МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
федеральное государственное автономное образовательное учреждение высшего образования
«Санкт–Петербургский государственный университет
аэрокосмического приборостроения»

ФАКУЛЬТЕТ СРЕДНЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
РУКОВОДИТЕЛЬ _____

преподаватель		
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ ПО ЛАБОРАТОРНЫМ РАБОТАМ

по дисциплине МДК 01.01 часть 3

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	С326		Э.С. Тигранян
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

Оглавление

Лабораторная работа №1

Тема: Создание приложения «Микропроводник».

Цель работы: получение практических навыков при работе с пространством имен System.IO.

Задание

Разработать приложение «Микропроводник»,

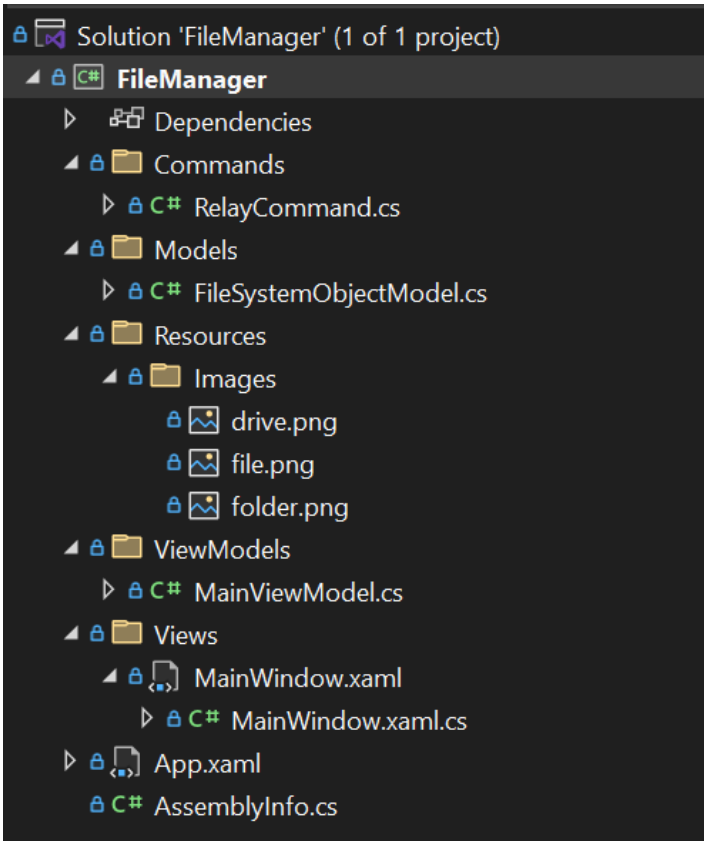
Используя дополнительные компоненты,

- для выделенного диска необходимо выводить сведения: объем диска, свободное пространство;
- для выделенного каталога: полное название каталога, время создания каталога, корневой каталог.

При выделении файла в списке должно запускаться соответствующее приложение. Сохранить в отдельный текстовый файл имена файлов, которые открывались за последние 10 секунд работы приложения.

Код программы

Программа реализованная по паттерну MVVM, есть разделение на View – MainWindow, ViewModel – MainViewModel, Model – FileSystemObjectModel.



Структура проекта

Код файла RelayCommand.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
using FileManager.Commands;
using FileManager.Models;

namespace FileManager.Commands
{
    public class RelayCommand : ICommand
    {
        private readonly Action<object> _execute;
        private readonly Func<object, bool> _canExecute;

        public RelayCommand(Action<object> execute, Func<object, bool> canExecute =
null)
        {
            _execute = execute ?? throw new ArgumentNullException(nameof(execute));
            _canExecute = canExecute;
        }
        public bool CanExecute(object parameter)
        {
            return _canExecute == null || _canExecute(parameter);
```

```
        }

        public void Execute(object parameter)
        {
            _execute(parameter);
        }

        public event EventHandler CanExecuteChanged
        {
            add { CommandManager.RequerySuggested += value; }
            remove { CommandManager.RequerySuggested -= value; }
        }
    }
}
```

Назначение файла: Модели файловой системы (диск, каталог, файл) для приложения «Микропроводник».

Код файла FileSystemObjectModel.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Automation;

namespace FileManager.Models
{
    public enum E_ObjectType
    {
        File,
        Directory,
        Drive
    }
    public abstract class FileSystemObjectModel
    {
        public FileSystemObjectModel()
        {
            Children = new ObservableCollection<FileSystemObjectModel>();
        }
             public override string ToString() => Name;
        public ObservableCollection<FileSystemObjectModel> Children { get; set; }
             public abstract E_ObjectType ObjectType { get; }

             private string _name;
             public string Name
             {
                    get => _name;
                    set { _name = value; }
             }

             private string _fullpath;
             public string FullPath
             {
                    get => _fullpath;
                    set { _fullpath = value; }
             }
```

```
        private DateTime _createDate;
        public DateTime CreateDate
    {
            get => _createDate;
            set { _createDate = value; }
        }
}
  public class File : FileSystemObjectModel
  {
        public override E_ObjectType ObjectType => E_ObjectType.File;

    private long _size; // Size in bytes

        public long Size
        {
            get => _size;
            set { _size = value; }
        }
  }
public class Directory : FileSystemObjectModel
{
    public override E_ObjectType ObjectType => E_ObjectType.Directory;

        private int _itemAmount;

        public int ItemAmount
        {
            get => _itemAmount;
            set { _itemAmount = value; }
        }

        private long _size; // Size in bytes

    public long Size
    {
        get => _size;
        set { _size = value; }
    }
}
public class Drive : FileSystemObjectModel
{
        public Drive(String Letter)
        {
        DriveInfo[] drives = DriveInfo.GetDrives();
        foreach (var d in drives)
            {
                if (d.Name[0].ToString() == Letter)
                {
                    this.SpaceLeft = d.AvailableFreeSpace;
                    this.SpaceOverall = d.TotalSize;
                    this.FullPath = d.Name;
                    break;
                }
            }
    }
        public Drive() { }
    public override E_ObjectType ObjectType => E_ObjectType.Drive;

    private long _spaceLeft; // Size in bytes

    public long SpaceLeft
    {
        get => _spaceLeft;
        set { _spaceLeft = value; }
    }
```

```
            private long _spaceOverall;

            public long SpaceOverall
            {
                  get => _spaceOverall;
                  set { _spaceOverall = value; }
            }

      }
}
```

Код файла MainViewModel.cs

```
using FileManager.Commands;
using FileManager.Models;
using System;
using System.Collections;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Timers;
using System.Windows;
using System.Windows.Input;
using static System.Net.Mime.MediaTypeNames;

namespace FileManager.ViewModels
{
    public class MainViewModel : INotifyPropertyChanged
    {

        private Hashtable openFiles_openFileTime = new Hashtable();

        private static System.Timers.Timer aTimer;

        string filePath = "latelyOpenedPrograms.txt";
        public MainViewModel()
        {
            RootItems = new ObservableCollection<Drive>();
            LoadDrivesCommand = new RelayCommand(LoadDrives);
            LoadChildrenCommand = new RelayCommand(LoadChildren);
            OpenFileCommand = new RelayCommand(OpenFile);
            SelectItemCommand = new RelayCommand(SelectItem);
            MouseDoubleClickCommand = new RelayCommand(MouseDoubleClick);
            LoadDrives(null);

            SetTimer();
        }
        private void SetTimer()
        {
            aTimer = new System.Timers.Timer(1000);
            aTimer.Elapsed += OnTimedEvent;
            aTimer.AutoReset = true;
            aTimer.Enabled = true;
        }
        private void OnTimedEvent(Object source, ElapsedEventArgs e)
        {
            Console.WriteLine("The Elapsed event was raised at {0:HH:mm:ss.fff}",
                              e.SignalTime);
            var keysToRemove = new List<object>();
```

```
            foreach (DictionaryEntry entry in openFiles_openFileTime)
            {
                if (entry.Value is DateTime value)
                {
                    if (DateTime.Now - value > TimeSpan.FromSeconds(10)) // should
be 10 second
                    {
                        keysToRemove.Add(entry.Key);
                    }
                }
            }

            // Remove all collected keys
            foreach (var key in keysToRemove)
            {
                openFiles_openFileTime.Remove(key);
            }

            using (StreamWriter writer = new StreamWriter(filePath))
            {
                foreach (DictionaryEntry entry in openFiles_openFileTime)
                {
                    writer.WriteLine($"{entry.Key}={entry.Value}");
                }
            }
        }

        private ObservableCollection<Drive> _rootItems;
        private FileSystemObjectModel _selectedObject;

        public ObservableCollection<Drive> RootItems
        {
            get => _rootItems;
            set { _rootItems = value;
                OnPropertyChanged(); }
        }


        private String _selectedFilePreviewText;
        public String SelectedFilePreviewText
        {
            get { return _selectedFilePreviewText; }
            set
            {
                _selectedFilePreviewText = value;
                OnPropertyChanged();
            }
        }

        public FileSystemObjectModel SelectedObject
        {
            get => _selectedObject;
            set { _selectedObject = value;
                OnPropertyChanged(); }
        }

        private String _driveInfo;
        public String SelectedDriveInfo
        {
            get => _driveInfo;
            set { _driveInfo = value; OnPropertyChanged(); }
        }
        private String _directiryInfo;
        public String SelectedDirectoryInfo
        {
```

```
            get => _directiryInfo;
            set { _directiryInfo = value; OnPropertyChanged(); }
        }

        public ICommand LoadDrivesCommand { get; }
        public ICommand LoadChildrenCommand { get; }
        public ICommand OpenFileCommand { get; }
        public ICommand SelectItemCommand { get; }
        public ICommand MouseDoubleClickCommand { get; }
        private void InfoUpdate(object parameter)
        {
            String RootDirectoryLetter = SelectedObject.FullPath[0].ToString();
            Models.Drive RootDirectory = new Models.Drive(RootDirectoryLetter);
            SelectedDriveInfo =
                    "Информация о диске " + RootDirectory.FullPath + "\n" +
                "Объем диска: " + RootDirectory.SpaceOverall / 1024 / 1024 / 1024 +
" Гб \n" +
                "Свободное пространство: " + RootDirectory.SpaceLeft / 1024 / 1024 /
1024 + " Гб \n"
                ;

            if (SelectedObject is Models.Directory directory)
            {
                SelectedDirectoryInfo =
                    "Информация о директории\n" +
                    "Директория: " + directory.FullPath + "\n" +
                    "Время создания: " + Convert.ToDateTime(directory.CreateDate) +
"\n" +
                    "Корневой каталог: " + RootDirectory.FullPath + "\n"
                    ;
            }
            else if (SelectedObject is Models.File file)
            {
                SelectedDirectoryInfo =
                    "Информация о файле\n" +
                    "Имя файла: " + file.FullPath + "\n"
                    ;
            }
            else if (SelectedObject is Models.Drive drive)
            {
                SelectedDirectoryInfo =
                    "Информация о диске\n" +
                    "Объем диска: " + drive.SpaceOverall / 1024 / 1024 / 1024 + " Гб
\n" +
                    "Свободное пространство: " + drive.SpaceLeft / 1024 / 1024 /
1024 + " Гб \n"
                    ;
            }
        }
        private void PreviewText(object parameter)
        {
            if (SelectedObject is Models.File file)
            {
                try
                {
                    if (Path.GetExtension(file.FullPath).ToLower() == ".txt")
                    {
                        SelectedFilePreviewText =
System.IO.File.ReadAllText(file.FullPath);
                    }
                    else
                    {
                        byte[] bytes = System.IO.File.ReadAllBytes(file.FullPath);
                        var bytesFirst500 = bytes.Take(500);
```

```
                        SelectedFilePreviewText =
BitConverter.ToString(bytesFirst500.ToArray()).Replace("-"," ");
                    }
                }
                catch (Exception ex) {
                    SelectedFilePreviewText = ($"Access denied: {file.FullPath}");
                }
            }
            else {
                SelectedFilePreviewText = "Select File";
            }
        }
        private void MouseDoubleClick(object parameter)
        {
            if (SelectedObject is Models.File file) {
                try {
                    ProcessStartInfo startInfo = new
ProcessStartInfo(file.FullPath);
                    startInfo.UseShellExecute = true;
                    System.Diagnostics.Process.Start(startInfo);
                    openFiles_openFileTime.Add(file.Name, DateTime.Now);
                }
                catch {

                }
            }
        }
        private void SelectItem(object parameter)
        {
            InfoUpdate(null);
            PreviewText(null);
        }
        private String GetRootDirectory(String path) => path[0].ToString();

        private void LoadDrives(object parameter)
        {
            try
            {
                DriveInfo[] drives = DriveInfo.GetDrives();
                foreach (DriveInfo drive in drives)
                {
                    if (drive.IsReady)
                    {
                        RootItems.Add(
                        new Drive
                        {
                            Name = drive.Name,
                            FullPath = drive.Name,
                            SpaceOverall = drive.TotalSize,
                            SpaceLeft = drive.AvailableFreeSpace
                        });
                    }
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Ошибка загрузки дисков: {ex.Message}");
            }
        }

        private void LoadChildren(object parameter)
        {
            if (parameter is FileSystemObjectModel item &&
                System.IO.Directory.Exists(item.FullPath))
            {
```

```
                try
                {
                    // Load directories
                    string[] dirs =
System.IO.Directory.GetDirectories(item.FullPath);
                    foreach (string dirPath in dirs)
                    {
                        var dirInfo = new System.IO.DirectoryInfo(dirPath);
                        item.Children.Add(new Models.Directory
                        {
                            Name = dirInfo.Name,
                            FullPath = dirPath,
                            CreateDate = dirInfo.CreationTime,
                        });
                    }

                    // Load files
                    string[] files = System.IO.Directory.GetFiles(item.FullPath);
                    foreach (string filePath in files)
                    {
                        var fileInfo = new System.IO.FileInfo(filePath);
                        item.Children.Add(new Models.File
                        {
                            Name = fileInfo.Name,
                            FullPath = filePath,
                            CreateDate = fileInfo.CreationTime,
                            Size = fileInfo.Length
                        });
                    }
                }
                catch (UnauthorizedAccessException)
                {
                    Console.WriteLine($"Access denied: {item.FullPath}");
                }
                catch (Exception ex)
                {
                    Console.WriteLine($"Error loading children: {ex.Message}");
                }
            }
        }

        private void OpenFile(object parameter) =>
            // Заглушка для реализации открытия файла
            Console.WriteLine("OpenFile command executed");

        public event PropertyChangedEventHandler PropertyChanged;
        protected virtual void OnPropertyChanged([CallerMemberName] string
propertyName = null)
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Назначение файла: XAML-разметка главного окна приложения.

Код файла MainWindow.xaml

```
<Window x:Class="FileManager.Views.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

```
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:FileManager.Views"
        xmlns:vm="clr-namespace:FileManager.ViewModels"
        xmlns:models="clr-namespace:FileManager.Models"
        mc:Ignorable="d"
        Title="MainWindow" Height="600" Width="900"
        ResizeMode="NoResize">

    <Window.DataContext>
        <vm:MainViewModel/>
    </Window.DataContext>

    <Window.Resources>
        <HierarchicalDataTemplate DataType="{x:Type models:File}"
                                  ItemsSource="{Binding Children}">
            <StackPanel Orientation="Horizontal">
                <Image Source="pack://application:,,,/Resources/Images/file.png"
Width="16" Height="16" Margin="0,0,5,0"/>
                <TextBlock Text="{Binding Name}"/>
            </StackPanel>
        </HierarchicalDataTemplate>

        <HierarchicalDataTemplate DataType="{x:Type models:Directory}"
                                  ItemsSource="{Binding Children}">
            <StackPanel Orientation="Horizontal">
                <Image Source="pack://application:,,,/Resources/Images/folder.png"
Width="16" Height="16" Margin="0,0,5,0"/>
                <TextBlock Text="{Binding Name}"/>
            </StackPanel>
        </HierarchicalDataTemplate>

        <HierarchicalDataTemplate DataType="{x:Type models:Drive}"
                                  ItemsSource="{Binding Children}">
            <StackPanel Orientation="Horizontal">
                <Image Source="pack://application:,,,/Resources/Images/drive.png"
Width="16" Height="16" Margin="0,0,5,0"/>
                <TextBlock Text="{Binding Name}"/>
                <TextBlock Text="{Binding SpaceLeft, StringFormat=' ({0} free)'}"
                           Foreground="Gray" FontStyle="Italic"/>
            </StackPanel>
        </HierarchicalDataTemplate>
    </Window.Resources>

    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>

        <TreeView Grid.Column="0"
                  Background="DarkSlateGray"
                ItemsSource="{Binding RootItems}"
                SelectedItemChanged="TreeView_SelectedItemChanged">
            <TreeView.ItemContainerStyle>
                <Style TargetType="{x:Type TreeViewItem}">
                    <Setter Property="Foreground" Value="White"></Setter>
                    <Setter Property="Margin" Value="0,5,0,0"></Setter>
                    <Setter Property="IsExpanded" Value="{Binding IsExpanded,
Mode=TwoWay}"/>
                    <Setter Property="IsSelected" Value="{Binding IsSelected,
Mode=TwoWay}"/>
                    <EventSetter Event="Expanded" Handler="TreeViewItem_Expanded"/>
                    <EventSetter Event="MouseDoubleClick"
Handler="TreeViewItem_MouseDoubleClick"/>
                </Style>
```

```
            </TreeView.ItemContainerStyle>
        </TreeView>
        <Grid Grid.Column="1">
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="auto"></RowDefinition>
                <RowDefinition Height="auto"></RowDefinition>
                <RowDefinition Height="*"></RowDefinition>
            </Grid.RowDefinitions>
            <TextBlock Grid.Row="0" Grid.ColumnSpan="2" Height="auto"
Text="Информация" FontWeight="Bold" FontSize="20"
HorizontalAlignment="Center"></TextBlock>
            <TextBlock Grid.Row="1" Grid.Column="0"
                       Text="{Binding SelectedDriveInfo}"
                       TextWrapping="Wrap"
                       FontSize="13"
                       Padding="10"
                       />
            <TextBlock Grid.Row="1" Grid.Column="1"
                       Text="{Binding SelectedDirectoryInfo}"
                       TextWrapping="Wrap"
                       FontSize="13"
                       Padding="10"
                       />
            <TextBlock Grid.Row="2" Grid.ColumnSpan="2"
                       Text="{Binding SelectedFilePreviewText}"
                       TextWrapping="Wrap"
                       FontSize="13"
                       Padding="10"
                       />
        </Grid>


         </Grid>
</Window>
```

Назначение файла: Код-бихайнд главного окна. Обработка событий интерфейса.

Код файла MainWindow.xaml.cs

```
using System.Windows;
using System.Windows.Controls;
using FileManager.Models;
using FileManager.ViewModels;

namespace FileManager.Views
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
        private void TreeView_SelectedItemChanged(object sender,
RoutedPropertyChangedEventArgs<object> e)
        {
```

```
            if (DataContext is MainViewModel viewModel && e.NewValue is
FileSystemObjectModel selectedItem)
            {
                viewModel.SelectedObject = selectedItem;
                viewModel.SelectItemCommand.Execute(null);
            }
        }
        private void TreeViewItem_Expanded(object sender, RoutedEventArgs e)
        {
            var treeViewItem = e.OriginalSource as TreeViewItem;
            if (treeViewItem?.DataContext is FileSystemObjectModel item)
            {
                var viewModel = DataContext as MainViewModel;
                if (item.Children.Count == 0)
                {
                    viewModel?.LoadChildrenCommand.Execute(item);
                }
            }
        }
        private void TreeViewItem_MouseDoubleClick(object sender,
System.Windows.Input.MouseButtonEventArgs e)
        {
            if(DataContext is MainViewModel viewModel)
            {
                viewModel.MouseDoubleClickCommand.Execute(null);
            }
        }
    }
}
```

Описание кода

Взаимодействие файлов и слоёв приложения

1. Запуск приложения и установка связей MVVM.

   При открытии окна MainWindow автоматически создаётся MainViewModel (через Window.DataContext). С этого момента интерфейс (TreeView и текстовые блоки справа) работает с данными через биндинги к свойствам ViewModel (например, RootItems, SelectedDriveInfo, SelectedDirectoryInfo, SelectedFilePreviewText).

2. Первичная загрузка дисков.

   В конструкторе MainViewModel создаётся коллекция RootItems и выполняется загрузка доступных дисков через DriveInfo.GetDrives(). Для каждого готового диска создаётся модель Drive (с именем/путём и размерами свободного/общего места), и эта модель добавляется в RootItems. TreeView отображает эти элементы, потому что привязан к RootItems.

3. Раскрытие узла в TreeView (загрузка дочерних элементов по требованию).

   Когда пользователь раскрывает диск или папку, в View срабатывает событие Expanded. Code-behind проверяет: если у выбранного элемента Children пустой, вызывается команда LoadChildrenCommand во ViewModel и передаётся текущий элемент. ViewModel через System.IO.Directory.GetDirectories() и GetFiles() подгружает подкаталоги и файлы и добавляет их в Children. После этого TreeView визуально отображает новые дочерние элементы.
4. Выбор элемента (диск/папка/файл) и обновление информации справа.

   При смене выбранного элемента в TreeView срабатывает SelectedItemChanged. View устанавливает SelectedObject во ViewModel и запускает SelectItemCommand. Внутри ViewModel выполняются два действия:

- InfoUpdate: формирует текст с информацией о диске/директории/файле (объём/свободно; полный путь; дата создания; корневой каталог и т.д.) и записывает в свойства SelectedDriveInfo и SelectedDirectoryInfo;
- PreviewText: если выбран файл — строит предпросмотр (см. следующий шаг).

   Изменение этих свойств приводит к автоматическому обновлению текста в правой части окна из-за биндингов.

5. Предпросмотр файла.

   Если выбран объект типа File, ViewModel проверяет расширение:

- для .txt читает содержимое как текст и выводит полностью;
- для не-текстовых файлов читает байты и отображает первые ~500 байт в шестнадцатеричном виде (через BitConverter).

   Если доступ запрещён (например, нет прав), в предпросмотр выводится сообщение об ошибке (“Access denied …”).

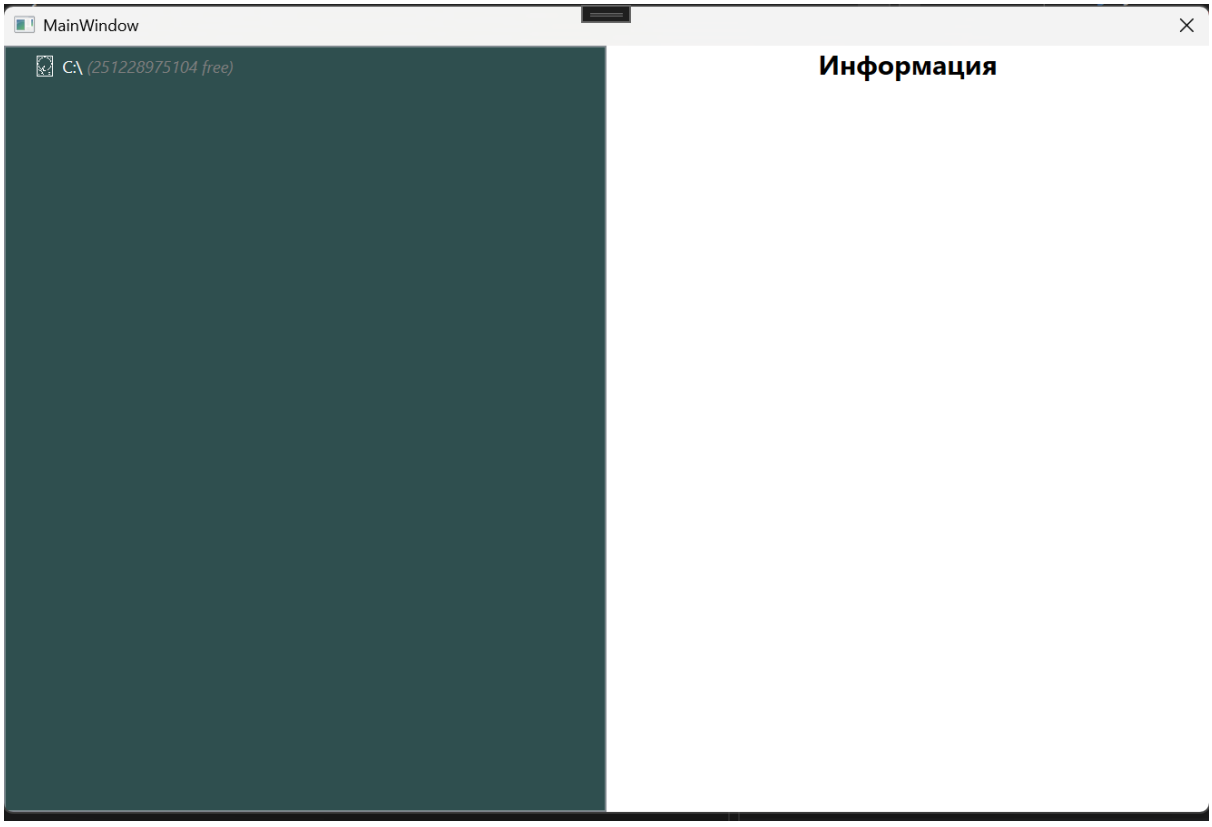
6. Двойной клик по файлу (запуск приложения ОС).

   Если пользователь делает двойной клик по элементу TreeView, View

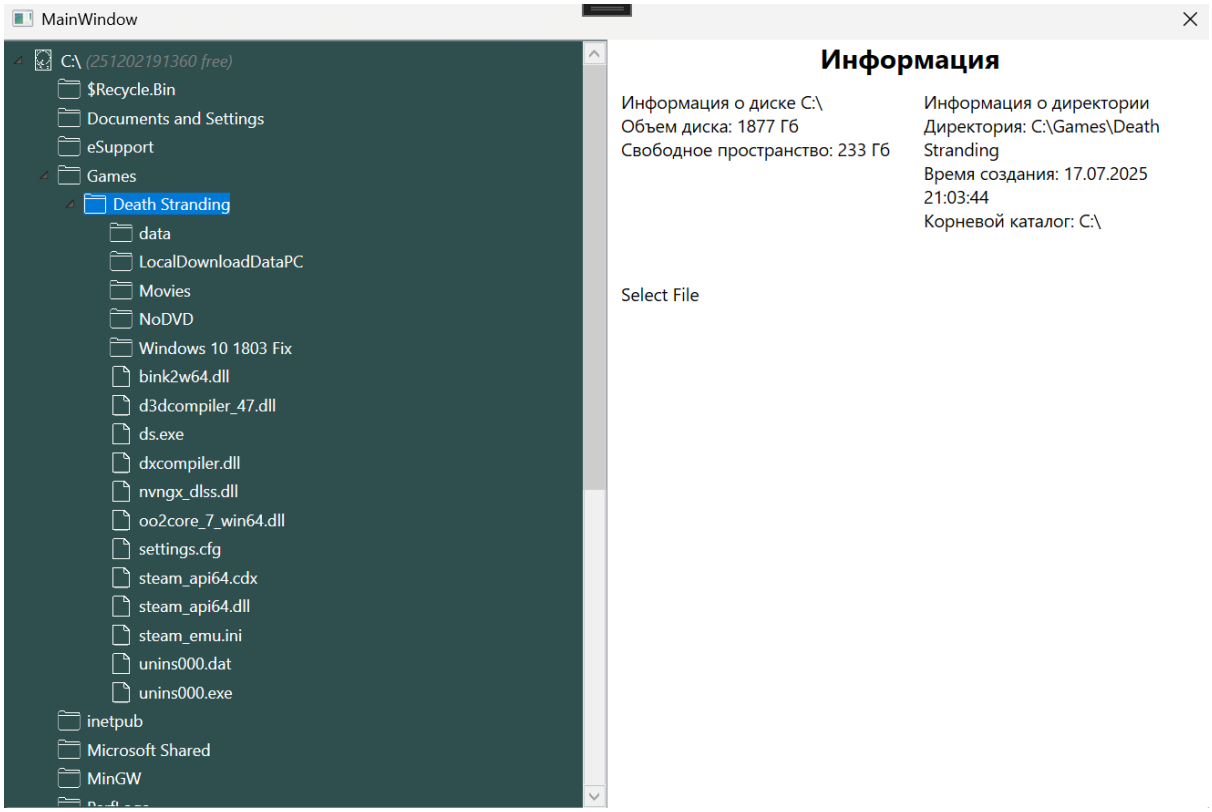
вызывает MouseDoubleClickCommand. Если выбранный объект — файл, ViewModel запускает его через Process.Start с UseShellExecute=true, то есть открытие выполняется средствами операционной системы “по умолчанию”.

7. Фиксация “открытых за последние 10 секунд”.
   При каждом успешном запуске файла ViewModel добавляет запись в таблицу/коллекцию вида “имя файла - время открытия”. Параллельно работает таймер (период 1 секунда): он удаляет из коллекции записи старше 10 секунд и перезаписывает текстовый файл latelyOpenedPrograms.txt, сохраняя актуальный список открытых файлов за последние 10 секунд.

Результат

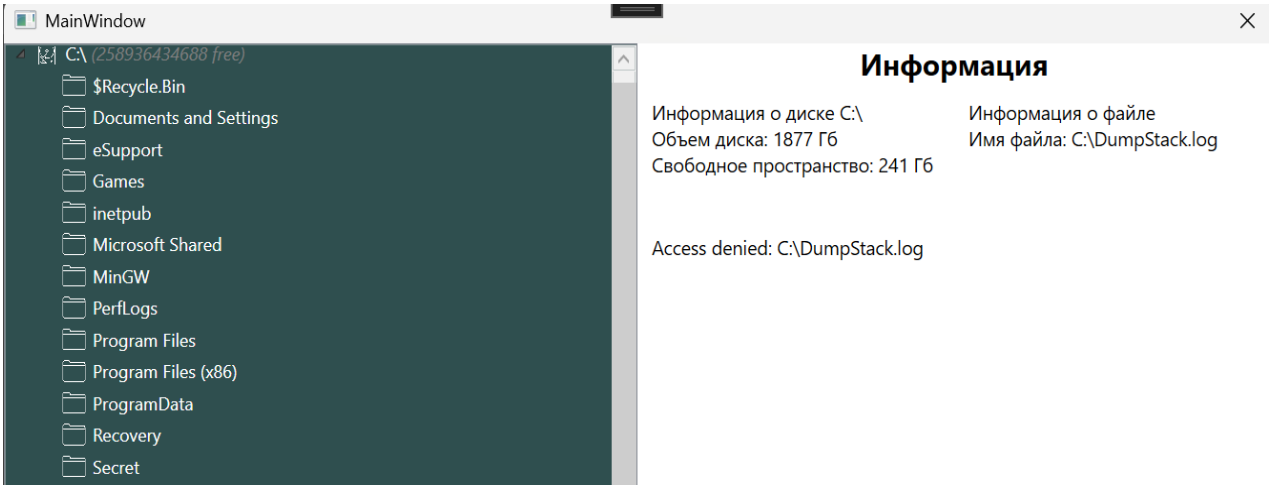


Изначальный вид программы



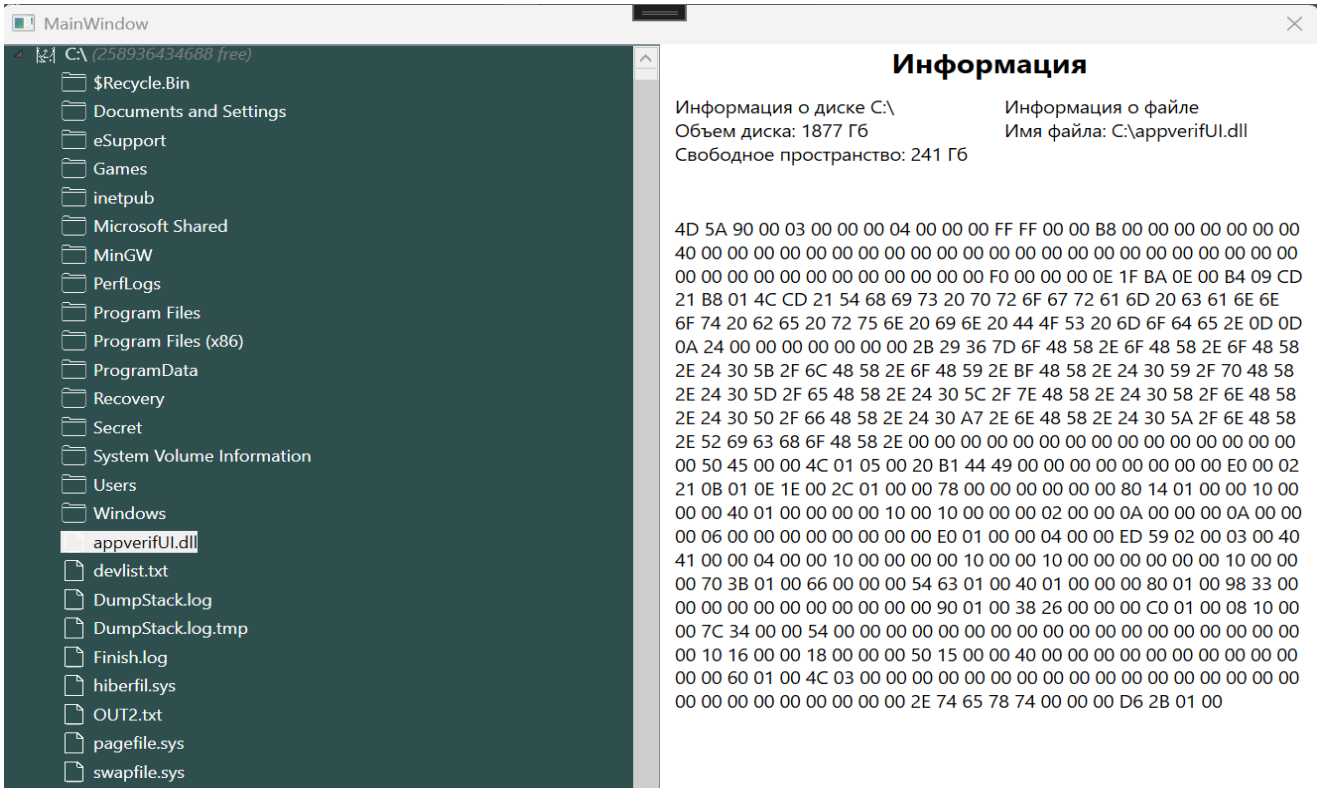
Вид раскрытого дерева

В программе реализован предпросмотр. При веделении текстового файла идёт показ содержимого в виде текста, при веделении bin файла идёт показ содержимого в шестнадцеричном виде. При двойном нажатии на файл через операционную среду этот файл запускается. Также при выделении файла, директории, диска на правом экране выводится соответствующая информация.

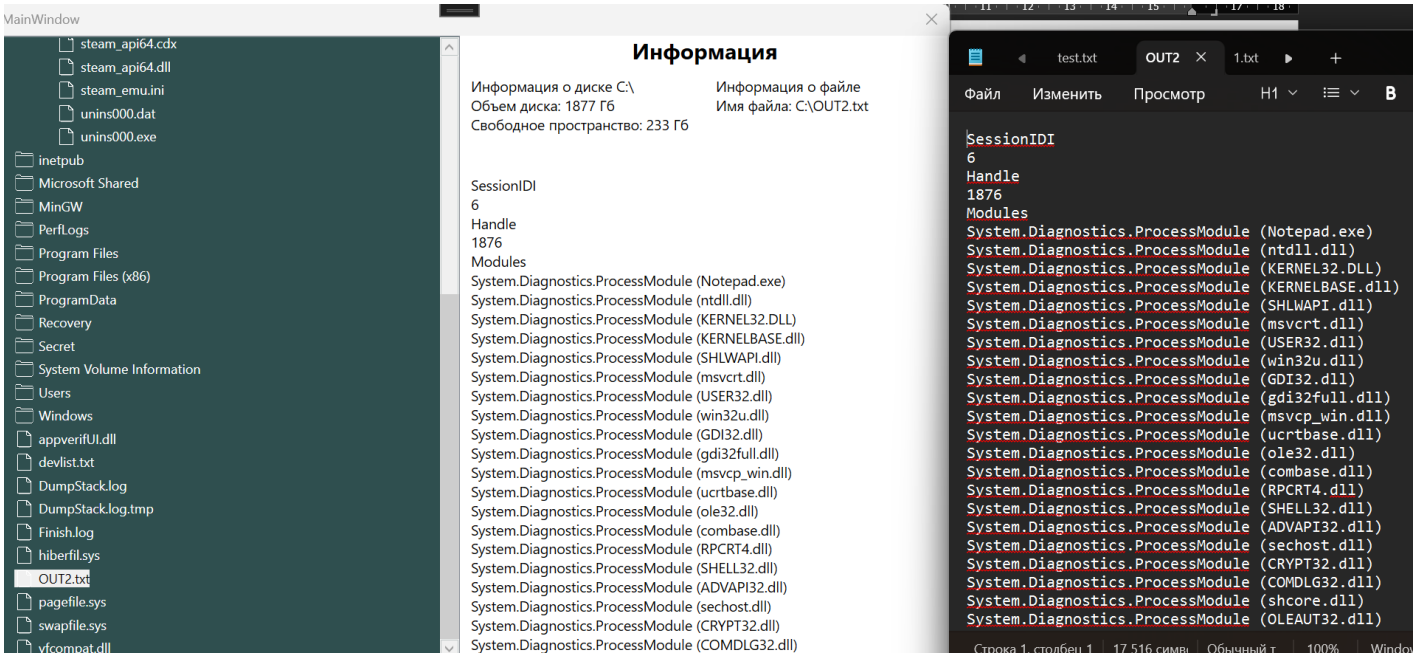


Отказ доступа предпросмотра файла

На предпросмотр бинарного файла выделяется 500 символов

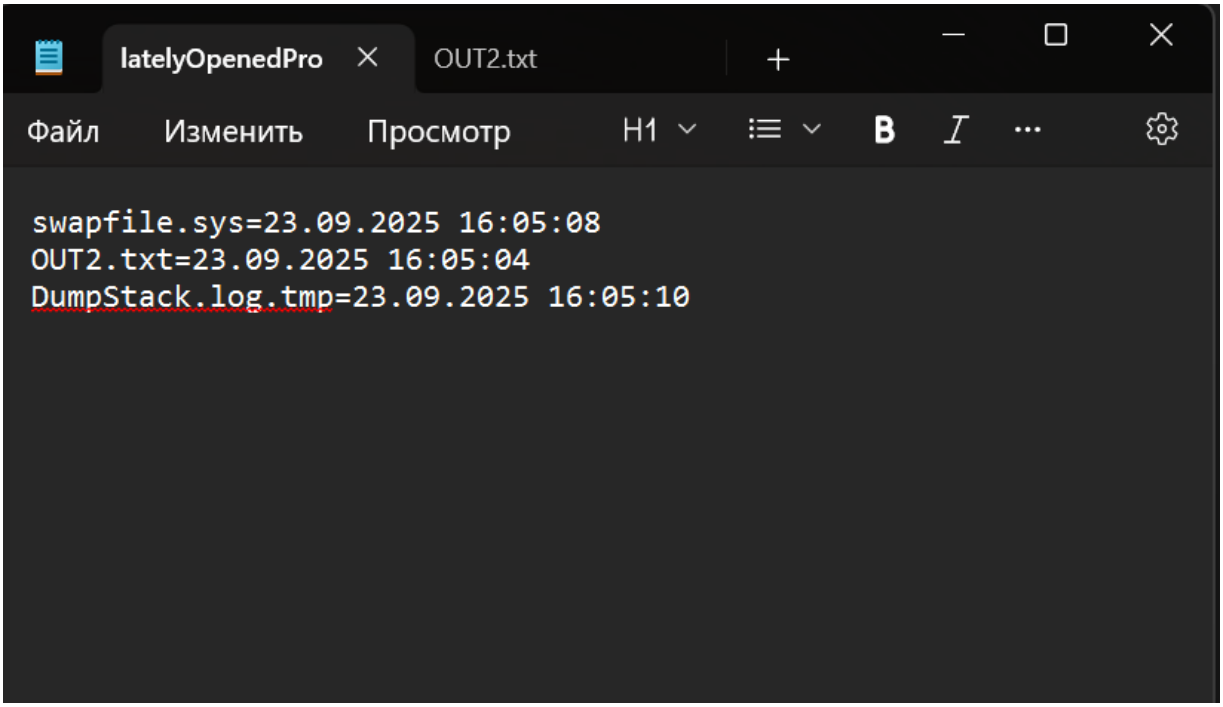


предпросмотр бинарного файла

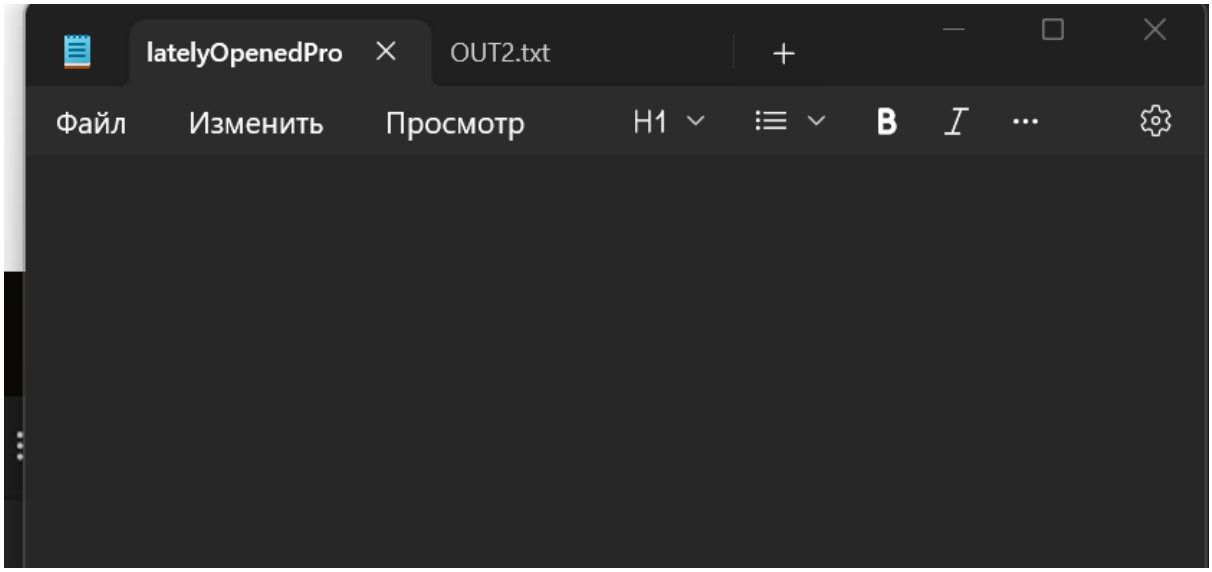


Результат открытия и выделения файла

В программе реализован предпросмотр. При веделении текстового файла идёт показ содержимого в виде текста, при веделении bin файла идёт показ содержимого в шестнадцеричном виде. При двойном нажатии на файл через операционную среду этот файл запускается.



Результат открытия программ за 10 секунд



Результат после десяти секунд

Лабораторная работа №2

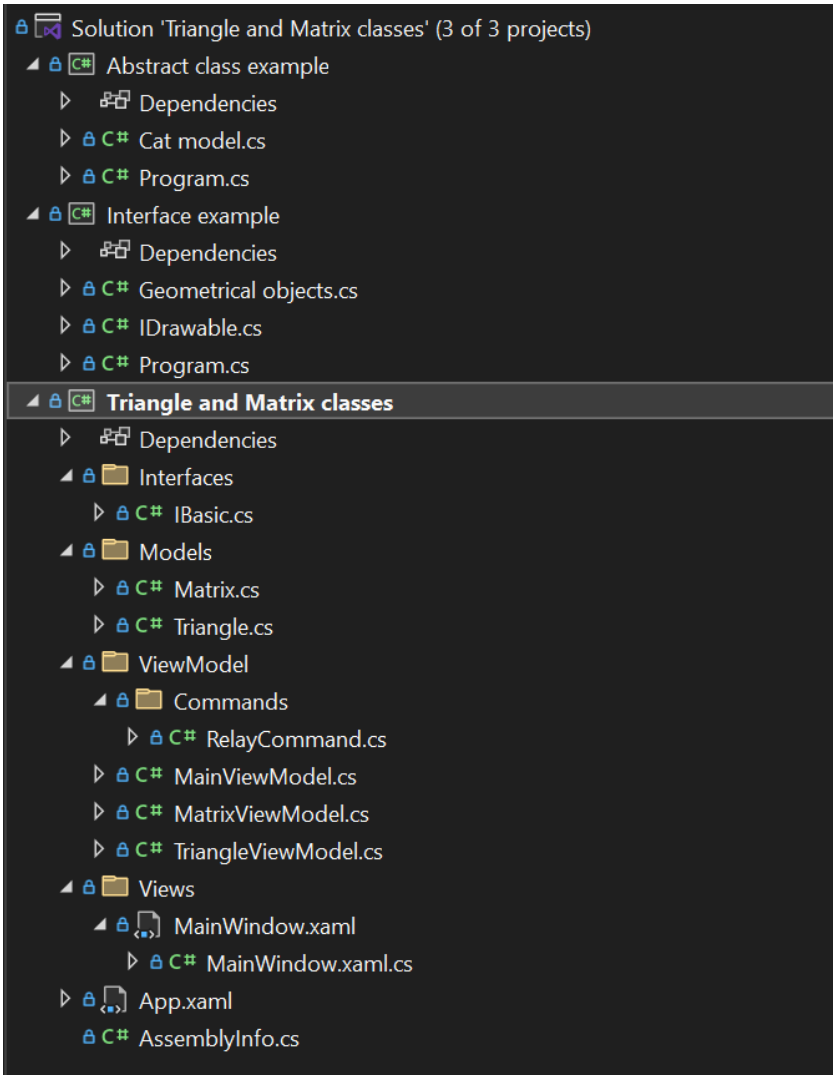
Тема: Создание собственных классов в C#. Работа с абстрактными классами и интерфейсами.

Цель работы: получение практических навыков при создании и наследовании классов в C#, при работе с абстрактными классами и интерфейсами.

Выполнение работы:

Задание 1, 2, 3

Создание классов по вариантам.



Структура проекта

Папка Interfaces

Файл IBasic.cs

Содержит интерфейсы, которые задают общий контракт для моделей и ViewModel. Определяет методы вычислений, которые должны быть реализованы.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;

namespace Triangle_and_Matrix_classes.Interfaces
{
    public interface Matrix
    {
        void Draw();
    }
    public interface IGeometrical
    {
        void GetPerimeter();
        void GetArea();
    }
    public interface IViewModelBase
    {
        public void Calculate(object parameter);
    }
}
```

Папка Models

Папка ViewModel

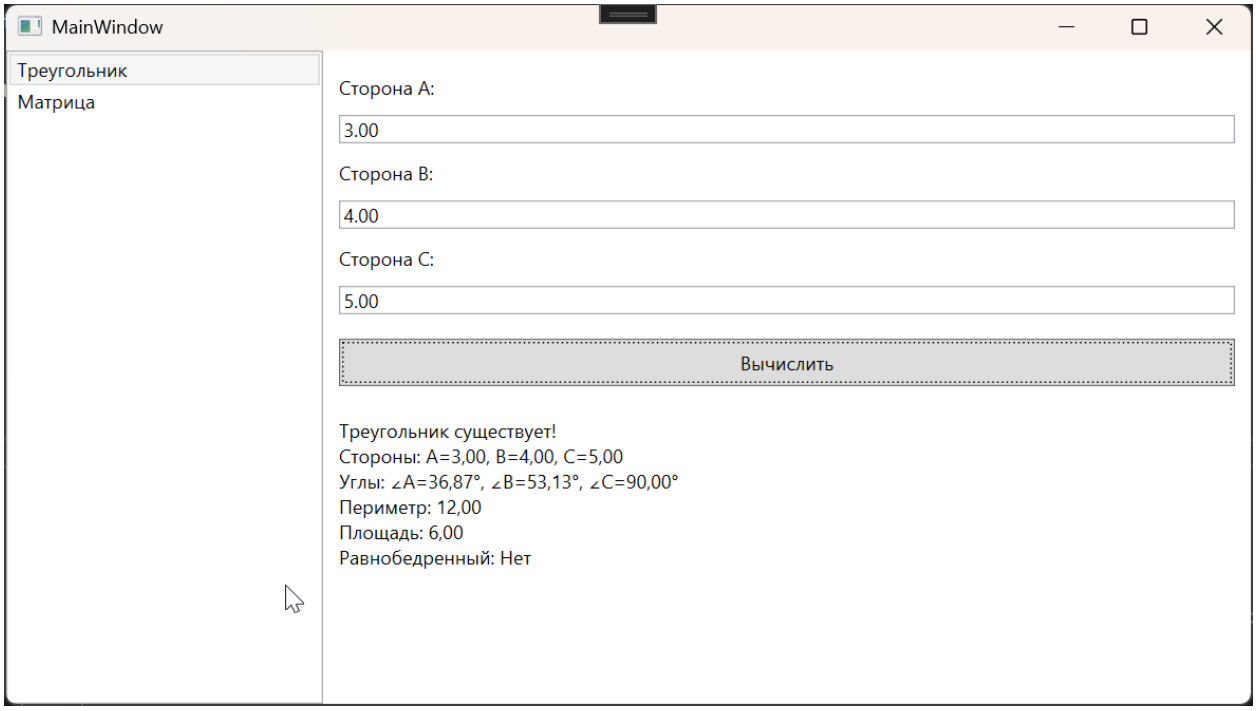
Папка Views

Описание код

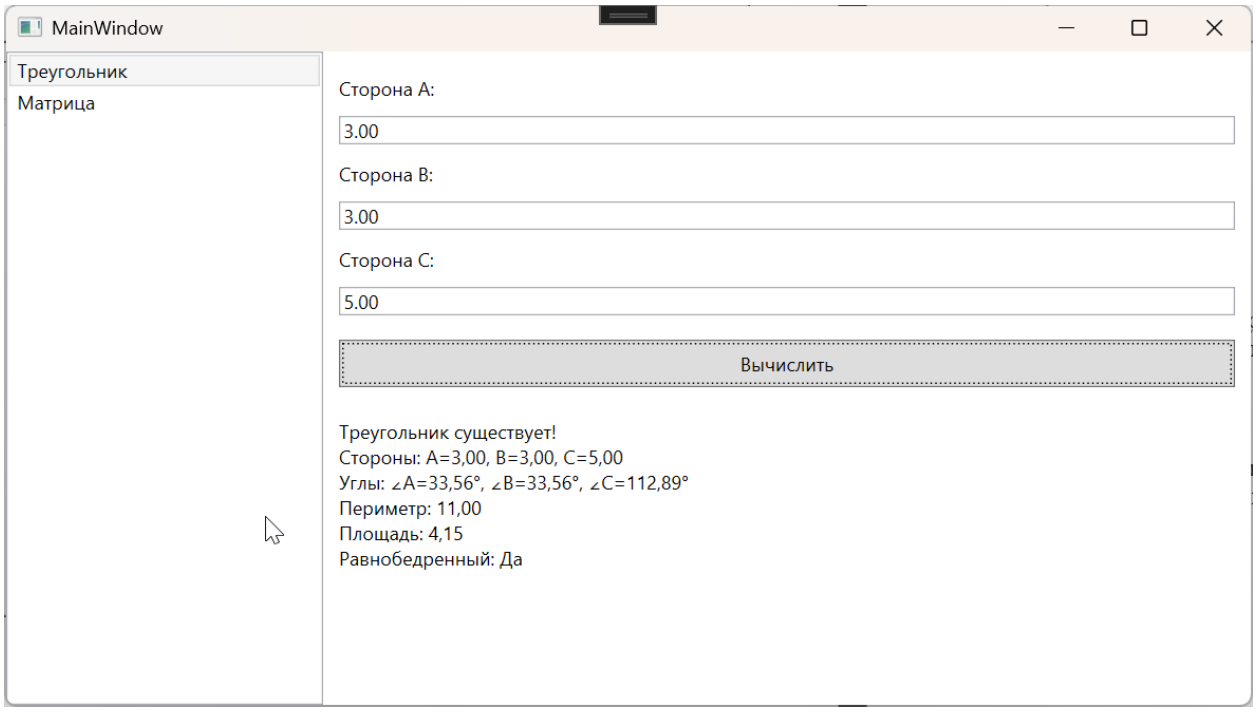
Результат

Главный экран приложения

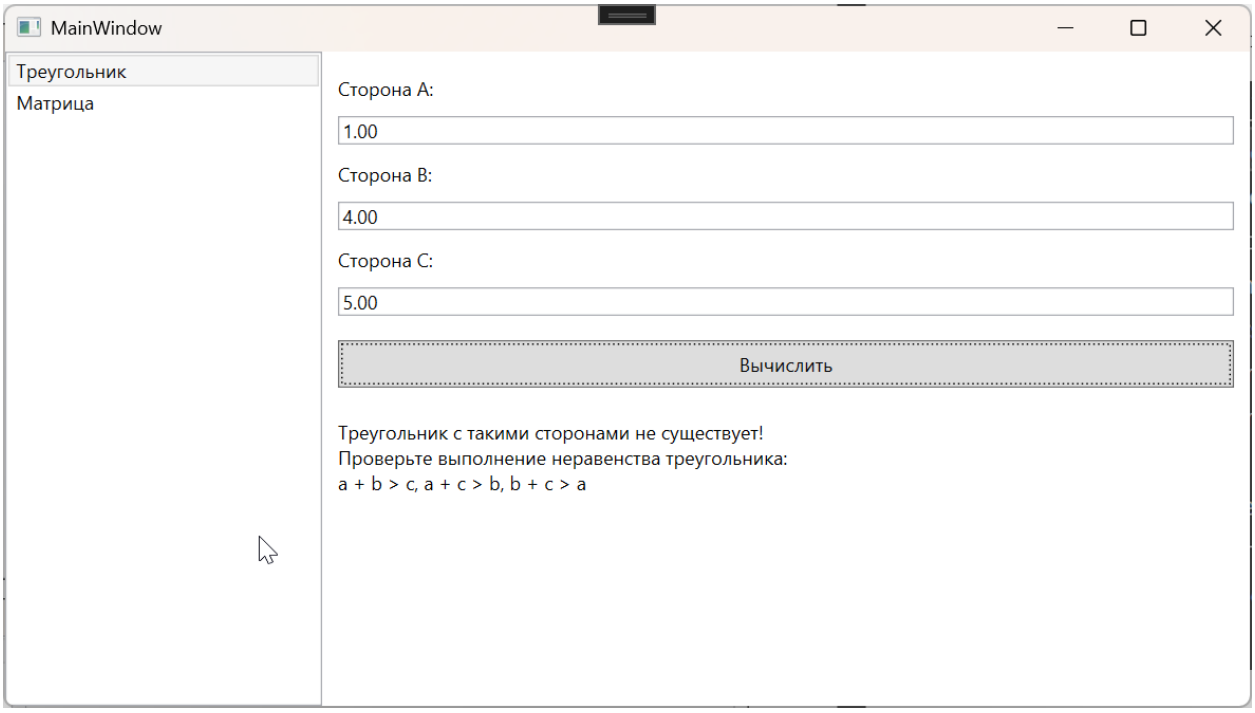
Задание с треугольником



Пример задания данных египетского треугольника

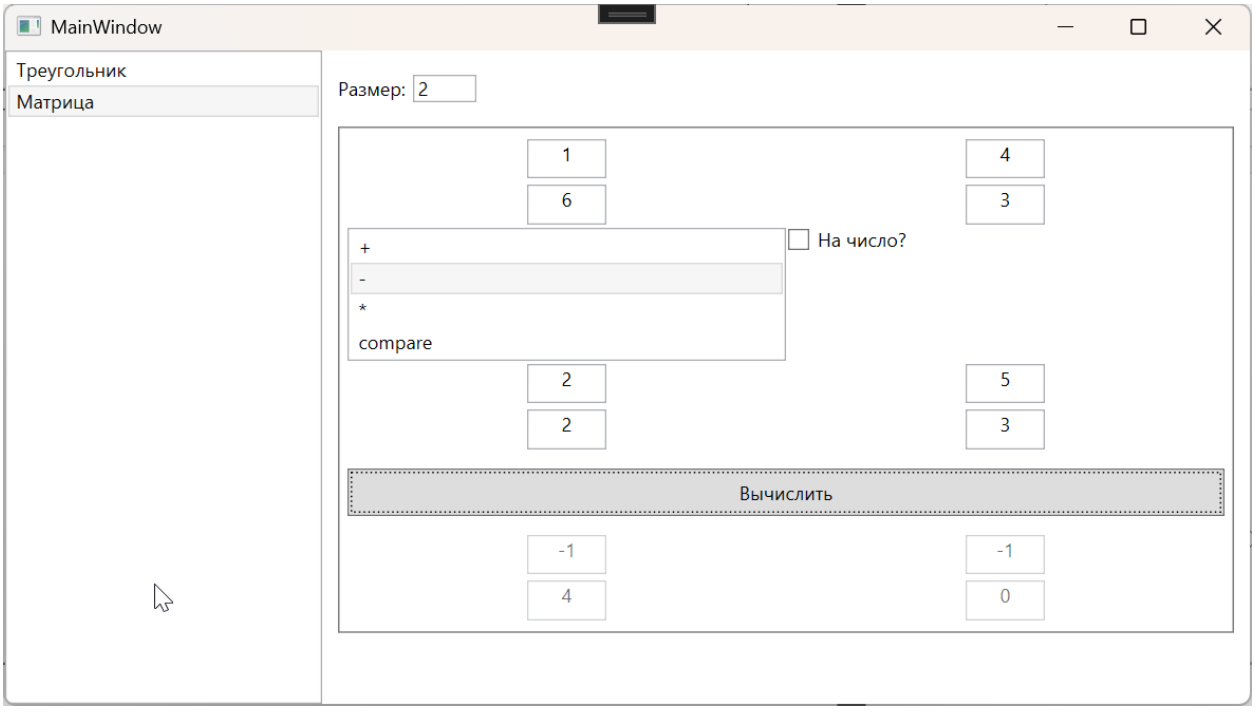


Пример введения данных равнобедренного треугольника

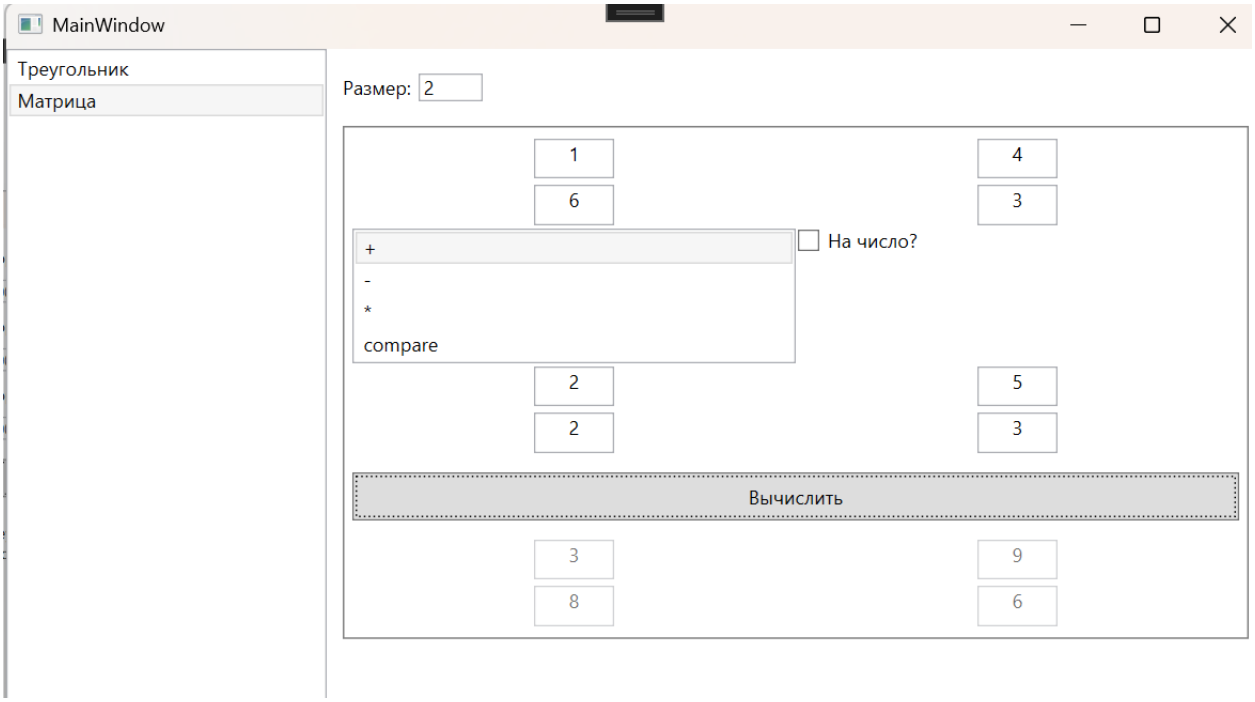


Пример введения данных несуществующего треугольника

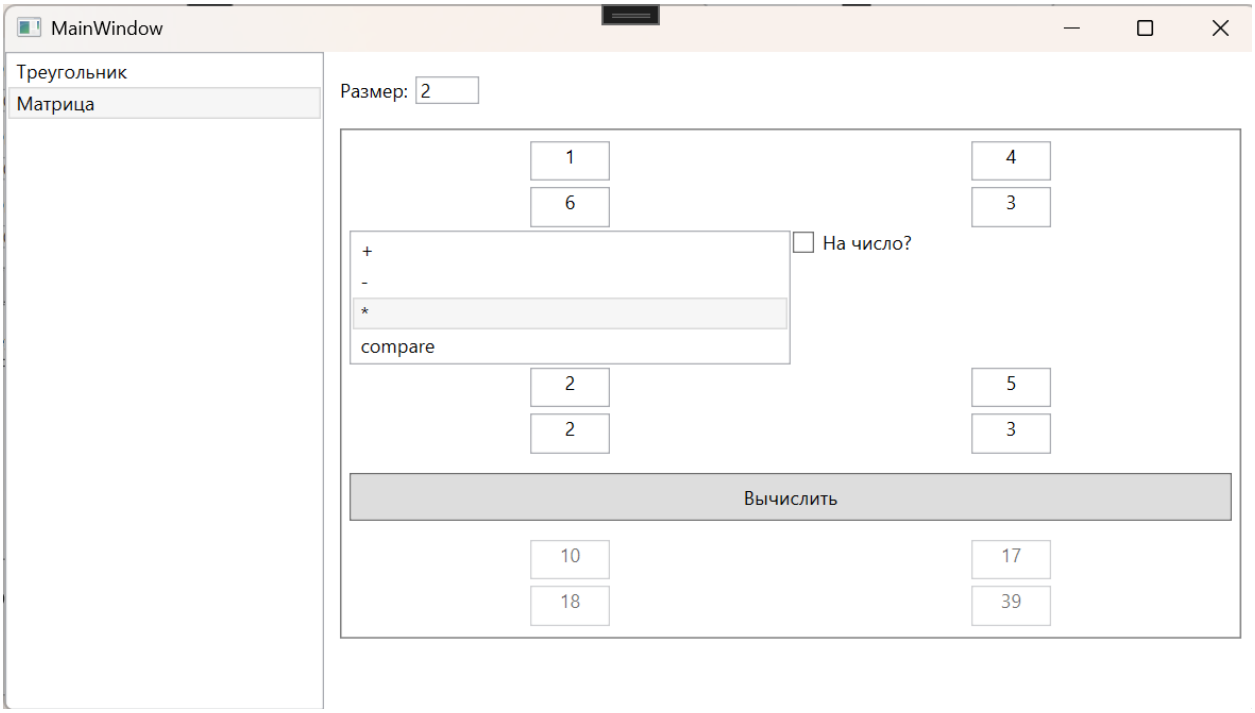
Задание с матрице



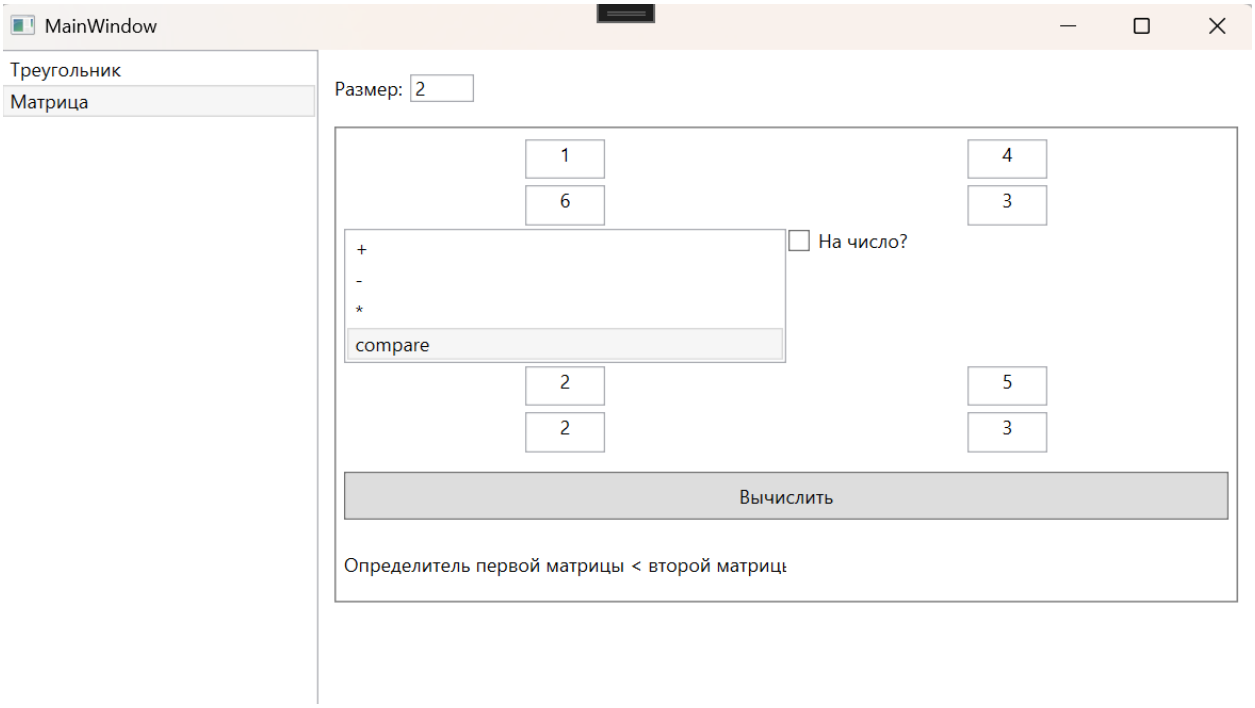
Пример использования операции разности матриц



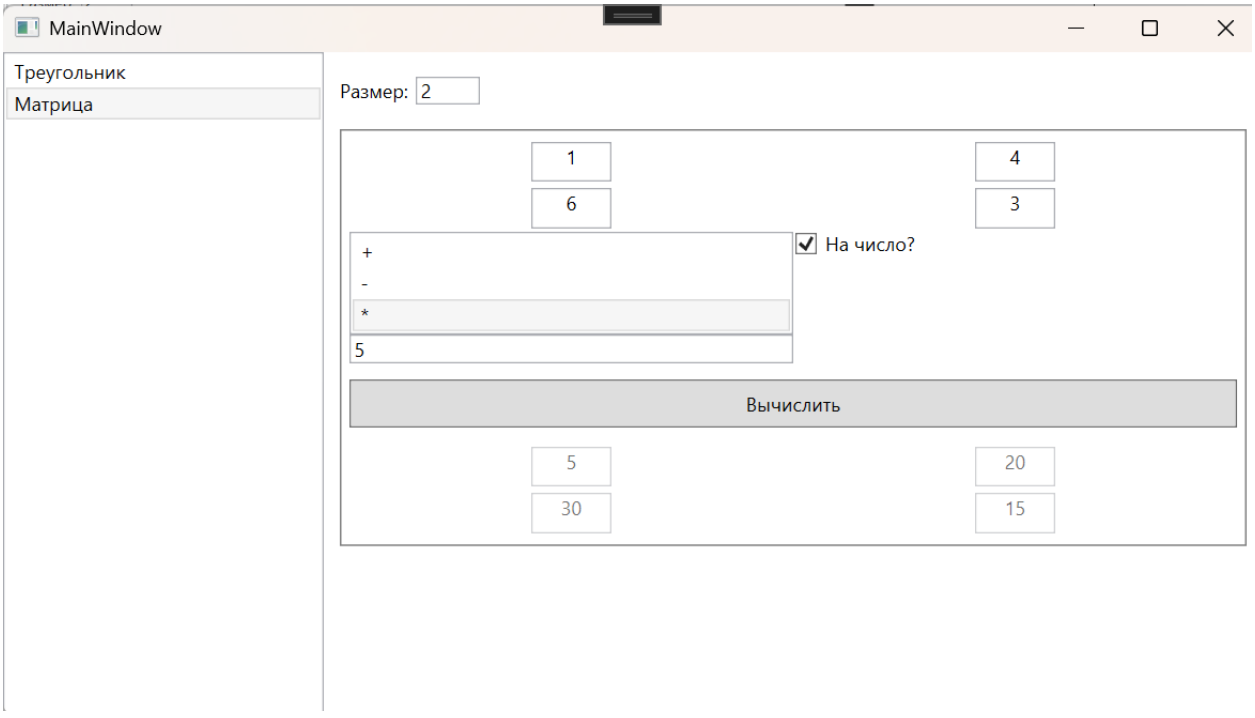
Пример использование операции сложения



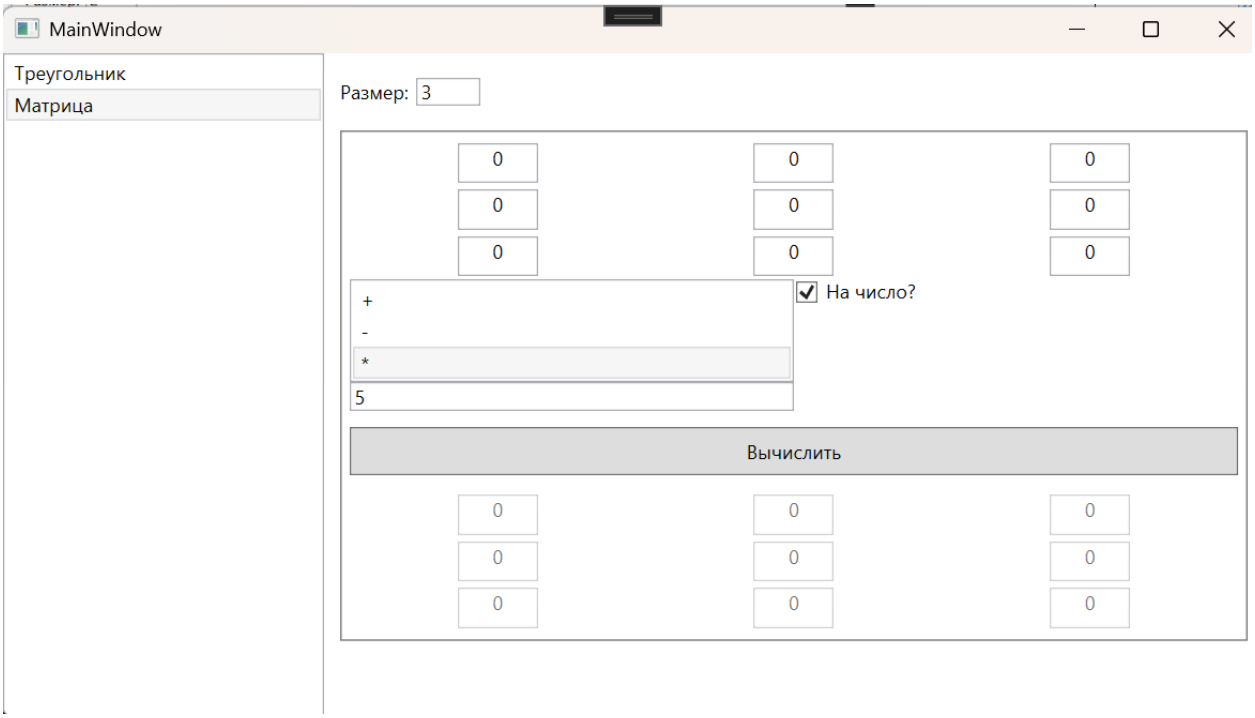
Пример использования операции умножения



Использование операции сравнения



Использования операции умножение на число



Задание 4

Создать класс **Пароль**. Поле класса – пароль. Метод класса – проверка пароля с выводом информационного сообщения: «Пароль верный» или «Пароль неверный». Для простоты пароль будет задаваться программистом в основной программе. Создать класс **Надежный пароль**, который является потомком класса **Пароль** и имеет собственный метод анализа надежности пароля. Этот метод будет вызываться при регистрации пользователя на определенной цифровой платформе.

Надёжный пароль отвечает следующим требованиям:

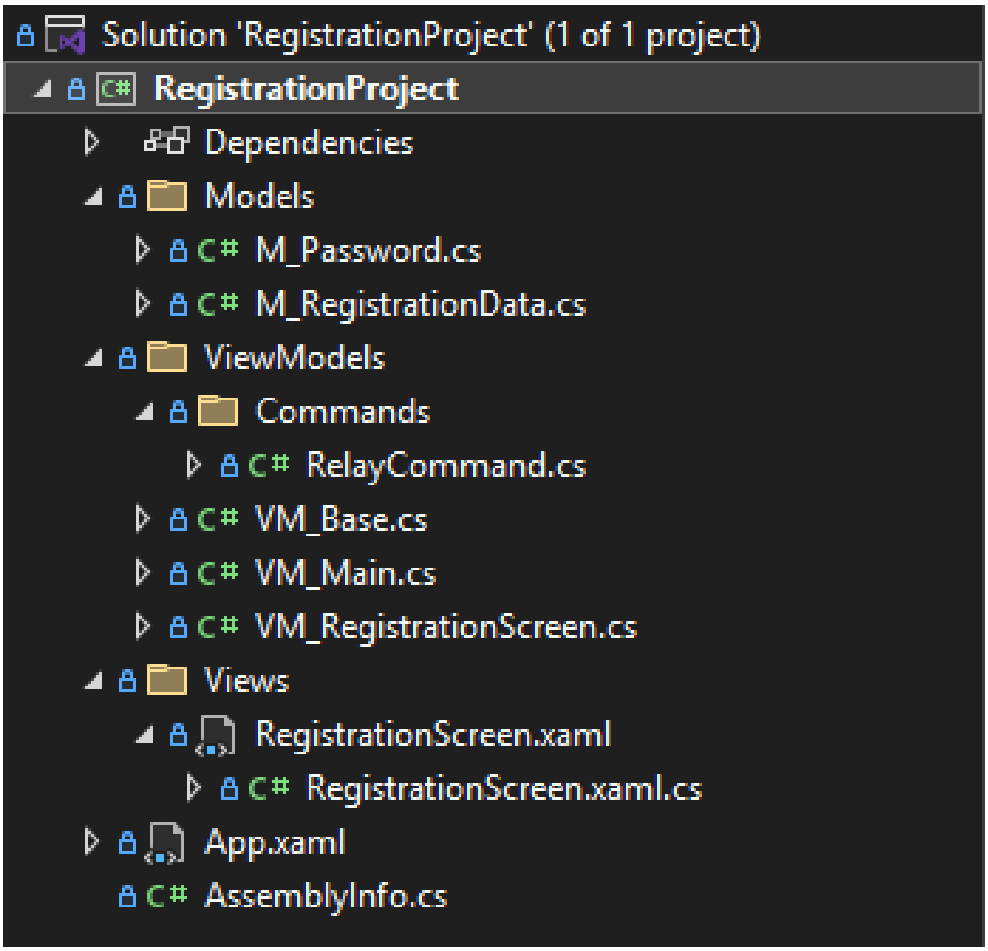
Состоит более чем из 12 знаков. Содержит в себе строчные и ПРОПИСНЫЕ буквы, цифры и символы.

Не содержит последовательность более чем из двух знаков, расположенных рядом друг с другом на клавиатуре (например, использование комбинаций 1234 и qwerty не допускается).

Не содержит ту информацию, которую легко найти о пользователе в социальных сетях. То есть например, не содержит личную информацию(имя, фамилия, дата рождения пользователя(как минимум 2 последние цифры), никнейм).

Взаимодействия с базой данных в этом задании не предполагается, поэтому разрешается хранение личных данных пользователя при регистрации в массивах, списках или вспомогательных переменных.

а) подключить модуль с описанными классами;
б) разместить на форме текстовое поле для ввода пароля;
в) в обработчике события кнопки **Проверка** реализовать работу с созданными классами.



Структура проект

Код файла M_Password

Содержит классы Password и ReliablePassword, реализующие проверку совпадения и надёжности пароля.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using System.Runtime.CompilerServices;

namespace Models
{
```

```
    public class Password : INotifyPropertyChanged
    {
        protected string correctPassword = "MySecret123!";

        public event PropertyChangedEventHandler? PropertyChanged;

        public Password(string password = "MySecret123!")
        {
            Registrate(password);
        }
        public void Registrate(string newPassword)
        {
            correctPassword = newPassword;
        }
        public virtual string Check(string input)
        {
            return input == correctPassword ? "Пароль верный" : "Пароль неверный";
        }
        public void OnPropertyChanged([CallerMemberName] string propertyName = null)
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
    }
    public class ReliablePassword : Password
    {
        private readonly string[] forbiddenPatterns = { "1234567890",
"qwertyuyiop[]","asdfghjkl;'","zxcvbnm,./","!@#$%^&*()_+","QWERTYUIOP{}","ASDFGHJKL:
","ZXCVBNM<>?" };
        private M_RegistrationData m_RegistrationData;
        private string[] personalInfo;

        private string _passwordErrorMessage;

        public string PasswordErrorMessage
        {
            get { return _passwordErrorMessage; }
            set { _passwordErrorMessage = value; }
        }

        public ReliablePassword(string password, M_RegistrationData
m_RegistrationData)
        {
            this.m_RegistrationData = m_RegistrationData;

            this.personalInfo = new string[5] {
                m_RegistrationData.UserName,                        //Ник
пользователя
                m_RegistrationData.Name,                            //Имя
пользователя
                m_RegistrationData.SecondName,                      //Фамилия
пользователя
                m_RegistrationData.BornDate,              //Дата рождения
                m_RegistrationData.BornDate.Substring(2) //Дата рождения (только два
последних символа)
            };

            Registrate (password, personalInfo);
        }

        public ReliablePassword()
        {

        }
```

```
        private bool ContainsKeyboardSequence(string password)
        {
            string lowerPass = password.ToLower();

            foreach (var pattern in forbiddenPatterns)
            {
                string lowerPattern = pattern.ToLower();
                string reversedPattern = new
string(lowerPattern.Reverse().ToArray());

                if (HasSequence(lowerPass, lowerPattern) || HasSequence(lowerPass,
reversedPattern))
                    return true;
            }

            return false;
        }

        private bool HasSequence(string text, string pattern)
        {
            for (int seqLen = 4; seqLen <= pattern.Length; seqLen++)
            {
                for (int i = 0; i <= pattern.Length - seqLen; i++)
                {
                    string segment = pattern.Substring(i, seqLen);
                    if (text.Contains(segment))
                        return true;
                }
            }
            return false;
        }

        public string Registrate(string password, string[] personalInfo)
        {
            bool isReliable = true;
            if (password.Length <= 12)
                PasswordErrorMessage = "Меньше 12 символов";

            else if (!password.Any(char.IsUpper))
                PasswordErrorMessage = "Нет прописных символов";

            else if(!password.Any(char.IsLower))
                PasswordErrorMessage = "Нет строчных символов";

            else if(!password.Any(char.IsDigit))
                PasswordErrorMessage = "Нет цифр";

            else if(!Regex.IsMatch(password, @"[!@#$%^&*(),.?""{}|<>]"))
                PasswordErrorMessage = "Нет специальных симолов";

            else if (ContainsKeyboardSequence(password))
                PasswordErrorMessage = "Содержит последовательность более 3 символов
подряд (например, 1234, qwerty)";

            else if (personalInfo.Any(info => password.ToLower().Contains(info)))
                PasswordErrorMessage = "Содержит личные данные пользователя";

            else
                PasswordErrorMessage = "Пароль принят";

            correctPassword = password;
            return PasswordErrorMessage;
        }
    }
}
```

Код файла M_RegistrationData:

Модель данных пользователя, содержащая персональные сведения, используемые при проверке надёжности пароля.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;

namespace Models
{
    public class M_RegistrationData : INotifyPropertyChanged
    {
            private string _name;
            public string Name
            {
                  get { return _name; }
                  set { _name = value; }
            }

            private string _secondName;
            public string SecondName
            {
                  get { return _secondName; }
                  set { _secondName = value; }
            }

        private string _userName;
        public string UserName
        {
            get { return _userName; }
            set { _userName = value; }
        }

        private string _bornDate;
             public string BornDate
             {
                  get { return _bornDate; }
                  set { _bornDate = value; }
             }

             private ReliablePassword reliablePassword;
             public ReliablePassword ReliablePassword
             {
                  get { return reliablePassword; }
                  set { reliablePassword = value; }
             }

        public event PropertyChangedEventHandler? PropertyChanged;
        public void OnPropertyChanged([CallerMemberName] string propertyName = null)
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Код файла VM_Base

Базовый класс ViewModel, реализующий интерфейс `INotifyPropertyChanged`.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;

namespace ViewModels
{
    public class VM_Base : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler? PropertyChanged;
        public void OnPropertyChanged([CallerMemberName] string propertyName = null)
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Код файла VM_RegistrationScreen

ViewModel экрана регистрации. Выполняет проверку пароля на основе введённых данных и формирует текстовое сообщение о результате.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
using Models;
using ViewModels.Commands;

namespace ViewModels
{
    public class VM_RegistrationScreen : VM_Base
    {
        private ReliablePassword _ReliablePassword;
        private M_RegistrationData _RegistrationData;
        public VM_RegistrationScreen() {
            _RegistrationData = new M_RegistrationData();
            _ReliablePassword = new ReliablePassword();
            RegistrateCommand = new RelayCommand(Registrate);
        }

        public ICommand RegistrateCommand { get; }
        private void Registrate()
        {
            if (!(string.IsNullOrWhiteSpace(Name) ||
                    string.IsNullOrWhiteSpace(SecondName) ||
                    string.IsNullOrWhiteSpace(UserName) ||
                    string.IsNullOrWhiteSpace(BornDate) ||
                    string.IsNullOrWhiteSpace(ReliablePassword))))
                    {
```

```
                    string[] Info = new string[] {
Name,SecondName,UserName,BornDate.ToString(),BornDate.ToString().Substring(2)
                    };
                PasswordError = _ReliablePassword.Registrate(ReliablePassword,
Info);
            }
            else
            {
                PasswordError = "Не все поля заполнены";
            }
        }

        public string Name
        {
            get { return _RegistrationData.Name; }
            set { _RegistrationData.Name = value;
                OnPropertyChanged();
            }
        }
        public string SecondName
        {
            get { return _RegistrationData.SecondName; }
            set { _RegistrationData.SecondName = value;
                OnPropertyChanged();
            }
        }
        public string UserName
        {
            get { return _RegistrationData.UserName; }
            set { _RegistrationData.UserName = value;
                OnPropertyChanged();
            }
        }
        public string BornDate
        {
            get {
                return _RegistrationData.BornDate;
            }
            set {
                _RegistrationData.BornDate = value;
                OnPropertyChanged();
            }
        }
        private string _password;
        public string ReliablePassword
        {
            get { return _password; }
            set { _password = value;
                OnPropertyChanged();
            }
        }

        private string _ErrorMessage;
        public string PasswordError
        {
            get { return _ErrorMessage; }
            set {
                _ErrorMessage = value;
                OnPropertyChanged();
            }
        }
    }
}
```

Код файла RegistrationScreen

Файл разметки формы регистрации пользователя.

```
<Window x:Class="LoginProject.Views.RegistrationScreen"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:vm="clr-namespace:ViewModels"
        mc:Ignorable="d"
        Title="RegistrationScreen" Height="450" Width="300">
    <Window.DataContext>
        <vm:VM_RegistrationScreen/>
    </Window.DataContext>
    <Grid>
        <StackPanel>

            <TextBlock Text="Регистрация"
                       FontSize="23"
                       Margin="10,10,10,0"
                       Background="BlueViolet"
                       Foreground="White"
                       TextAlignment="Center"
                       Padding="5"
                       ></TextBlock>

            <TextBlock  Text="UserName"
                        Margin="10,10"
                        ></TextBlock>

            <TextBox
                        Margin="10,0"
                        Text="{Binding UserName}"></TextBox>

            <TextBlock
                        Margin="10,10"
                        Text="Name"></TextBlock>

            <TextBox
                        Margin="10,0"
                        Text="{Binding Name}"></TextBox>

            <TextBlock
                        Margin="10,10"
                        Text="SecondName"></TextBlock>

            <TextBox
                        Margin="10,0"
                        Text="{Binding SecondName}"></TextBox>

            <TextBlock
                        Margin="10,10"
                        Text="BornDate"></TextBlock>

            <TextBox
                        Margin="10,0"
                        Text="{Binding BornDate}"></TextBox>

            <TextBlock
                        Margin="10,10"
                        Text="Password"></TextBlock>
            <TextBox
                        Margin="10,0"
```

```
                    Text="{Binding ReliablePassword}"></TextBox>

            <TextBlock
                    Margin="10,10"
                    Foreground="IndianRed"
                    Text="{Binding PasswordError}"></TextBlock>

            <Button
                    Margin="10,0"
                    Background="BlueViolet"
                    Foreground="White"
                    Height="40"
                    FontSize="20"
                    Content="Registre" Command="{Binding
RegistrateCommand}"></Button>
        </StackPanel>
    </Grid>
</Window>
```

Описание кода

View связывается с VM_RegistrationScreen.

ViewModel получает данные пользователя, передаёт их в модель ReliablePassword, получает результат проверки и отображает его в интерфейсе.

Модели инкапсулируют правила проверки и не зависят от пользовательского интерфейса.

Взаимодействие файлов и слоев приложения

1. Запуск окна регистрации.

   Открывается экран регистрации (View), который связан с VM_RegistrationScreen через DataContext. Все поля ввода (ФИО/ник/дата рождения/пароль) привязаны (Binding) к свойствам ViewModel.

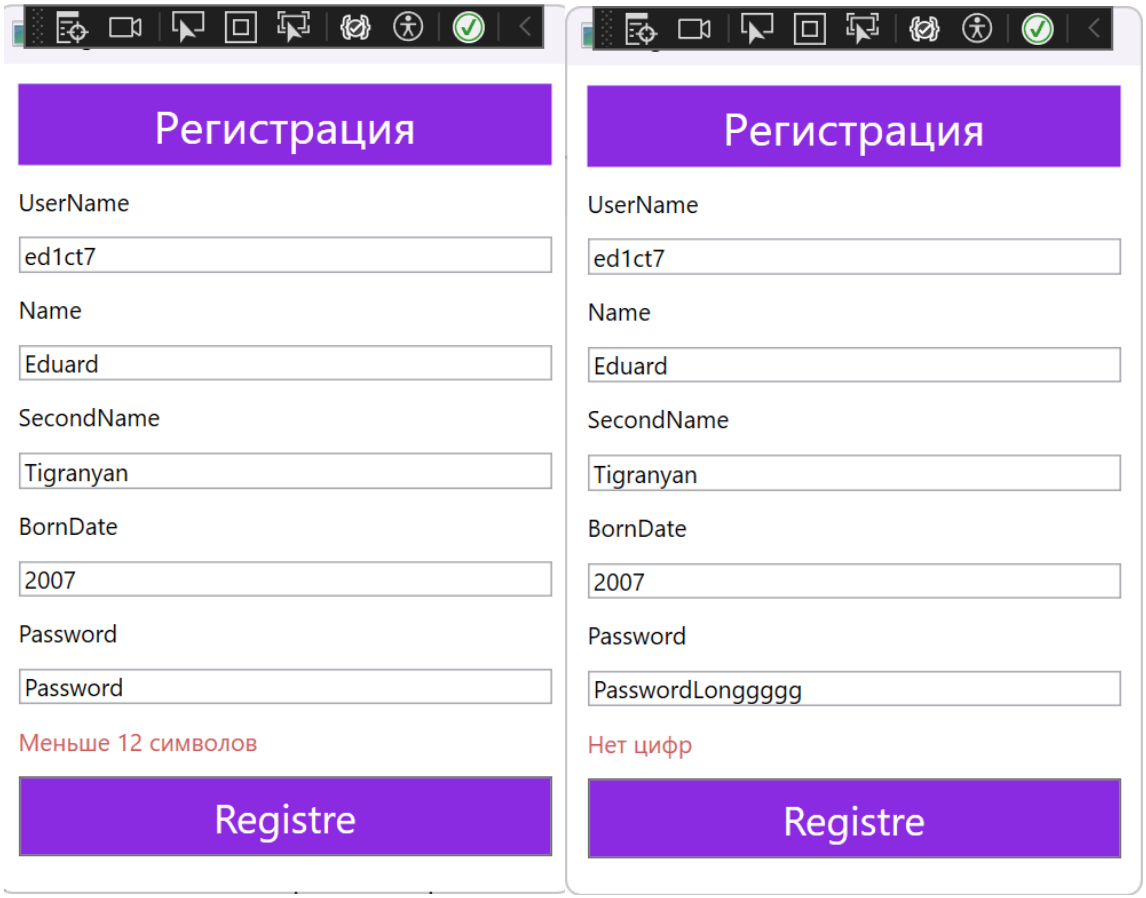
2. Ввод данных пользователем.

   Пользователь вводит персональные данные и пароль. WPF автоматически обновляет соответствующие свойства во ViewModel (двусторонняя привязка), без прямого вызова методов формы.

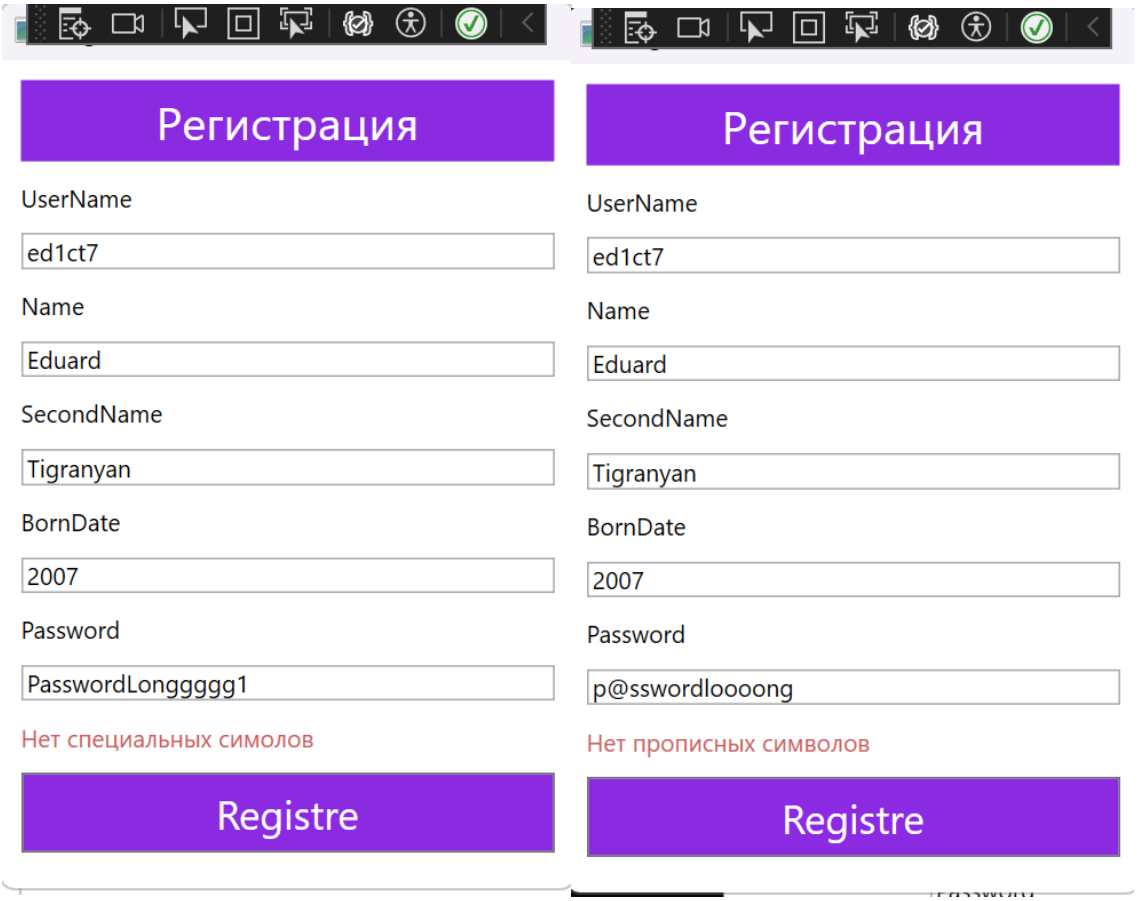
3. Нажатие кнопки “Регистрация/Проверить”.
   Кнопка в интерфейсе вызывает команду ViewModel (через ICommand). Это запускает основной сценарий проверки.
4. Сбор данных во ViewModel.
   VM_RegistrationScreen берёт введённые значения из своих свойств и формирует данные для проверки: пароль + персональные сведения (имя, фамилия, ник, дата рождения).
5. Передача данных в модель проверки пароля.
   ViewModel не проверяет пароль сама, а передаёт данные в модель ReliablePassword (и связанные классы) — именно там находятся правила надёжности.
6. Проверка в модели (основная логика).
   ReliablePassword выполняет валидацию: длина, наличие разных типов символов (регистр/цифры/спецсимволы), а также проверку на использование персональных данных и очевидных последовательностей (то есть “небезопасных” шаблонов). По итогу формируется результат: “пароль подходит” или конкретная причина, почему не подходит.
7. Возврат результата во ViewModel.
   Модель возвращает результат проверки обратно во ViewModel (например, текст ошибки/успеха). ViewModel записывает этот результат в своё свойство (например, PasswordError / сообщение статуса).
8. Отображение результата в интерфейсе.
   Так как текст результата привязан к элементу интерфейса, View автоматически обновляет отображение (показывает сообщение пользователю) без дополнительного кода во View.

Результат

Проверка на базовые критерии безопасности:

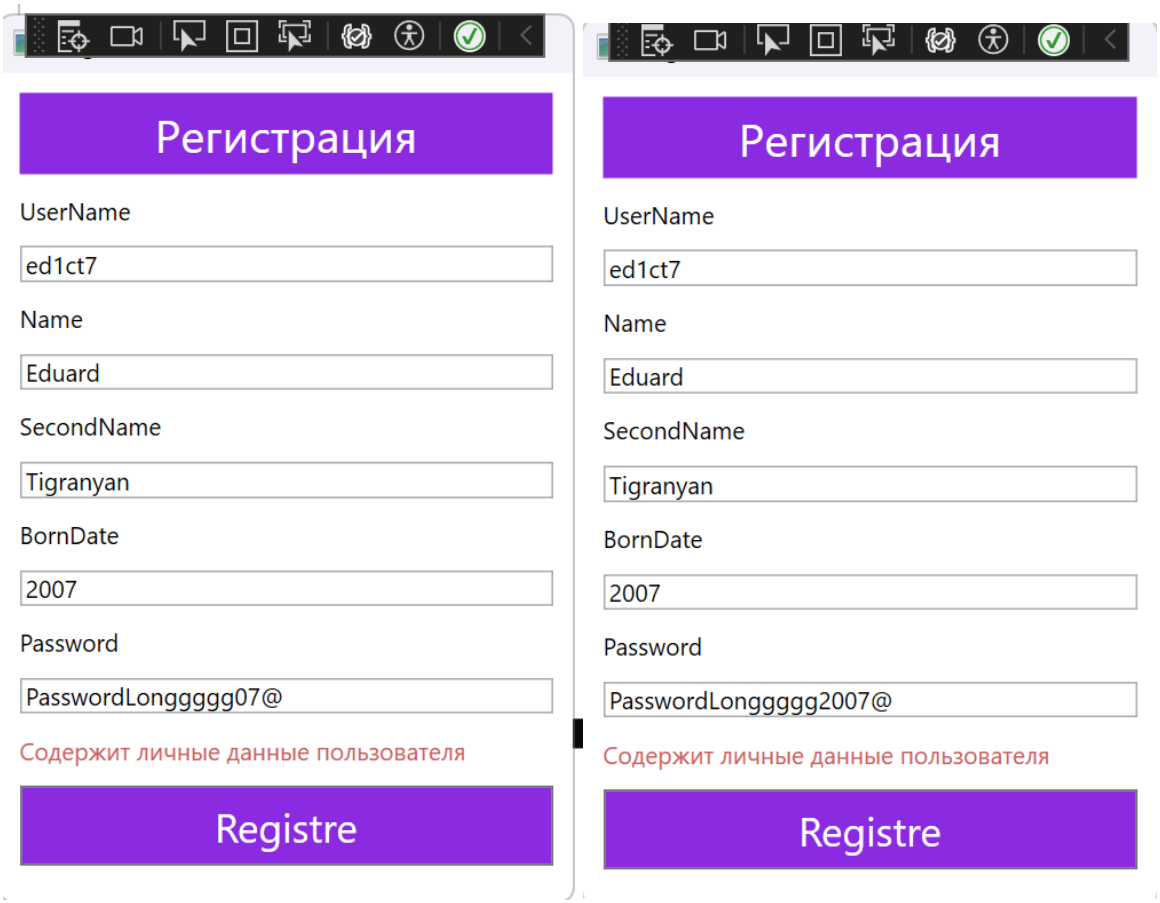


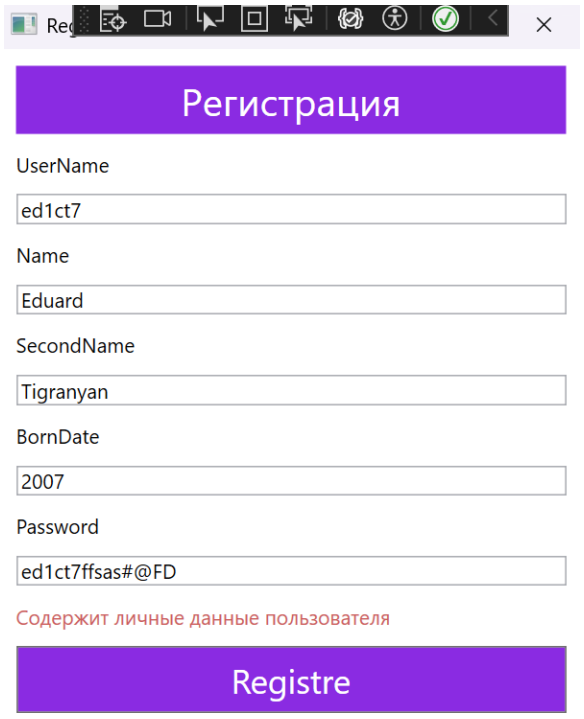
Проверки на количество символов и на присутствие цифр



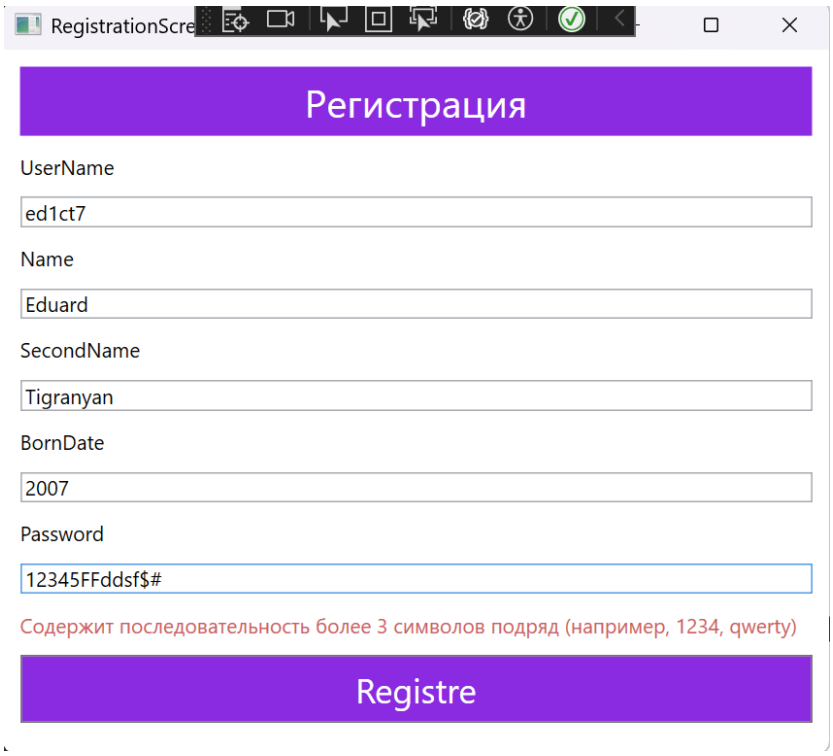
Проверки на присутствие специальных и прописных символов

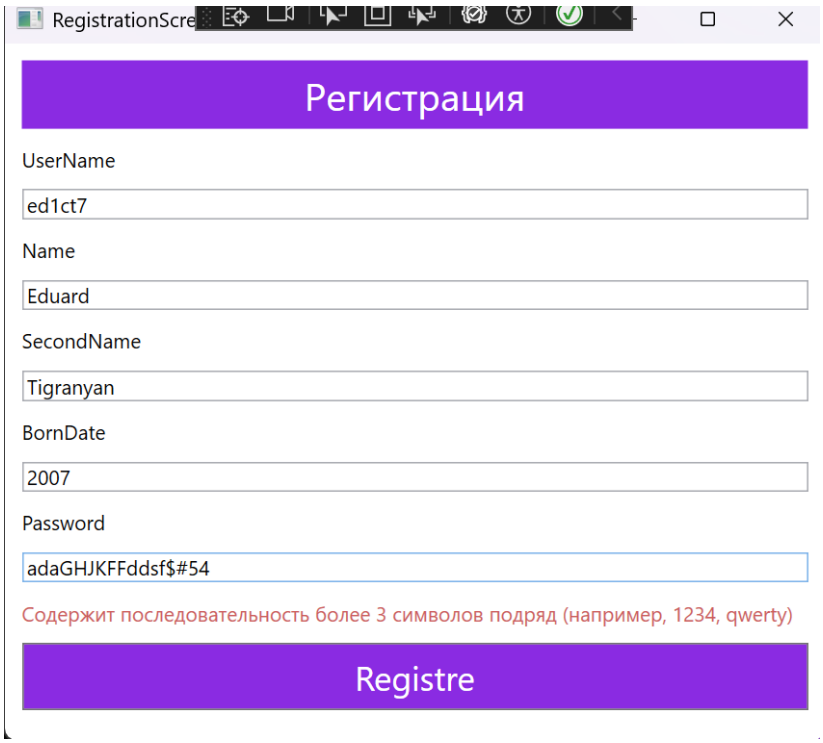
Проверка на личные данные пользователя:

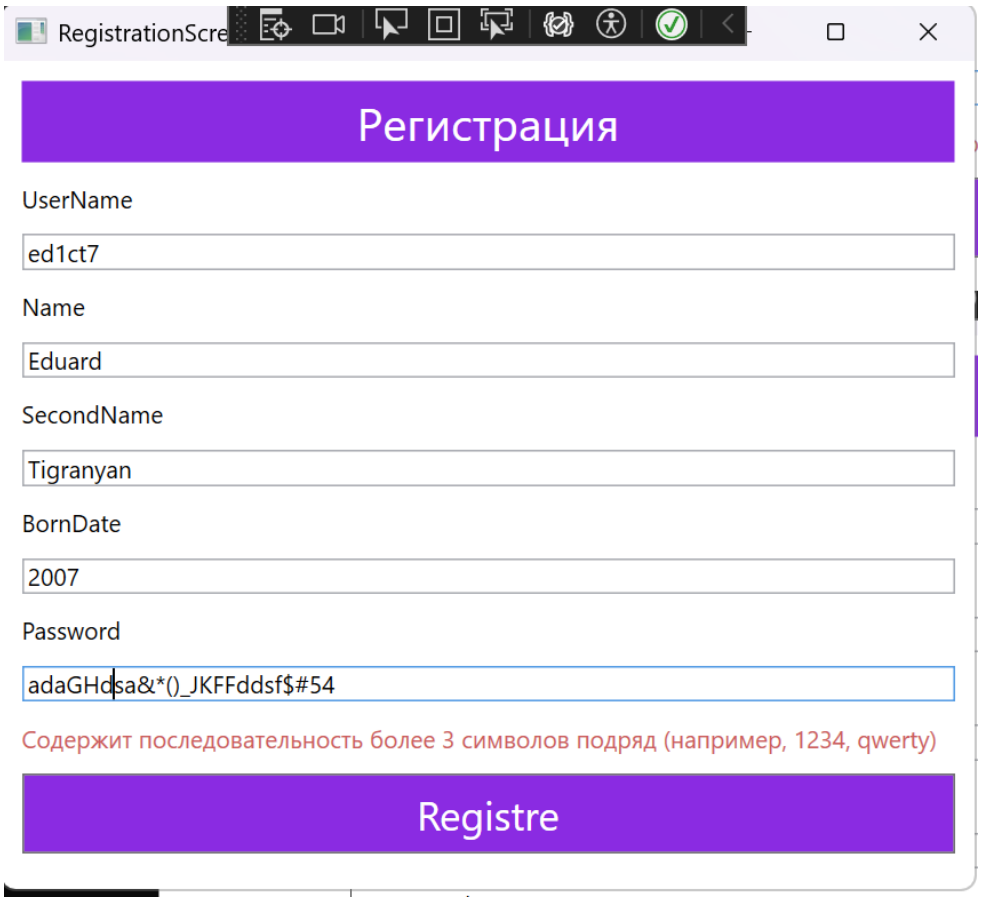


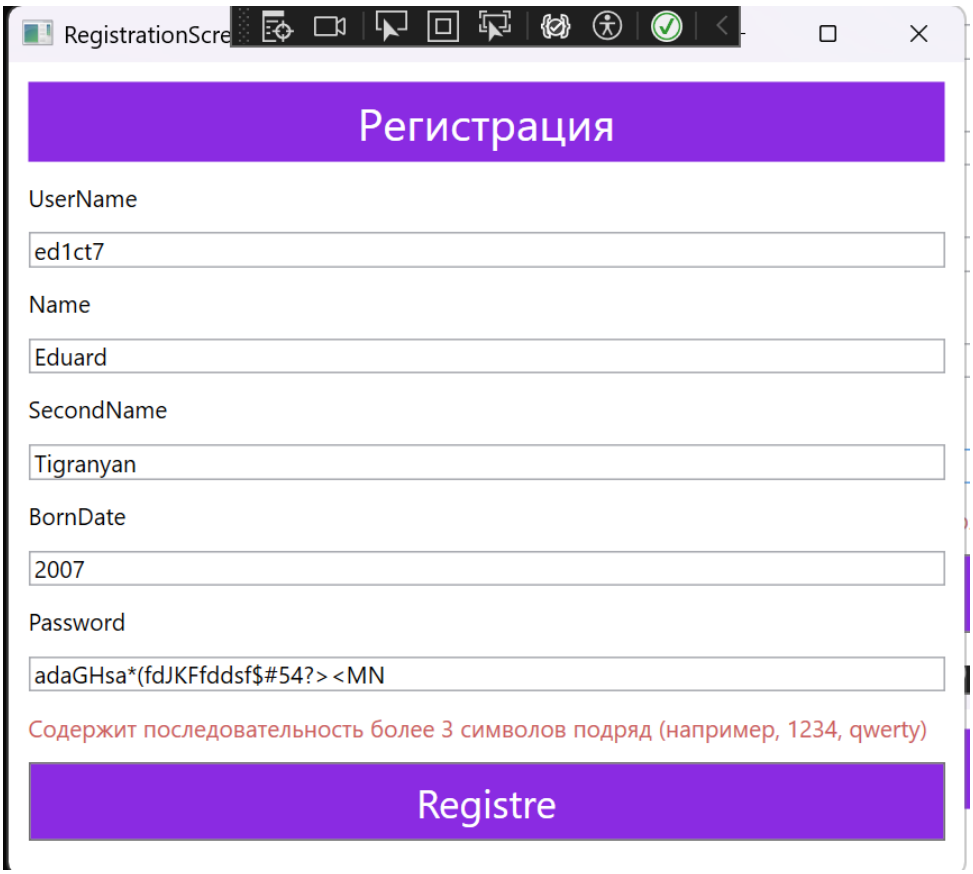


Проверка на символы подряд:

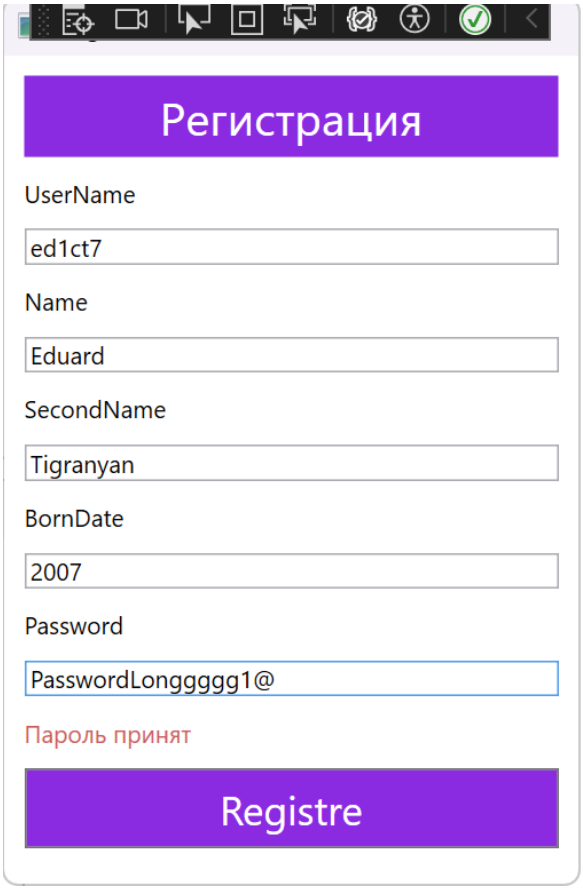





RegistrationScre
Регистрация
UserName
ed1ct7
Name
Eduard
SecondName
Tigranyan
BornDate
2007
Password
adaGHdsa&*()_JKFFddsf$#54
Содержит последовательность более 3 символов подряд (например, 1234, qwerty)
Registre

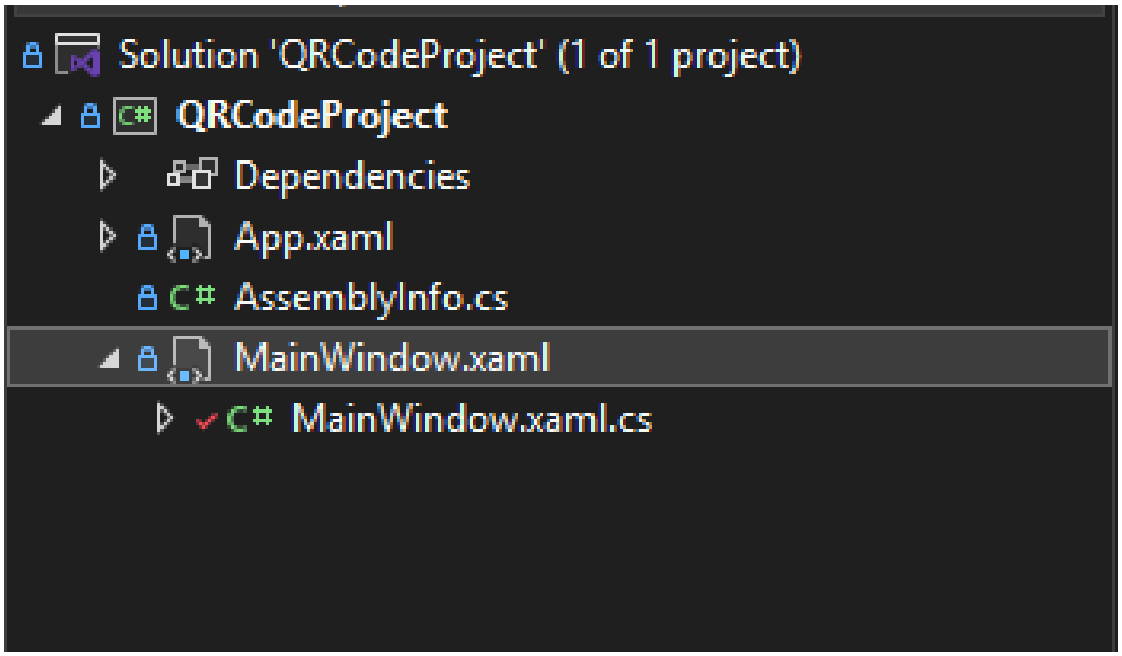

RegistrationScre
Регистрация
UserName
ed1ct7
Name
Eduard
SecondName
Tigranyan
BornDate
2007
Password
adaGHsa*(fdJKFfddsf$#54?><MN
Содержит последовательность более 3 символов подряд (например, 1234, qwerty)
Registre

Успешный пароль



Задание 5

По требованиям заказчика необходимо отслеживать качество работы сервиса по отзывам клиентов. В функционал программного модуля в этих целях необходимо добавить генерацию QR-кода для оценки работы сервиса (при сканировании кода в телефоне выдаётся ссылка на форму с опросом (ссылка в ресурсах). Можно использовать библиотеку для генерации QR-кодов на языке C#(с открытым исходным кодом) - **QRCoder** .



Структура проекта

Код файла MainWindow

Файл XAML-разметки главного окна приложения «QR-код генератор». Определяет элементы интерфейса: поле ввода данных, кнопку запуска генерации QR-кода, элемент Image для отображения сформированного изображения, а также текстовую подсказку о сохранении по двойному клику.

```
<Window x:Class="QRCodeGeneratorApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="QR-код генератор" Height="450" Width="400"
        WindowStartupLocation="CenterScreen">

    <Grid Margin="20">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>

        <TextBlock Text="QRCoder" FontSize="20"
```

```
                    FontWeight="Bold" HorizontalAlignment="Center"/>

        <StackPanel Grid.Row="1" Orientation="Horizontal" Margin="0,20,0,10">
            <TextBlock Text="Данные:" VerticalAlignment="Center" FontSize="14"
Margin="0,0,5,0"/>
            <TextBox x:Name="DataTextBox" Width="200" FontSize="14"/>
            <Button Content="Создать" Click="CreateQR_Click"
                    Margin="10,0,0,0" Padding="10,2" FontSize="14"/>
        </StackPanel>

        <Image x:Name="QrImage" Grid.Row="2"
               Stretch="None" HorizontalAlignment="Center"
VerticalAlignment="Center"
               MouseDown="QrImage_MouseDown"/>

        <TextBlock Grid.Row="3" Text="Нажмите дважды по изображению, чтобы сохранить
его"
                   FontStyle="Italic" FontSize="12"
                   HorizontalAlignment="Center" Margin="0,10,0,0"/>
    </Grid>
</Window>
```

Код файла MainWindow.xaml.cs

Code-behind файл главного окна. Реализует прикладную логику:

- обработку события нажатия кнопки «Создать»;
- формирование QR-кода с использованием библиотеки QRCoder;
- преобразование Bitmap в BitmapImage для вывода в WPF-интерфейсе;
- обработку двойного клика по изображению и сохранение результата в файл PNG через SaveFileDialog.

```
using QRCoder;
using System;
using System.Drawing;
using System.IO;
using System.Windows;
using System.Windows.Input;
using System.Windows.Media.Imaging;

namespace QRCodeGeneratorApp
{
    public partial class MainWindow : Window
    {
        private BitmapImage qrImage;

        public MainWindow()
        {
            InitializeComponent();
        }

        // Создание QR-кода
        private void CreateQR_Click(object sender, RoutedEventArgs e)
        {
            string text = DataTextBox.Text.Trim();

            if (string.IsNullOrEmpty(text))
            {
```

```
                MessageBox.Show("Введите текст!");
                return;
            }

            var generator = new QRCodeGenerator();
            var data = generator.CreateQrCode(text, QRCodeGenerator.ECCLevel.Q);
            var qrCode = new QRCode(data);
            Bitmap bmp = qrCode.GetGraphic(20);

            qrImage = ConvertBitmap(bmp);
            QrImage.Source = qrImage;
        }

        // Конвертация Bitmap → BitmapImage
        private BitmapImage ConvertBitmap(Bitmap bmp)
        {
            var ms = new MemoryStream();
            bmp.Save(ms, System.Drawing.Imaging.ImageFormat.Png);
            ms.Position = 0;

            var img = new BitmapImage();
            img.BeginInit();
            img.StreamSource = ms;
            img.CacheOption = BitmapCacheOption.OnLoad;
            img.EndInit();
            return img;
        }

        // Сохранение по двойному клику
        private void QrImage_MouseDown(object sender, MouseButtonEventArgs e)
        {
            if (e.ClickCount != 2 || qrImage == null) return;

            var dialog = new Microsoft.Win32.SaveFileDialog
            {
                Filter = "PNG файл|*.png",
                FileName = "qrcode.png"
            };

            if (dialog.ShowDialog() == true)
            {
                var encoder = new PngBitmapEncoder();
                encoder.Frames.Add(BitmapFrame.Create(qrImage));
                using var file = File.Create(dialog.FileName);
                encoder.Save(file);

                MessageBox.Show("QR-код сохранён!");
            }
        }
    }
}
```

Описание кода

Пользовательский интерфейс, описанный в MainWindow.xaml, инициирует события управления:

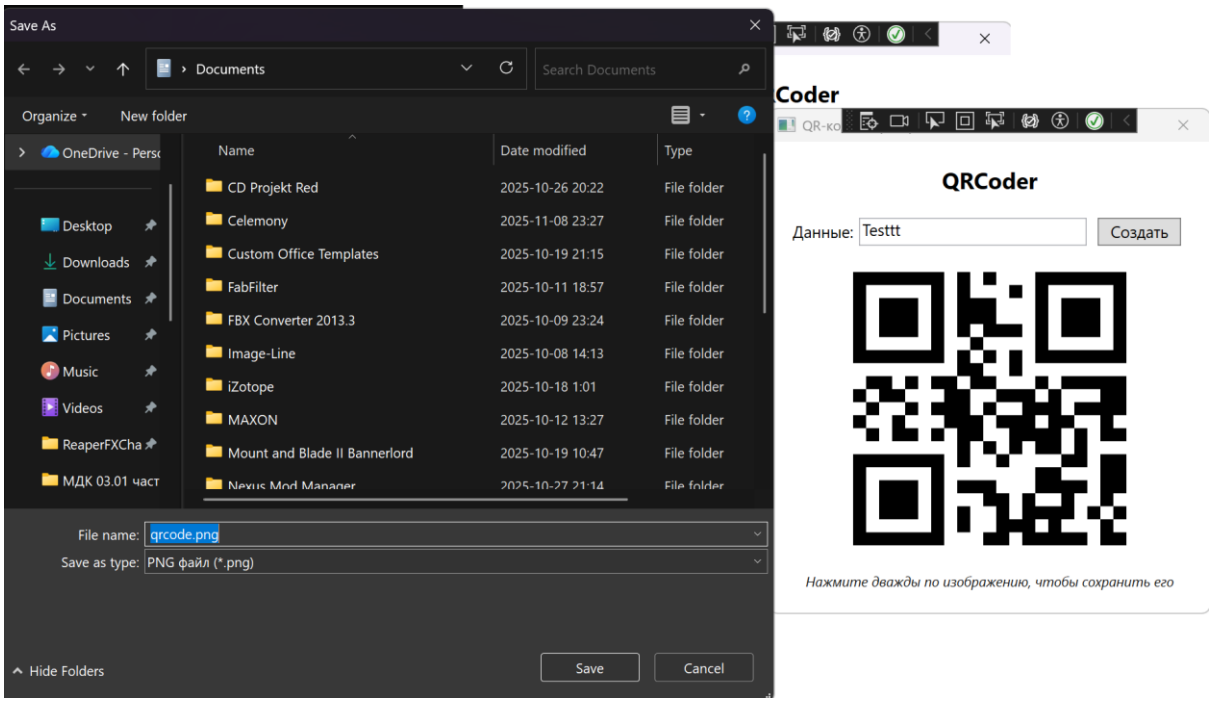
нажатие кнопки вызывает обработчик CreateQR_Click;

двойной клик по изображению вызывает обработчик QrImage_MouseDown.

Результат

Главное окно

Генерация QRCode



Результат двойного нажатия

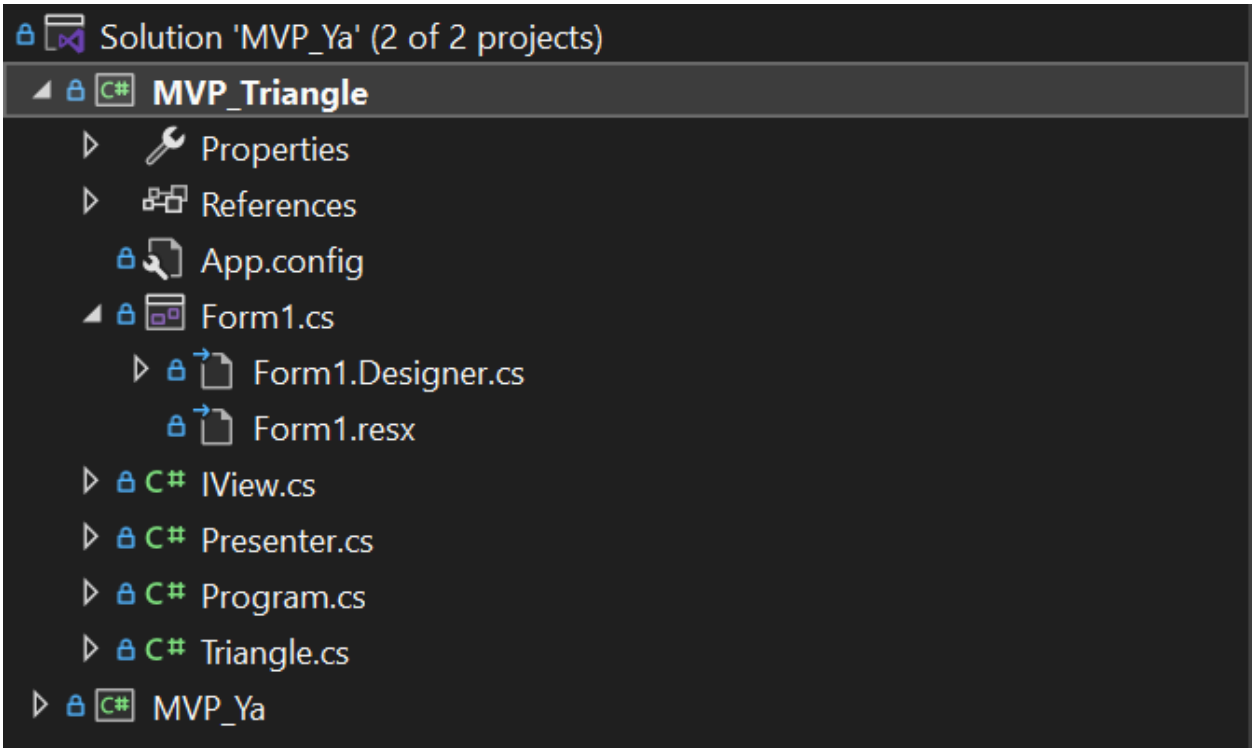
Лабораторная работа №3

Тема: Знакомство с паттернами проектирования MVP и MVVM в MS Visual Studio(C#).

Цель работы: получение практических навыков при использовании паттернов при решении прикладных задач.

Задание 1

Реализовать задание в соответствии с индивидуальным вариантом из лабораторной работы по классам — реализовать индивидуальное задание с использованием паттерна MVP.



Структура проекта

Файл Form1.cs

Представление (View). Реализует интерфейс `IView`, предоставляет данные ввода и отображает результат.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
```

```
namespace MVP_Triangle
{
    public partial class Form1 : Form, IView
    {
        Presenter presenter;
        public Form1()
        {
            InitializeComponent();
            presenter = new Presenter(this);
        }

        public event EventHandler<EventArgs> SetA;
        public event EventHandler<EventArgs> SetB;
        public event EventHandler<EventArgs> SetC;

        public string Data
        {
            get => label5.Text;
            set => label5.Text = value;
        }

        public T TryFuncCheck<T>(Func<T> func)
        {
            try
            {
                return func();
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        public double InputA
        {
            get => TryFuncCheck(() => Convert.ToDouble(textBox1.Text));
            set { textBox1.Text = value.ToString(); }
        }

        public double InputB
        {
            get => TryFuncCheck(() => Convert.ToDouble(textBox2.Text));
            set { textBox2.Text = value.ToString(); }
        }

        public double InputC
        {
            get => TryFuncCheck(() => Convert.ToDouble(textBox3.Text));
            set { textBox3.Text = value.ToString(); }
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {
            if(SetA != null) SetA(this, EventArgs.Empty);
        }

        private void textBox2_TextChanged(object sender, EventArgs e)
        {
            if (SetB != null) SetB(this, EventArgs.Empty);
        }

        private void textBox3_TextChanged(object sender, EventArgs e)
        {
            if (SetC != null) SetC(this, EventArgs.Empty);
        }
```

```
        private void label3_Click(object sender, EventArgs e)
        {

        }
    }
}
```

Файл Form1.Designer.cs

Автоматически сгенерированный файл, содержащий описание элементов формы.

```
namespace MVP_Triangle
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed;
otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.label1 = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();
            this.label3 = new System.Windows.Forms.Label();
            this.textBox1 = new System.Windows.Forms.TextBox();
            this.textBox2 = new System.Windows.Forms.TextBox();
            this.textBox3 = new System.Windows.Forms.TextBox();
            this.label4 = new System.Windows.Forms.Label();
            this.label5 = new System.Windows.Forms.Label();
            this.SuspendLayout();
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Location = new System.Drawing.Point(52, 148);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(89, 20);
            this.label1.TabIndex = 0;
```

```
            this.label1.Text = "Сторона A";
            //
            // label2
            //
            this.label2.AutoSize = true;
            this.label2.Location = new System.Drawing.Point(212, 148);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(89, 20);
            this.label2.TabIndex = 1;
            this.label2.Text = "Сторона B";
            //
            // label3
            //
            this.label3.AutoSize = true;
            this.label3.Location = new System.Drawing.Point(366, 148);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(89, 20);
            this.label3.TabIndex = 2;
            this.label3.Text = "Сторона C";
            //
            // textBox1
            //
            this.textBox1.Location = new System.Drawing.Point(56, 200);
            this.textBox1.Name = "textBox1";
            this.textBox1.Size = new System.Drawing.Size(100, 26);
            this.textBox1.TabIndex = 3;
            this.textBox1.TextChanged += new
System.EventHandler(this.textBox1_TextChanged);
            //
            // textBox2
            //
            this.textBox2.Location = new System.Drawing.Point(216, 200);
            this.textBox2.Name = "textBox2";
            this.textBox2.Size = new System.Drawing.Size(100, 26);
            this.textBox2.TabIndex = 4;
            this.textBox2.TextChanged += new
System.EventHandler(this.textBox2_TextChanged);
            //
            // textBox3
            //
            this.textBox3.Location = new System.Drawing.Point(370, 200);
            this.textBox3.Name = "textBox3";
            this.textBox3.Size = new System.Drawing.Size(100, 26);
            this.textBox3.TabIndex = 5;
            this.textBox3.TextChanged += new
System.EventHandler(this.textBox3_TextChanged);
            //
            // label4
            //
            this.label4.AutoSize = true;
            this.label4.Location = new System.Drawing.Point(52, 291);
            this.label4.Name = "label4";
            this.label4.Size = new System.Drawing.Size(230, 20);
            this.label4.TabIndex = 6;
            this.label4.Text = "Информация о треугольнике";
            //
            // label5
            //
            this.label5.AutoSize = true;
            this.label5.Location = new System.Drawing.Point(56, 325);
            this.label5.Name = "label5";
            this.label5.Size = new System.Drawing.Size(51, 20);
            this.label5.TabIndex = 7;
            this.label5.Text = "label5";
            //
```

```
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(9F, 20F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(707, 513);
            this.Controls.Add(this.label5);
            this.Controls.Add(this.label4);
            this.Controls.Add(this.textBox3);
            this.Controls.Add(this.textBox2);
            this.Controls.Add(this.textBox1);
            this.Controls.Add(this.label3);
            this.Controls.Add(this.label2);
            this.Controls.Add(this.label1);
            this.Name = "Form1";
            this.Text = "Form1";
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.TextBox textBox1;
        private System.Windows.Forms.TextBox textBox2;
        private System.Windows.Forms.TextBox textBox3;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.Label label5;
    }
}
```

Файл IView.cs

Интерфейс представления, определяющий свойства и события для взаимодействия с Presenter.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MVP_Triangle
{
    public interface IView
    {
        string Data { get; set; }
        double InputA { get; set; }
        double InputB { get; set; }
        double InputC { get; set; }

        event EventHandler<EventArgs> SetA;
        event EventHandler<EventArgs> SetB;
        event EventHandler<EventArgs> SetC;
    }
}
```

Файл Presenter.cs

Посредник между View и моделью. Получает данные из View, выполняет расчёты и возвращает результат обратно в View.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MVP_Triangle
{
    public class Presenter
    {
        private Triangle _triangle = new Triangle();
        private IView _view;

        public Presenter(IView view)
        {
            _view = view;
            _view.SetA += new EventHandler<EventArgs>(OnSetA);
            _view.SetB += new EventHandler<EventArgs>(OnSetB);
            _view.SetC += new EventHandler<EventArgs>(OnSetC);
            RefreshView();
        }

        public void OnSetA(object sender, EventArgs e)
        {
            _triangle.Sides.A=_view.InputA;
            RefreshView();
        }
        public void OnSetB(object sender, EventArgs e)
        {
            _triangle.Sides.B = _view.InputB;
            RefreshView();
        }
        public void OnSetC(object sender, EventArgs e)
        {
            _triangle.Sides.C = _view.InputC;
            RefreshView();
        }

        public void RefreshView()
        {
            _view.Data = _triangle.ToString();
        }
    }
}
```

Файл Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MVP_Triangle
{
    internal static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
```

```
        [STAThread]
        static void Main()
        {
            Form1 view = new Form1();
            Presenter presenter = new Presenter(view);
            Application.Run(view);
        }
    }
}
```

Файл Triangle.cs

Модель треугольника из Лабораторной работы 2

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;

namespace MVP_Triangle
{
    public class Parameters
    {
        private double _a;
        public double A
        {
            get { return _a; }
            set
            {
                _a = value;
            }
        }
        private double _b;
        public double B
        {
            get { return _b; }
            set
            {
                _b = value;
            }
        }
        private double _c;
        public double C
        {
            get { return _c; }
            set
            {
                _c = value;
            }
        }

        public override string ToString()
        {
            return $"A: {_a:F2}, B: {_b:F2}, C: {_c:F2}";
        }
    }

    public class Triangle
    {
        public Triangle(double a, double b, double c)
        {
            _sides = new Parameters { A = a, B = b, C = c };
```

```
    CalculateAngles();
}

public Triangle()
{
    _sides = new Parameters { A = 1, B = 1, C = 1 };
    CalculateAngles();
}

public bool Isosceles
{
    get => Math.Abs(Sides.A - Sides.B) < double.Epsilon ||
           Math.Abs(Sides.A - Sides.C) < double.Epsilon ||
           Math.Abs(Sides.B - Sides.C) < double.Epsilon;
}

private Parameters _sides;
public Parameters Sides
{
    get { return _sides; }
    set
    {
        _sides = value;
        CalculateAngles();
    }
}

private Parameters _angles;
public Parameters Angles
{
    get { return _angles; }
    set
    {
        _angles = value;
    }
}

public bool IsExist()
{
    double a = Sides.A;
    double b = Sides.B;
    double c = Sides.C;

    return a > 0 && b > 0 && c > 0 &&
           a + b > c &&
           a + c > b &&
           b + c > a;
}

public double Perimeter()
{
    if (!IsExist()) return 0;
    return Sides.A + Sides.B + Sides.C;
}

public double Area()
{
    if (!IsExist()) return 0;

    // Using Heron's formula
    double p = Perimeter() / 2;
    double a = Sides.A;
    double b = Sides.B;
    double c = Sides.C;
```

```
            return Math.Sqrt(p * (p - a) * (p - b) * (p - c));
        }

        public void CalculateAngles()
        {
            if (!IsExist())
            {
                Angles = new Parameters { A = 0, B = 0, C = 0 };
                return;
            }

            double a = Sides.A;
            double b = Sides.B;
            double c = Sides.C;

            double angleA = Math.Acos((b * b + c * c - a * a) / (2 * b * c)) * (180
/ Math.PI);

            double angleB = Math.Acos((a * a + c * c - b * b) / (2 * a * c)) * (180
/ Math.PI);

            double angleC = Math.Acos((a * a + b * b - c * c) / (2 * a * b)) * (180
/ Math.PI);

            double sum = angleA + angleB + angleC;
            if (Math.Abs(sum - 180) > 0.001)
            {
                double correction = (180 - sum) / 3;
                angleA += correction;
                angleB += correction;
                angleC += correction;
            }

            Angles = new Parameters { A = angleA, B = angleB, C = angleC };
        }

        public override string ToString()
        {
            if (!IsExist())
            {
                return "Triangle doesn't exist";
            }
            else
            {
                CalculateAngles();
                return $"Triangle with sides: {Sides}\nAngles: {Angles}\nPerimeter:
{Perimeter():F2}\nArea: {Area():F2}\nIsosceles: {Isosceles}";
            }
        }
    }

    public class IsoscelesTriangle : Triangle
    {
        public new bool IsIsosceles()
        {
            if (!IsExist()) return false;

            double a = Sides.A;
            double b = Sides.B;
            double c = Sides.C;

            return Math.Abs(a - b) < double.Epsilon ||
                   Math.Abs(a - c) < double.Epsilon ||
                   Math.Abs(b - c) < double.Epsilon;
        }
```

```
        public override string ToString()
        {
            return $"Isosceles Triangle with sides: {Sides}\nAngles:
{Angles}\nPerimeter: {Perimeter():F2}\nArea: {Area():F2}";
        }
    }
}
```

Описание кода

Взаимодействие файлов и слоёв приложения

1. Пользователь вводит значения в элементы интерфейса.
2. View фиксирует изменение и генерирует событие (или пользователь нажимает кнопку — событие команды).
3. Presenter получает событие, считывает значения из View.
4. Presenter валидирует ввод
5. Presenter передаёт данные в Model для выполнения предметных вычислений.
6. Model выполняет расчёт и возвращает результат (или признак ошибки).
7. Presenter формирует итоговый вывод.
8. Presenter передаёт сформированный вывод во View.
9. View отображает полученные данные пользователю.

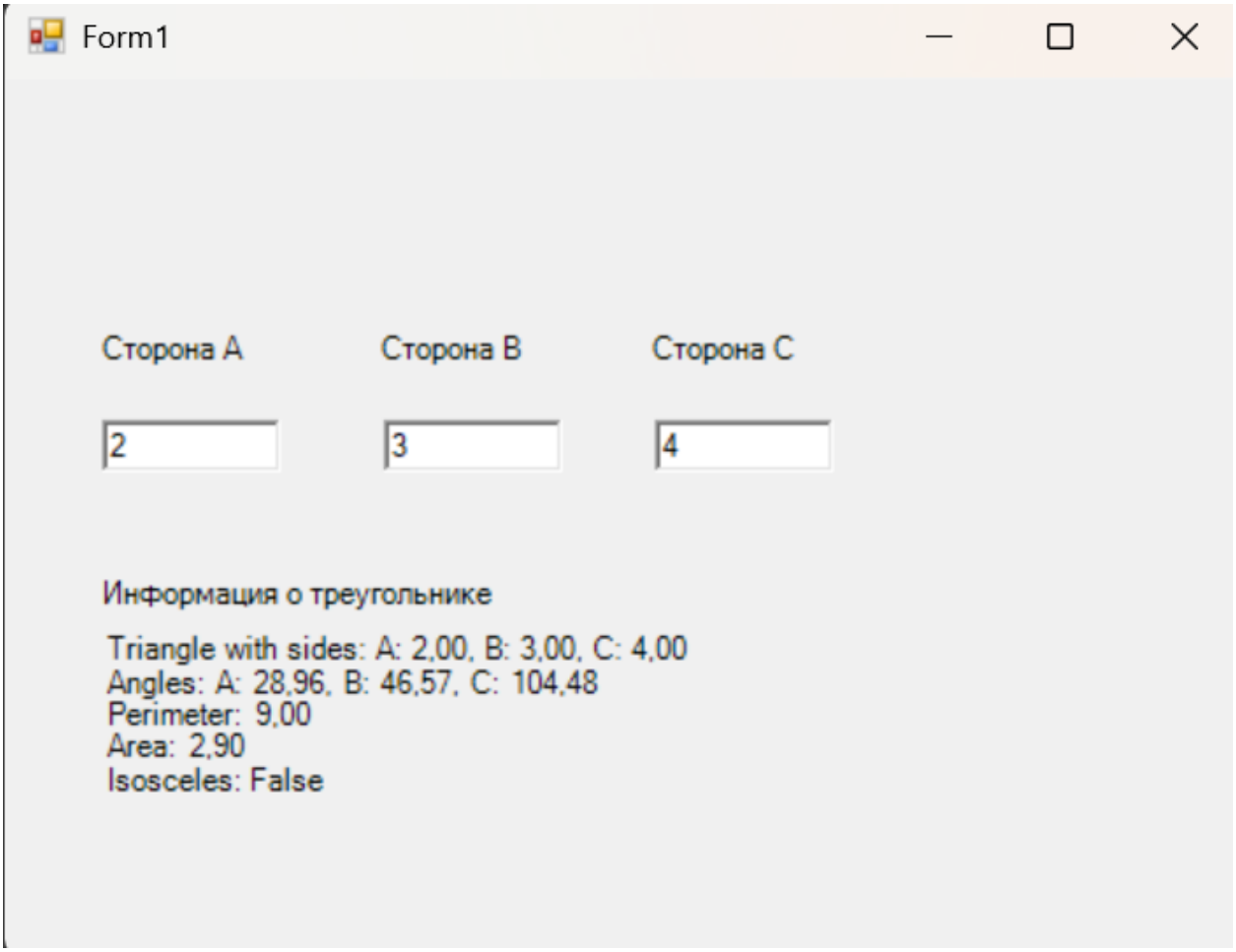
По итогу

View отвечает только за ввод данных пользователем и отображение результата. Никакой логики вычислений в нём нет — оно лишь передаёт данные дальше и показывает ответ.

Presenter получает данные из View, передаёт их в модель для вычислений и возвращает полученный результат обратно в View. Он управляет всей логикой взаимодействия.

Model выполняет все вычисления и проверки корректности данных. Она не знает ничего о пользовательском интерфейсе и используется только для обработки данных.

Результат



Введение данных египетского треугольника

Задание 2

Реализовать задание в соответствии с индивидуальным вариантом из лабораторной работы по классам — реализовать индивидуальное задание с использованием паттерна MVVM.

В качестве приложения для реализации паттерна MVVM была выбрана Лабораторная работа 1 по предмету МДК 04.01

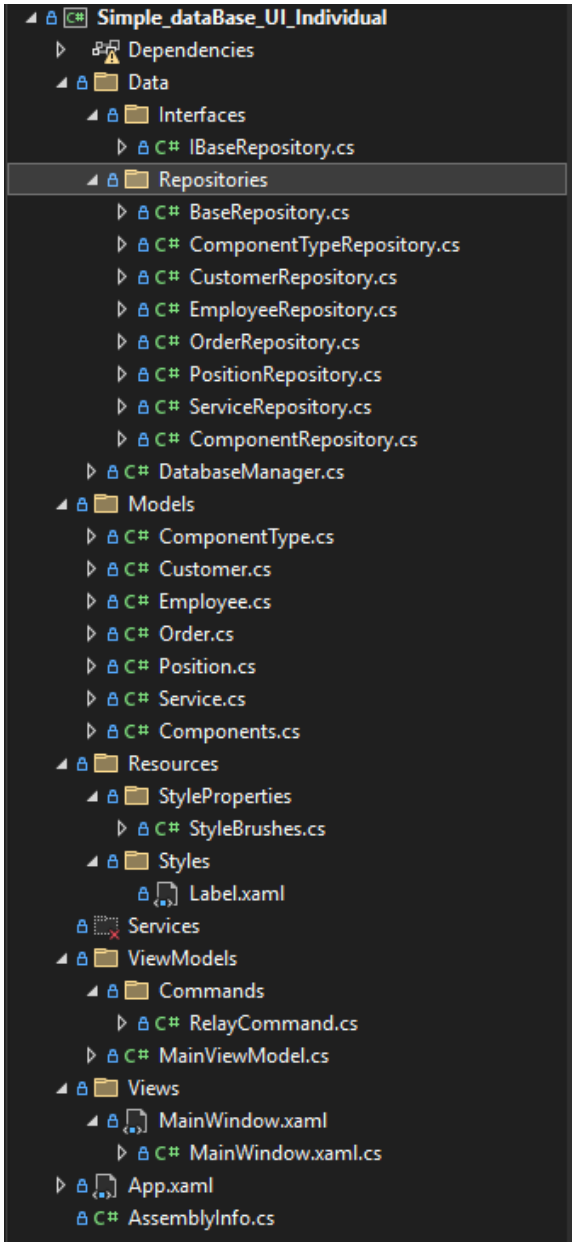
Клиентское приложение для взаимодействия с SQLite в соответствии с выданным вариантом.

1. Создать и связать любые три таблицы из индивидуального задания.
2. Показать, как работают следующие ограничения:

Вариант №19**:** БД Компьютерной фирмы.
Таблицы:
1) Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности).
2) Должности (Код должности, Наименование должности, Оклад, Обязанности,
Требования)

3) Виды комплектующих (Код вида, Наименование, Описание)
4) Комплектующие (Код комплектующего, Код вида, Марка, Фирма производитель, Страна производитель, Дата выпуска, Характеристики, Срок гарантия, Описание, Цена)
5) Заказчики (Код заказчика, ФИО, Адрес, Телефон).
6) Услуги (Код услуги, Наименование, Описание, Стоимость)
7) Заказы (Дата заказа, Дата исполнения, Код заказчика, Код комплектующего 1,
Код комплектующего 2, Код комплектующего 3, Доля предоплаты, Отметка об оплате, Отметка об исполнении, Общая стоимость, Срок общей гарантии, Код услуги 1, Код услуги 2, Код услуги 3, Код сотрудника).
Запросы:
Отобразить заказы отдельных заказчиков;
Отобразить комплектующие определенного производителя.

Структура проекта:



Структура проекта

Код MainWindow.xaml

Файл XAML-разметки главного окна приложения.

Определяет элементы интерфейса:

- ListBox — список таблиц базы данных;
- DataGrid — отображение и редактирование данных выбранной таблицы;
- привязки (Binding) к свойствам MainViewModel.

Все визуальные элементы связаны с данными через механизм привязки WPF.

```
<Window x:Class="Simple_dataBase_UI_Individual.Views.MainWindow"
```

```
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Simple_dataBase_UI_Individual.Views"
        xmlns:vm="clr-namespace:Simple_dataBase_UI_Individual.ViewModels"
        xmlns:i="http://schemas.microsoft.com/xaml/behaviors"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Window.DataContext>
        <vm:MainViewModel/>
    </Window.DataContext>
    <Border
        Background="DarkSlateGray"
        >

    <Grid Grid.RowSpan="1" Grid.Row="1" Grid.ColumnSpan="2">
        <Grid.RowDefinitions>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="auto"/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="auto"/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>



        <Label  Content="Tables"
                Style="{StaticResource S_Header}"
                ></Label>
        <Label  Content="Data"
                Grid.Column="1"
                Style="{StaticResource S_Header}"
                ></Label>

        <ListBox Grid.Row="1"
                 ItemsSource="{Binding Tables}"
                 SelectedItem="{Binding SelectedTable}"
                 Background="SlateGray"
                 Foreground="White"
                 FontWeight="DemiBold"
                 >
            <i:Interaction.Triggers>
                <i:EventTrigger EventName="SelectionChanged">
                    <i:InvokeCommandAction Command="{Binding
SelectionTableCommand}"></i:InvokeCommandAction>
                </i:EventTrigger>
            </i:Interaction.Triggers>
        </ListBox>



        <DataGrid Grid.Column="1" Grid.RowSpan="2" Grid.Row="1"
                  ItemsSource="{Binding DataTableC}"
                  Background="SlateGray"
                  Foreground="Black"
                  >
            <i:Interaction.Triggers>
                <i:EventTrigger EventName="CellEditEnding">
                    <i:InvokeCommandAction Command="{Binding CellEditEndingCommand}"
                                PassEventArgsToCommand="True"/>
                </i:EventTrigger>
```

```
                <i:EventTrigger EventName="RowEditEnding">
                    <i:InvokeCommandAction Command="{Binding RowEditEndingCommand}"
                                           PassEventArgsToCommand="True"/>
                </i:EventTrigger>
            </i:Interaction.Triggers>
        </DataGrid>
        </Grid>
    </Border>
</Window>
```

Код IBaseRepository.cs

Интерфейс репозитория, определяющий базовый контракт для работы с данными.

Содержит объявления стандартных CRUD-операций:

- получение всех записей;
- добавление;
- обновление;
- удаление;
- сохранение изменений.

Интерфейс обеспечивает единообразие работы с различными сущностями базы данных.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Simple_dataBase_UI_Individual.Data.Interfaces
{
    public interface IBaseRepository<T> where T : class
    {
        DataTable GetAll();
        void Add(T entity);
        void Update(T entity);
        void Delete(int id);
        void Save();
    }
}
```

Код BaseRepository

Базовый абстрактный класс репозитория, реализующий общую логику взаимодействия с SQLite:

- хранит подключение к базе данных;

- выполняет SQL-запросы;
- реализует метод GetAll() для получения данных в виде DataTable.

Данный класс не зависит от конкретной модели и используется как основа для специализированных репозиториев.

```
using Simple_dataBase_UI_Individual.Data;
using Simple_dataBase_UI_Individual.Data.Interfaces;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SQLite;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
namespace Simple_dataBase_UI_Individual.Data.Repositories
{
    public abstract class BaseRepository<T> : IBaseRepository<T> where T : class
    {
        protected readonly string tableName;
        public BaseRepository()
        {
            this.tableName = typeof(T).Name;
        }
        public T CreateInstance()
        {
            return Activator.CreateInstance<T>();
        }
        public virtual DataTable GetAll()
        {
            DataTable dataTable = new DataTable();
            try
            {
                string sqlQuery = $"SELECT * FROM \"{tableName}\"";
                using (SQLiteDataAdapter adapter = new SQLiteDataAdapter(sqlQuery,
DatabaseManager.m_dbConn))
                {
                    adapter.Fill(dataTable);
                }
                Console.WriteLine($"Retrieved {dataTable.Rows.Count} rows from
{tableName}");
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error getting all from {tableName}:
{ex.Message}");
                MessageBox.Show($"Error loading data from {tableName}:
{ex.Message}");
            }
            return dataTable;
        }
        public virtual T CreateInstanceFromDataRow(DataRow row) { throw new
NotImplementedException(); }
        public virtual void Add(T entity){throw new NotImplementedException();}
        public virtual void Delete(int id) { throw new NotImplementedException(); }
        public virtual void Save() { throw new NotImplementedException(); }
        public virtual void Update(T entity) { throw new NotImplementedException();
}
    }
}
```

Код ComponentTypeRepository

Конкретная реализация репозитория для сущности ComponentType.

Наследуется от BaseRepository<ComponentType> и реализует:

- преобразование строк таблицы (DataRow) в объекты ComponentType;
- SQL-команды INSERT, UPDATE, DELETE;
- проверку существования записи по идентификатору.

Таким образом, данный репозиторий полностью инкапсулирует логику работы с таблицей «ComponentType» в базе данных SQLite.

Остальные репозитории организованы по подобному принципу

```
using Simple_dataBase_UI_Individual.Data;
using Simple_dataBase_UI_Individual.Data.Interfaces;
using Simple_dataBase_UI_Individual.Models;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SQLite;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;



//  Виды комплектующих(
//          Код вида,
//          Наименование,
//          Описание)



namespace Simple_dataBase_UI_Individual.Data.Repositories
{
    public class ComponentTypeRepository : BaseRepository<ComponentType>
    {
        public ComponentTypeRepository(string dbFilePath) { }



        public override ComponentType CreateInstanceFromDataRow(DataRow row)
        {
            try
            {
                var componentType = new ComponentType();



                if (!row.IsNull("id") && row["id"] != DBNull.Value) {
componentType.Id = Convert.ToInt32(row["id"]); }
```

```
                else { componentType.Id = 0; }



                componentType.Name = row.IsNull("name") ? "" :
row["name"].ToString();
                componentType.Description = row.IsNull("de  scription") ? "" :
row["description"].ToString();



                return componentType;
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error creating ComponentType from DataRow:
{ex.Message}");
                throw;
            }
        }



        public bool IsIdExists(int id)
        {
            try
            {
                using (var command = new SQLiteCommand(DatabaseManager.m_dbConn))
                {
                    command.CommandText = "SELECT COUNT(*) FROM ComponentType WHERE
id = @id";
                    command.Parameters.AddWithValue("@id", id);



                    int count = Convert.ToInt32(command.ExecuteScalar());
                    return count > 0;
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error checking ID existence: {ex.Message}");
                return false;
            }
        }



        public override void Add(ComponentType entity)
        {
            try
            {
                // Проверяем, не существует ли уже запись с таким ID
                if (entity.Id != 0 && IsIdExists(entity.Id))
                {
                    MessageBox.Show($"ComponentType with ID {entity.Id} already
exists!");
                    return;
                }
```

```
                using (var command = new SQLiteCommand(DatabaseManager.m_dbConn))
                {
                    if (entity.Id != 0)
                    {
                        command.CommandText = @"
                            INSERT INTO ComponentType (id, name, description)
                            VALUES (@id, @name, @description)";
                        command.Parameters.AddWithValue("@id", entity.Id);
                    }
                    else
                    {
                        command.CommandText = @"
                            INSERT INTO ComponentType (name, description)
                            VALUES (@name, @description)";
                    }



                    command.Parameters.AddWithValue("@name", entity.Name ?? "");
                    command.Parameters.AddWithValue("@description",
entity.Description ?? "");



                    int rowsAffected = command.ExecuteNonQuery();
                    Console.WriteLine($"Rows affected: {rowsAffected}");



                    // Если ID не был указан, получаем сгенерированный ID
                    if (entity.Id == 0)
                    {
                        command.CommandText = "SELECT last_insert_rowid()";
                        entity.Id = Convert.ToInt32(command.ExecuteScalar());
                    }
                }
            }
            catch (SQLiteException ex)
            {
                MessageBox.Show("Error adding component type: " + ex.Message);
            }
        }



        public override void Update(ComponentType entity)
        {
            try
            {
                using (var command = new SQLiteCommand(DatabaseManager.m_dbConn))
                {
                    command.CommandText = @"
                        UPDATE ComponentType
                        SET name = @name,
                            description = @description
                        WHERE id = @id";



                    command.Parameters.AddWithValue("@name", entity.Name ?? "");
                    command.Parameters.AddWithValue("@description",
entity.Description ?? "");
```

```
                    command.Parameters.AddWithValue("@id", entity.Id);



                    command.ExecuteNonQuery();
                }
            }
            catch (SQLiteException ex)
            {
                MessageBox.Show("Error updating component type: " + ex.Message);
            }
        }
    }
}
```

Код ComponentType.cs

Содержит модель сущности базы данных «Тип комплектующего».

Класс ComponentType представляет одну запись таблицы и включает свойства:

- Id — уникальный идентификатор записи;
- Name — наименование типа комплектующего;
- Description — описание.

Модель не содержит логики работы с базой данных и используется для переноса данных между слоями приложения.

Остальные модели организованы подобным образов

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;



namespace Simple_dataBase_UI_Individual.Models
{
    //  Виды комплектующих(
    //          Код вида,
    //          Наименование,
    //          Описание)
    public class ComponentType
    {
        public ComponentType() { }
        public ComponentType(int Id, string Name, string Description) {
            this.Id = Id;
            this.Name = Name;
            this.Description = Description;
        }
        public ComponentType(List<object> array)
        {
            if (array == null || array.Count < 3)
                throw new ArgumentException("Array must contain at least 3
elements");
```

```
            this.Id = Convert.ToInt32(array[0]);
            this.Name = array[1]?.ToString() ?? string.Empty;
            this.Description = array[2]?.ToString() ?? string.Empty;
        }
        public int Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
    }
}
```

Код RelayCommand

Реализует интерфейс ICommand для использовани команд в интерфейсе

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
//using Simple_dataBase_UI_Individual.Models;



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



        public RelayCommand(Action<object> execute, Func<object, bool> canExecute =
null)
        {
            _execute = execute ?? throw new ArgumentNullException(nameof(execute));
            _canExecute = canExecute;
        }
        public bool CanExecute(object parameter)
        {
            return _canExecute == null || _canExecute(parameter);
        }



        public void Execute(object parameter)
        {
            _execute(parameter);
        }



        public event EventHandler CanExecuteChanged
        {
            add { CommandManager.RequerySuggested += value; }
            remove { CommandManager.RequerySuggested -= value; }
```

```
        }
    }
}
```

Код MainViewModel.cs

Главная ViewModel приложения. Выполняет следующие функции:

- хранит список доступных таблиц базы данных;
- отслеживает выбранную пользователем таблицу;
- инициализирует соответствующий репозиторий;
- загружает данные из базы данных и предоставляет их представлению;
- обрабатывает события редактирования данных в таблице.

ViewModel не содержит SQL-запросов напрямую, а работает исключительно через слой репозиториев.

```
using Simple_dataBase_UI_Individual.Data;
using Simple_dataBase_UI_Individual.Data.Interfaces;
using Simple_dataBase_UI_Individual.Data.Repositories;
using Simple_dataBase_UI_Individual.ViewModels.Commands;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Data;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Threading;



namespace Simple_dataBase_UI_Individual.ViewModels
{
    class MainViewModel : INotifyPropertyChanged
    {
        private object _selectedTable;
        public object SelectedTable
        {
            get { return _selectedTable; }
            set {
                _selectedTable = value;
                OnPropertyChanged();
            }
        }



        private DataTable _dataTable;
        public DataTable DataTableC
        {
```

```
    get { return _dataTable; }
    set { _dataTable = value;
        OnPropertyChanged();
    }
}
public ObservableCollection<string> Tables { get; set; }



private ObservableCollection<object> _items;
public ObservableCollection<object> Items
{
    get { return _items; }
    set { _items = value;
        OnPropertyChanged();
    }
}



private readonly Dictionary<string, object> _repositories;



public MainViewModel()
{
    DatabaseManager.getInstance("testdb.sqlite");



    _repositories = new Dictionary<string, object> {
        { "ComponentType", new ComponentTypeRepository("testdb")},
        { "Employee", new EmployeeRepository("testdb")},
        { "Order", new OrderRepository("testdb")},
        { "Position", new PositionRepository("testdb")},
        { "Service", new ServiceRepository("testdb")},
        { "Component", new ComponentRepository("testdb")},
        { "Customer", new CustomerRepository("testdb")}
        };



    SelectionTableCommand = new RelayCommand(SelectionTable);
    SelectionChangedCommand = new RelayCommand(SelectionChanged);
    CellEditEndingCommand = new RelayCommand(CellEditEnding);
    RowEditEndingCommand = new RelayCommand(RowEditEnding);




    Tables = new ObservableCollection<string>(_repositories.Keys);
}
public ICommand SelectionTableCommand { get; }
public ICommand SelectionChangedCommand { get; }
public ICommand CellEditEndingCommand { get; }
public ICommand RowEditEndingCommand { get; }
public void CellEditEnding(object parameter)
{
```

```
        }
        public void SelectionChanged(object parameter)
        {



        }
        public void RowEditEnding(object parameter)
        {
            if (parameter is DataGridRowEditEndingEventArgs e)
            {
                if (e.EditAction == DataGridEditAction.Commit)
                {
                    if (e.Row.DataContext is DataRowView dataRowView)
                    {
                        Dispatcher dispatcher =
System.Windows.Application.Current.Dispatcher;
                        Action myAction = delegate ()
                        {
                            ProcessRowEdit(dataRowView.Row);
                        };
                        dispatcher.BeginInvoke(myAction,
DispatcherPriority.Background);
                    }
                }
            }
        }
        private void ProcessRowEdit(DataRow dataRow)
        {
            try
            {
                bool isNewRow = dataRow.RowState == DataRowState.Added ||
                                dataRow.RowState == DataRowState.Detached ||
                                dataRow.IsNull("id") ||
                                dataRow["id"] == DBNull.Value ||
                                Convert.ToInt32(dataRow["id"]) == 0;



                dynamic repository = _repositories[SelectedTable.ToString()];
                var entity = repository.CreateInstanceFromDataRow(dataRow);



                if (isNewRow)
                {
                    repository.Add(entity);
                    dataRow["id"] = entity.Id;
                }
                else
                {
                    repository.Update(entity);
                }
                Dispatcher dispatcher =
System.Windows.Application.Current.Dispatcher;
                Action myAction = delegate ()
                {
                    RefreshDataTable(repository);
                };
                dispatcher.BeginInvoke(myAction,
DispatcherPriority.ApplicationIdle);
            }
            catch (Exception ex) { }
```

```
        }



        private void RefreshDataTable(dynamic repository)
        {
            var newDataTable = repository.GetAll();
            DataTableC = newDataTable;
        }
        private void SelectionTable(object parameter)
        {
            dynamic repository = _repositories[SelectedTable.ToString()];
            RefreshDataTable(repository);
        }



        public event PropertyChangedEventHandler? PropertyChanged;
        protected virtual void OnPropertyChanged([CallerMemberName] string
propertyName = null)
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Описание кода

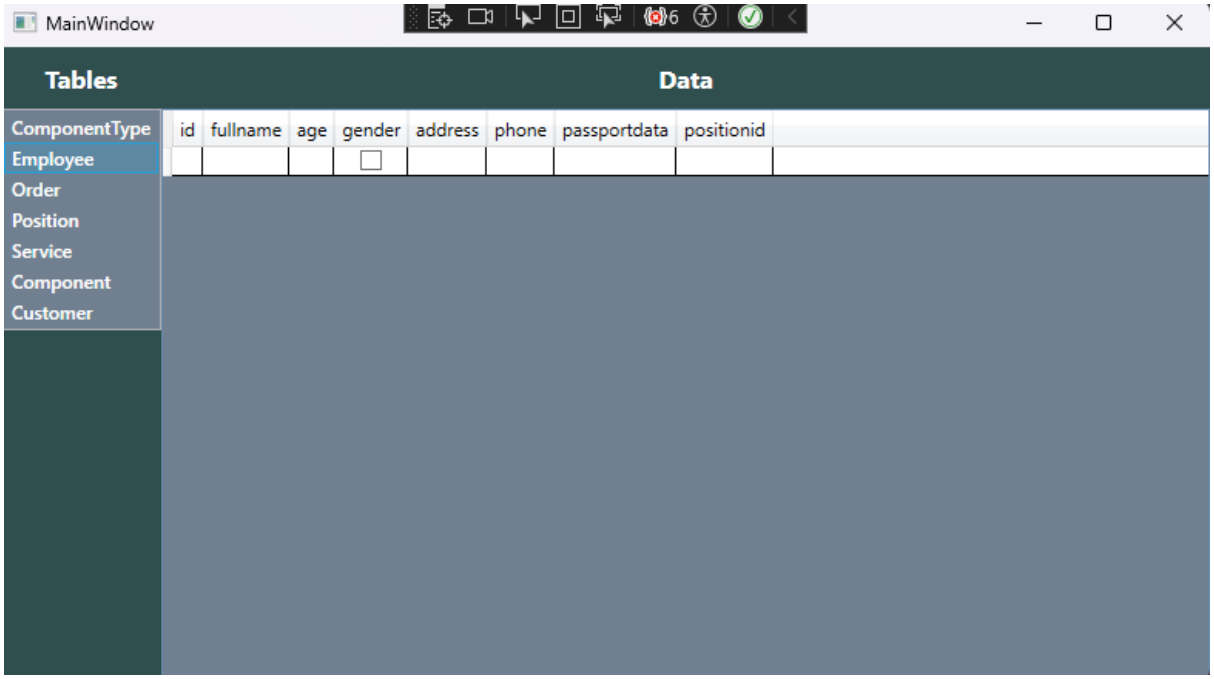
Взаимодействие файлов и слоёв приложения

1. Представление (View) (MainWindow.xaml) устанавливает объект MainViewModel в качестве DataContext, тем самым связывая элементы пользовательского интерфейса со свойствами и командами ViewModel.
2. Пользователь выбирает таблицу базы данных в элементе управления интерфейса (ListBox).
3. MainViewModel анализирует выбранное значение и определяет, какой конкретный репозиторий соответствует данной таблице (например, ComponentTypeRepository).
4. ViewModel обращается к выбранному репозиторию и вызывает метод GetAll() для получения данных.
5. Репозиторий выполняет SQL-запрос к базе данных SQLite, используя активное подключение к БД.
6. Результат SQL-запроса извлекается в виде набора строк, которые:
    - либо преобразуются в объекты модели (например, ComponentType),

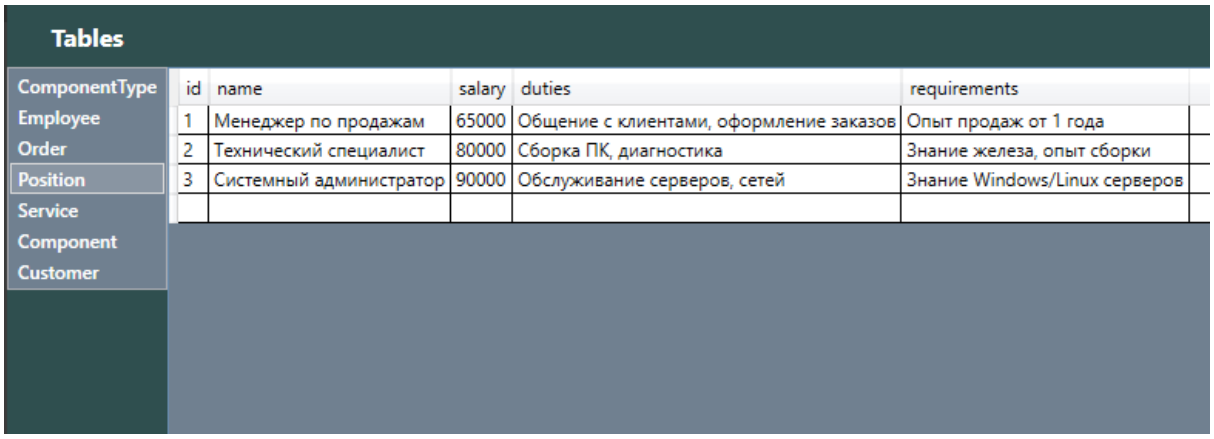
- либо формируются в структуру DataTable, связанную с моделью по составу полей.

7. Модель (ComponentType) на данном этапе выступает как представление сущности предметной области:
   каждая строка таблицы базы данных логически соответствует экземпляру модели, содержащему строго типизированные свойства (Id, Name, Description).
8. Репозиторий возвращает данные (модели или DataTable, сформированный на их основе) обратно во ViewModel.
9. ViewModel сохраняет полученные данные в своих свойствах и предоставляет их представлению через механизм привязки данных (Binding).
10. Представление (View) отображает данные в элементе DataGrid и предоставляет пользователю возможность их редактирования.
11. При добавлении или изменении строки пользовательским интерфейсом ViewModel формирует или обновляет соответствующий объект модели (ComponentType) на основе введённых данных.
12. ViewModel передаёт объект модели в репозиторий, вызывая методы Add() или Update().
13. Репозиторий, используя данные модели, формирует параметризованные SQL-запросы и сохраняет изменения в базе данных SQLite.

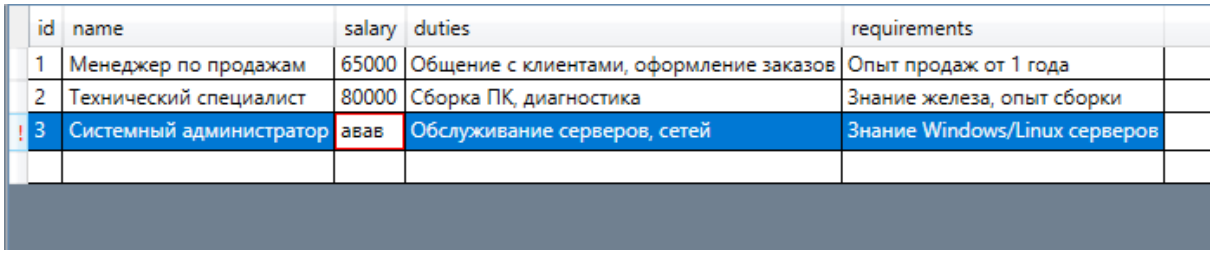
Результат



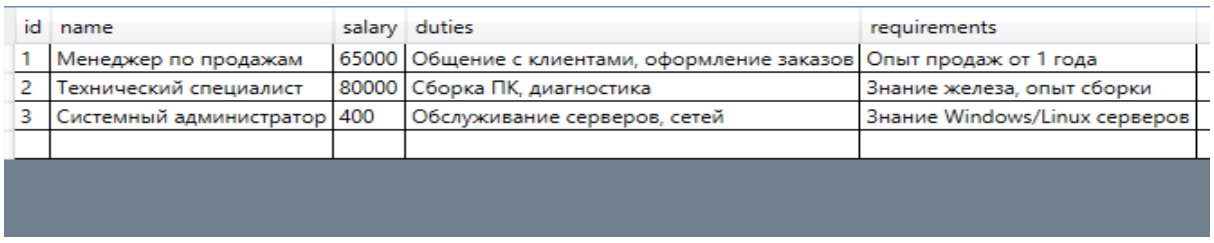
Интерфейс приложения



Пример заполнения таблицы

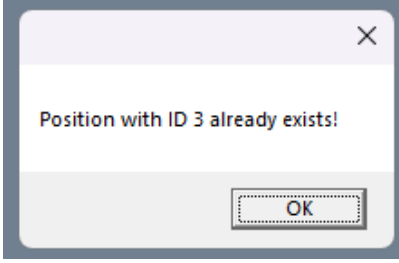


Пример попытки внесения некорректных данных



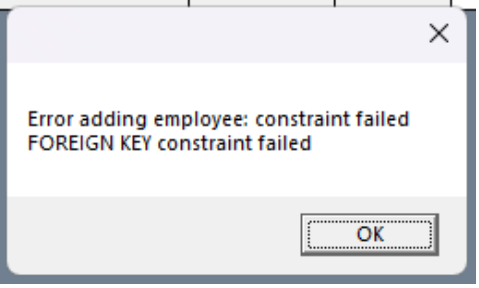
Пример изменения таблицы

id	name	salary	duties	requirements
1	Менеджер по продажам	65000	Общение с клиентами, оформление заказов	Опыт продаж от 1 года
2	Технический специалист	80000	Сборка ПК, диагностика	Знание железа, опыт сборки
3	Системный администратор	90000	Обслуживание серверов, сетей	Знание Windows/Linux серверов
3				



Попытка создание записи с не уникальным ключом

id	fullname	age	gender	address	phone	passportdata	positionid
101	Иванов Алексей Петрович	28	☑	Москва, ул. Ленина, 15	+7(495)111-22-33	4510 123456	76
			☐				



Попытка создания записи в таблицы с несуществующим Foreign Key

Tables

ComponentType
Employee
Order
Position
Service
Component
Customer

id	fullname	age	gender	address	phone	passportdata	positionid
101	Иванов Алексей Петрович	28	☑	Москва, ул. Ленина, 15	+7(495)111-22-33	4510 123456	1
102	Петрова Мария Сергеевна	32	☐	Москва, пр. Мира, 28	+7(495)222-33-44	4510 654321	2
103	Сидоров Дмитрий Иванович	35	☑	Москва, ул. Пушкина, 10	+7(495)333-44-55	4510 789012	3
			☐				

Пример заполнения таблицы с Foreign Key

Tables

ComponentType
Employee
Order
Position
Service
Component
Customer

id	name	description	price
1	Диагностика компьютера	Полная проверка оборудования и программного обеспечения	1500
2	Сборка ПК на заказ	Комплексная сборка компьютера из выбранных комплектующих	3000
3	Установка ОС	Установка и настройка операционной системы Windows/Linux	2000

Пример заполнения таблицы Service