Имплементиране на нов View в логиката изисква обръщане само към View-Model-ът `UIGeneratorViewModel`. Спрямо изискванията на новото View могат да се използват следните методи от `UIGenerator`:

- `addNewSearchGridRow(string selectedItem, ref Grid generatedFields)`:
При създаване на нов ред в грид с контроли за параметър, по който да се филтрира дадена информация подаваме избрания елемент и грида, в който искаме да генерираме дадения ред.
- `getFieldsForComboBox(string table)`: методът служи за зареждане на параметри, по които да филтрираме. Приема низова стойност, която отговаря на името на класа, който ще филтрираме (Трябва да е спазена конвенцията класовете да са с име `<>Model` и да са в namespace `Models`).
- `search(string selectedObject, ref Grid gridOut)` - служи за метод при натискане на бутон за търсене, подава се избран обект за търсене (име на клас) и грид, в който искаме да генерираме изходни данни.
- `clearGrid(Grid gridOutput)` - служи за изчистване на подаден грид (използва се за почистване на грид за изходни данни). Използва се за бутон `clear`.

- `clearGrid(Grid gridOutput, Grid gridGeneratedFields)` - служи за почистване на грид за входни и изходни данни (Използва се за бутон Return).

С комбинацията от тези полета може да се изгради лесно отделно View, което да имплементира същата логика, като тази в системата.