Разбор Модуля 2. Разработка программных модулей

Оглавление

[Создание проекта 2](#_Toc194428977)

[Добавление пакетов в проект 4](#_Toc194428978)

[Microsoft.EntityFrameworkCore.Tools 4](#_Toc194428979)

[Microsoft.EntityFrameworkCore.Design 4](#_Toc194428980)

[FirebirdSql.EntityFrameworkCore.Firebird 4](#_Toc194428981)

[Avalonia.ReactiveUI 5](#_Toc194428982)

[Тестовый запуск приложения 6](#_Toc194428983)

[Формирования классов сущностей 8](#_Toc194428984)

[Необходимые компоненты: 8](#_Toc194428985)

[Обязательные аргументы 8](#_Toc194428986)

[Connection string 8](#_Toc194428987)

[Имя поставщика 9](#_Toc194428988)

[Добавление базы данных 9](#_Toc194428989)

[Настройка навигации 10](#_Toc194428990)

[MainWindowViewModel.cs 11](#_Toc194428991)

[MainWindow.axaml 12](#_Toc194428992)

[Создание пользовательского элемента управления 13](#_Toc194428993)

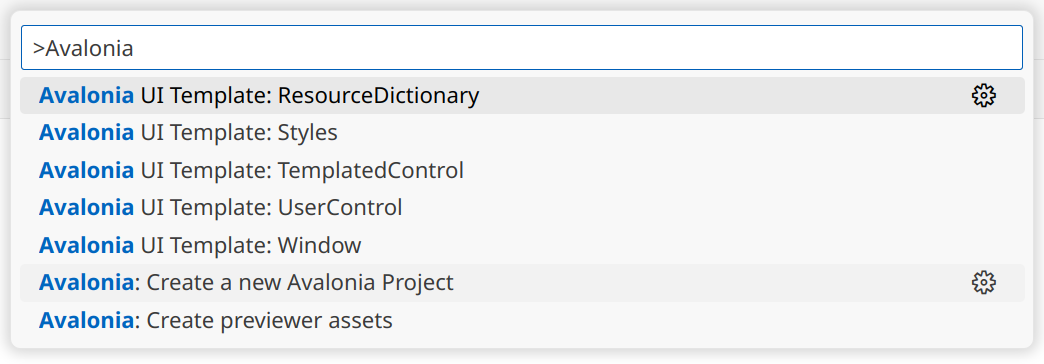
[Настройка ListBox 19](#_Toc194428994)

# **СОЗДАНИЕ ПРОЕКТА**

**1 Вариант:**

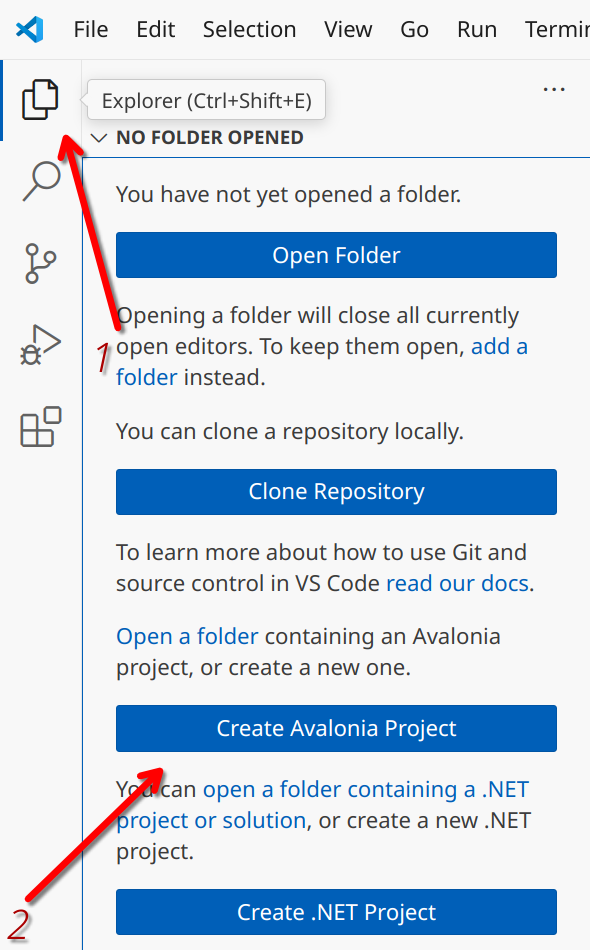
Для начала откроем Visual Studio Code, нажмём сочетание клавиш **Ctrl+Shift+P.**

В появившемся окне вводим Avalonia и выбираем пункт **Avalonia: Create a new Avalonia Project**:

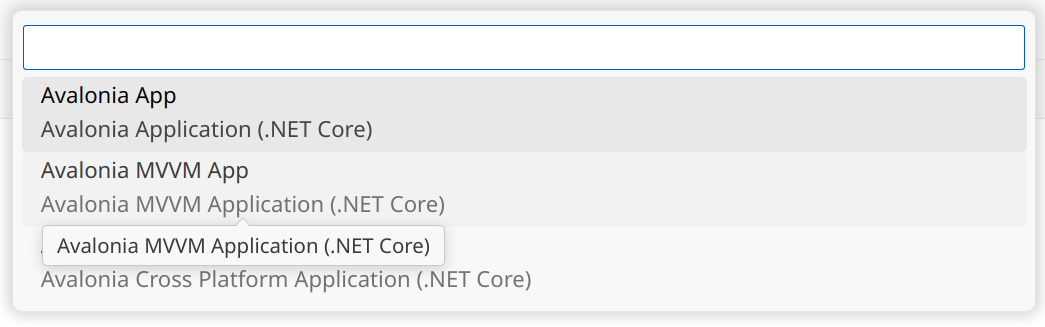


**2 Вариант:**

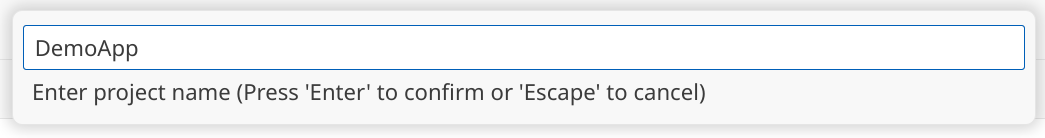
Необходимо в боковом меню выбрать пункт **Explorer** (1)и далее выбрать пункт **Create Avalonia Project** (2):



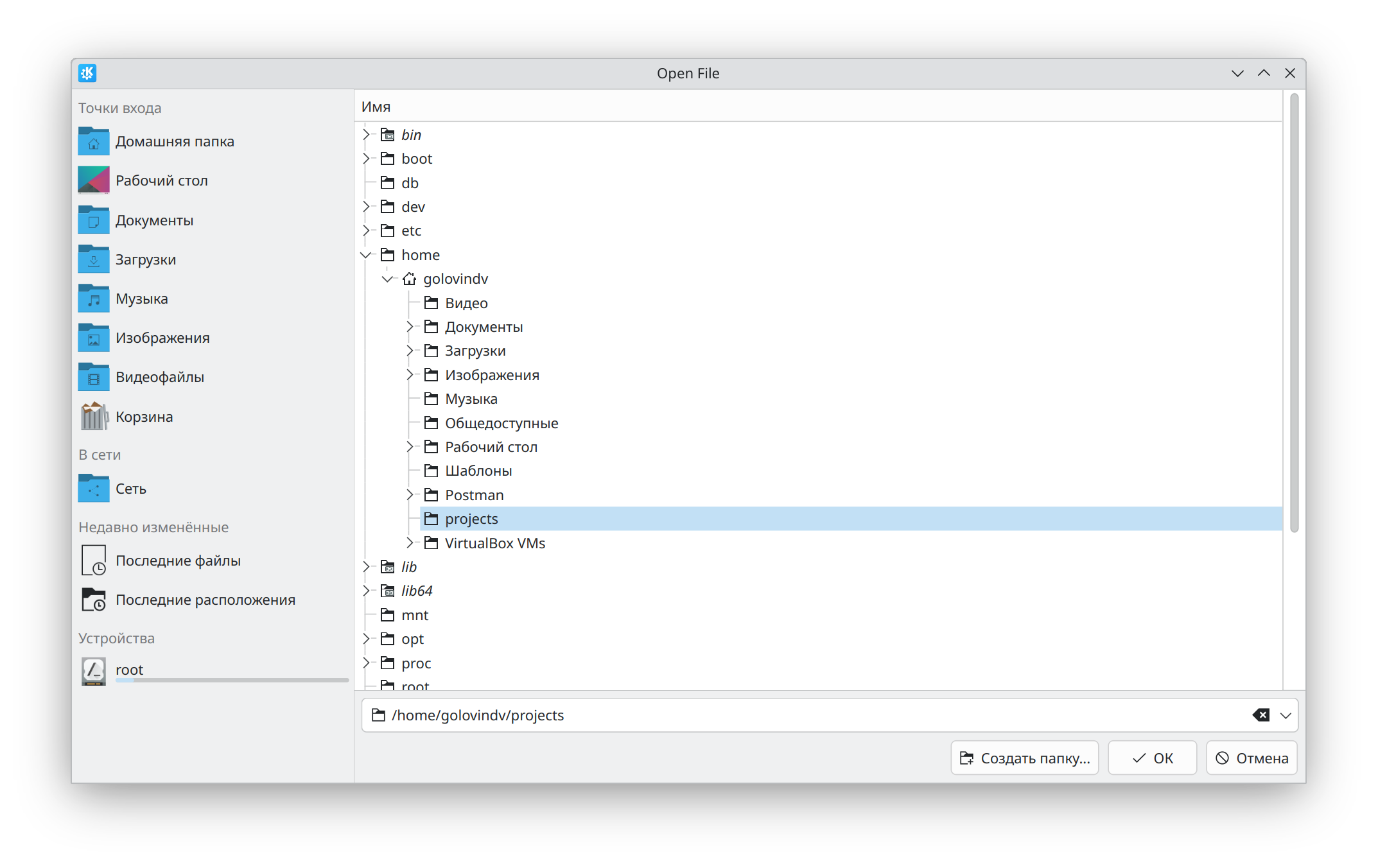
Далее выбираем пункт **Avalonia MVVM App**:



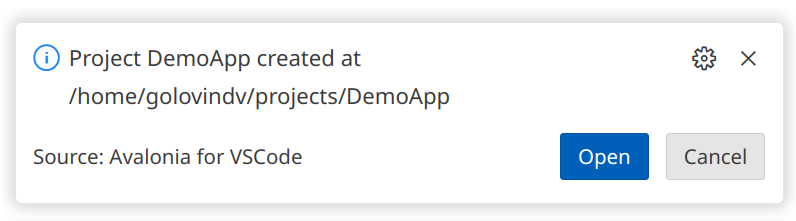
Вводим название нашего проекта **DemoApp**:



После выбираем директорию, где необходимо сохранить наш проект:



После создания проекта у нас должно появиться уведомление с предложением открыть нашу папку в редакторе. Нажимаем на **Open**:



# **ДОБАВЛЕНИЕ ПАКЕТОВ В ПРОЕКТ**

Для работы и построения ORM модели и связи приложения с базой данных нам необходимы следующие пакеты (команды необходимо выполнять в консоли в корне приложения), с сайта <https://www.nuget.org:>

## Microsoft.EntityFrameworkCore.Tools

Microsoft.EntityFrameworkCore.Tools помогают решать задачи, связанные с разработкой на этапе проектирования. В основном они используются для управления миграциями и создания DbContext и типов сущностей путём обратного проектирования схемы базы данных.

Этот пакет Microsoft.EntityFrameworkCore.Tools предназначен для инструментов PowerShell, которые работают в консоли диспетчера пакетов Visual Studio (PMC).

Добавление пакета:

dotnet add package Microsoft.EntityFrameworkCore.Tools --version 9.0.3

## Microsoft.EntityFrameworkCore.Design

Пакет Microsoft.EntityFrameworkCore.Design необходим для инструментов командной строки или консоли диспетчера пакетов и является зависимостью от dotnet-ef и Microsoft.EntityFrameworkCore.Tools.

Добавление пакета:

dotnet add package Microsoft.EntityFrameworkCore.Design --version 9.0.3

## FirebirdSql.EntityFrameworkCore.Firebird

Поставщик Entity Framework Core для Firebird позволяет разрабатывать приложения .NET, которые подключаются к базе данных Firebird с помощью Entity Framework Core.

Добавление пакета:

dotnet add package FirebirdSql.EntityFrameworkCore.Firebird --version 12.0.0

## Avalonia.ReactiveUI

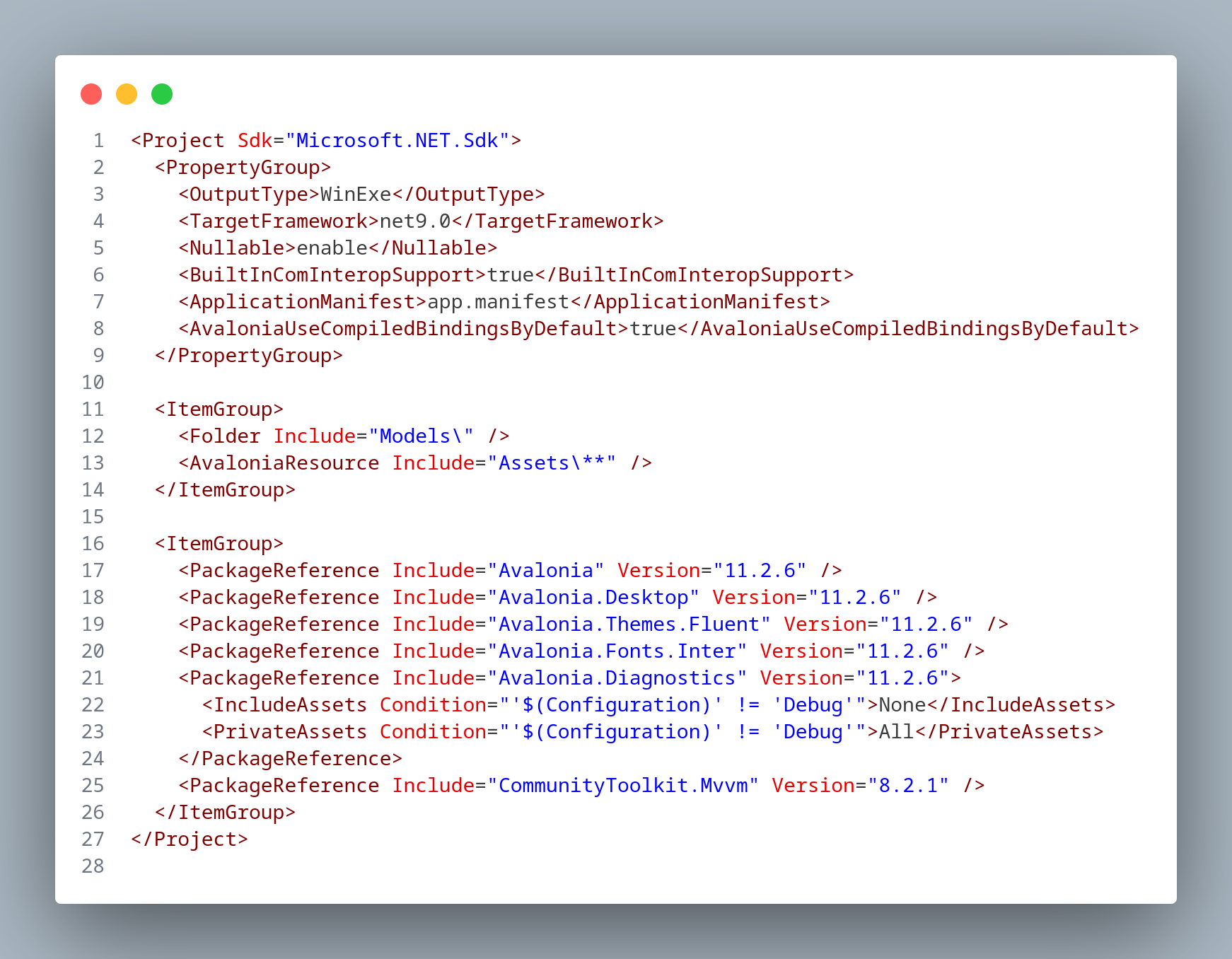
ReactiveUI — это компонуемый кроссплатформенный фреймворк для всех платформ .NET, основанный на функциональном реактивном программировании.

Добавление пакета:

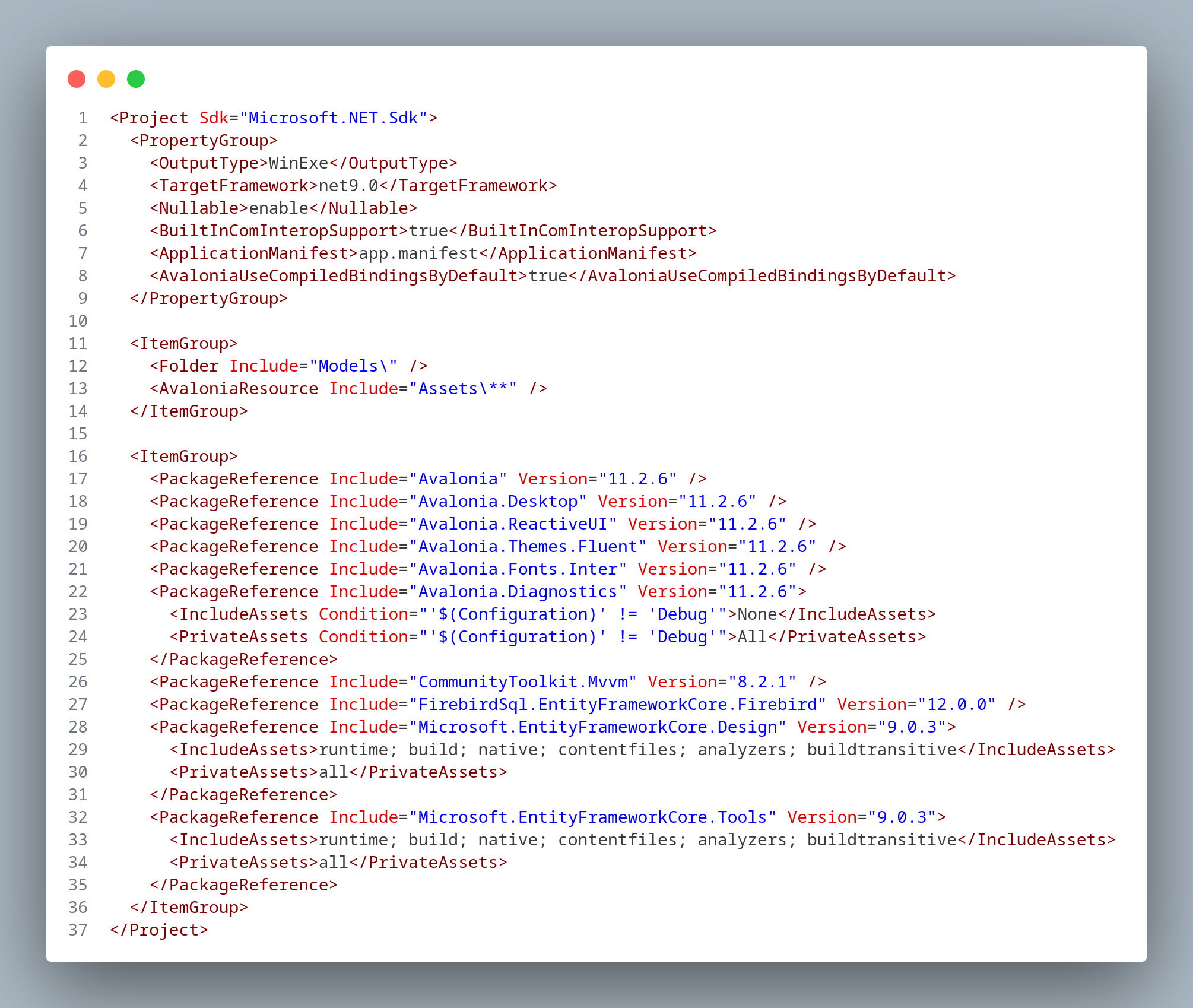
dotnet add package Avalonia.ReactiveUI --version 11.2.6

После выполнения всех команд, необходимо открыть файл DemoApp.csproj и вы заметите добавленные пакеты.

Было:

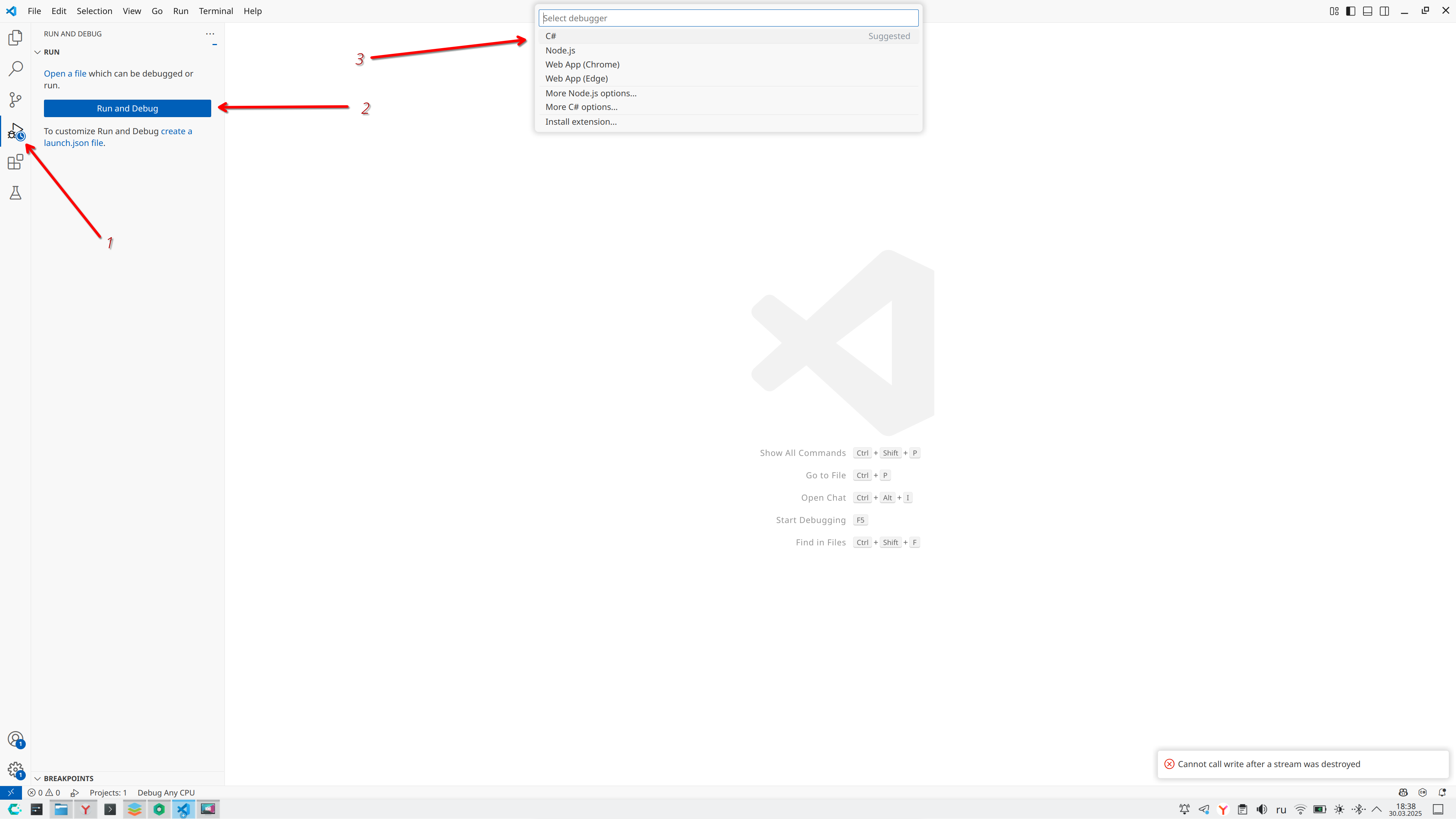


Стало:

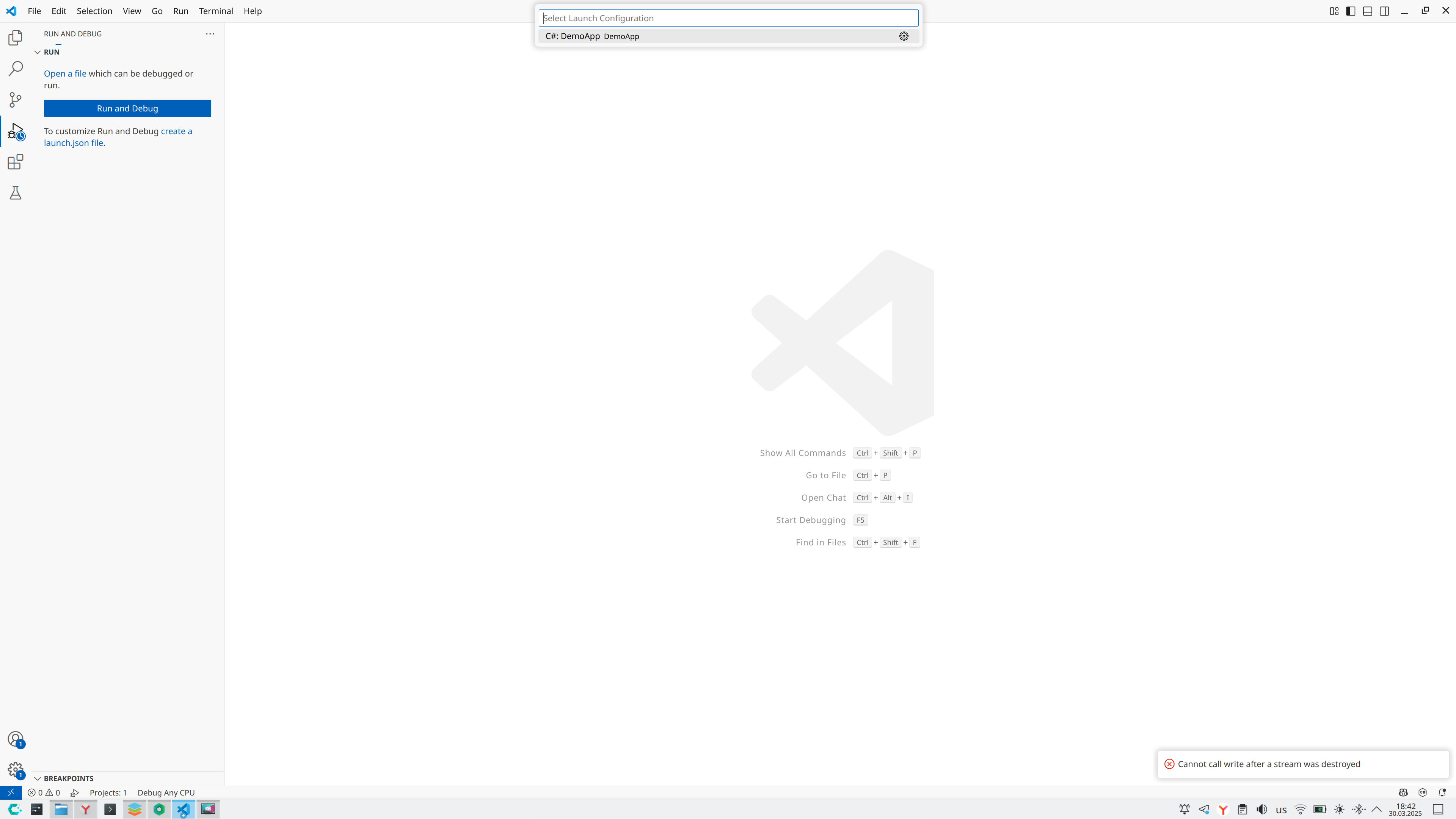


# ТЕСТОВЫЙ ЗАПУСК ПРИЛОЖЕНИЯ

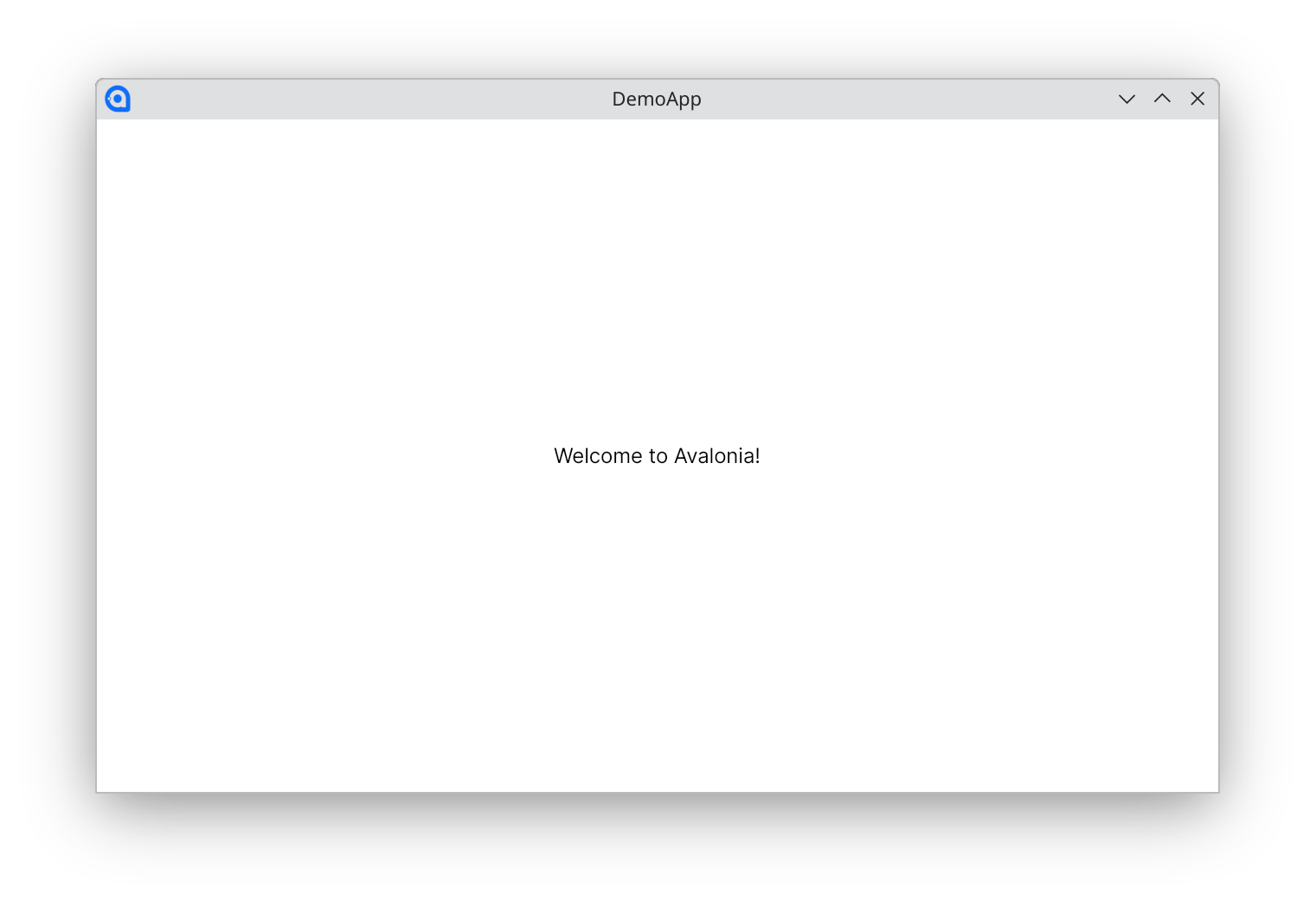
Для запуска приложения в боковом меню нам необходимо выбрать пункт **Run and Debug** (1) далее на открывшейся панели необходимо нажать на кнопку **Run and Debug** (2) и выбрать **C#** (3):



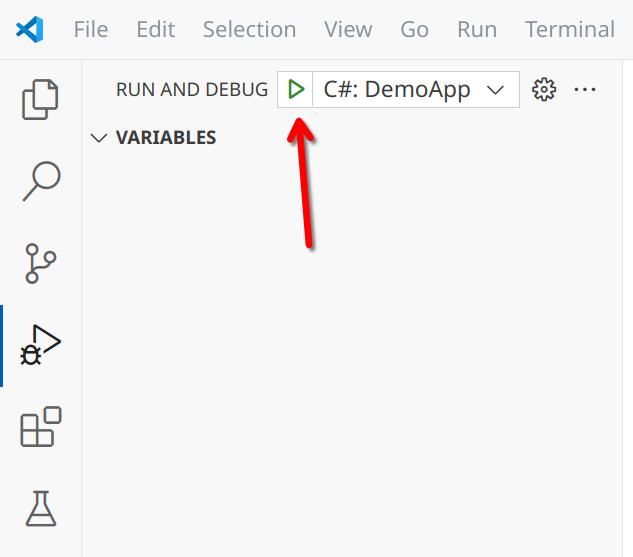
Далее необходимо выбрать наш проект:



Если все хорошо, то мы увидим следующее окно:



При последующих запусках наша панель **Run and Debug** будет иметь следующий вид:



Для последующих запусков необходимо нажимать на зеленую кнопку (F5).

# ФОРМИРОВАНИЯ КЛАССОВ СУЩНОСТЕЙ

Обратная инженерия — это процесс формирования классов типов сущностей и класса DbContext на основе схемы базы данных. Этот процесс можно выполнить с помощью команды **dotnet ef dbcontext scaffold** средств интерфейса командной строки (CLI) .NET.

### Необходимые компоненты:

* Перед формированием шаблонов необходимо установить средства .NET CLI, которые на всех платформах, поддерживаемых .NET.
* Установите пакет NuGet для Microsoft.EntityFrameworkCore.Design в проекте, в котором будут формироваться шаблоны.
* Установите пакет NuGet для поставщика базы данных, предназначенного для схемы базы данных, из которой требуется создать шаблон.

### Обязательные аргументы

Команда.NET CLI имеют два обязательных аргумента: строка подключения в базу данных и поставщик базы данных EF Core для использования.

### Connection string

Первый аргумент команды — строка подключения к базе данных. Средства используют эту строка подключения для чтения схемы базы данных.

Пример строки подключения:

"DataSource=localhost;Port=3050;Database=/db/demo.fdb;Username=sysdba;Password=123321"

DataSource – источник данных, т.к. база данных находится на ПК с приложением указываем localhost.

Port – порт на котором работает Red Database, совпадает с портом для Firebase и по умолчанию 3050.

Database – путь к базе данных.

Username – имя пользователя.

Password – пароль пользователя.

### Имя поставщика

Второй аргумент — это имя поставщика. Имя поставщика обычно совпадает с именем пакета NuGet поставщика. Например, для SQL Server или SQL Azure используйте Microsoft.EntityFrameworkCore.SqlServer. Для нашего случая поскольку Red Database основана на исходных кодах Firebird для нас подходит их поставщик услуг, а именно FirebirdSql.EntityFrameworkCore.Firebird.

### Добавление базы данных

Для добавления базы данных нам необходимо выполнить в терминале нашего приложения следующую команду:

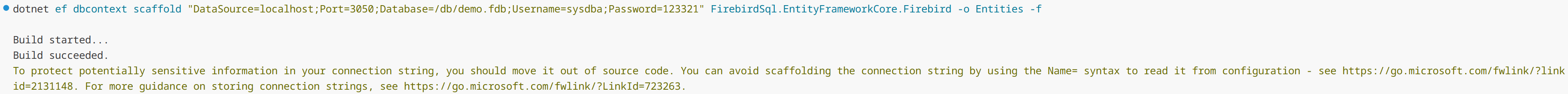
dotnet ef dbcontext scaffold "DataSource=localhost;Port=3050;Database=/db/demo.fdb;Username=sysdba;Password=7827940" FirebirdSql.EntityFrameworkCore.Firebird -o Entities -f

Здесь у нас появляется два новых параметра:

-o – используется для предоставление пользовательской папки для модели;

-f – используется для замены существующих файлов новой моделью (сгенерированной с использованием новой или обновлённой схемы).

После успешного выполнения команды в консоли вы увидите предупреждающее сообщение



И в проекте появится новая папка с файлами расширения \*.сs и названиями, которые мы указывали для таблиц базы данных.



# НАСТРОЙКА НАВИГАЦИИ

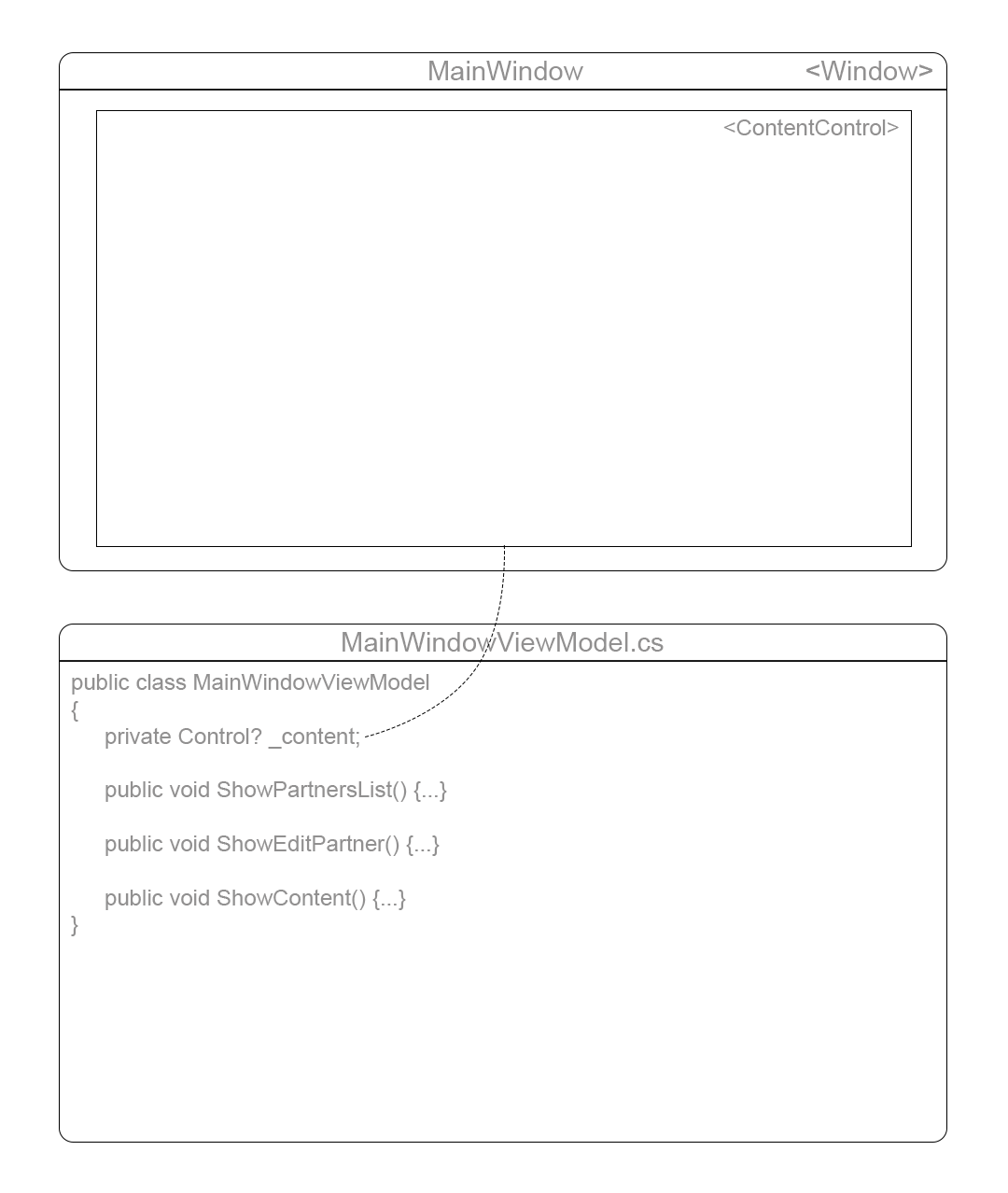
Все наше приложение будет расположено в окне класса Window.

Window является представителем высшего уровня ContentControl.

ContentControl — это контрол, который отображает данные, назначенные ему свойством Content.

Контролы — это базовые элементы для создания пользовательских интерфейсов.

Для навигации внутри приложения нам в главное окно MainWindow необходимо будет разместить ContentControl, в которой в будущем мы будем помещать пользовательские элементы управления (UserControl). Схематично это можно представить так:



## MainWindowViewModel.cs

Необходимо вместо строки:

public string Greeting { get; } = "Welcome to Avalonia!";

вставить следующий код:

[ObservableProperty]

private Control? \_content;

public void ShowContent(Control content)

{

Content = content;

}

public void ShowPartnersList()

{

//вставить код для отображения партнеров

}

Также вверху файла необходимо подключить пространства имен:

using Avalonia.Controls;

using CommunityToolkit.Mvvm.ComponentModel;

Выглядеть это должно так:



Тип ObservableProperty — это атрибут, позволяющий создавать наблюдаемые свойства из аннотированных полей. Его цель заключается в значительном уменьшении количества элементов, необходимых для определения наблюдаемых свойств.

## MainWindow.axaml

Необходимо вместо строки:

<TextBlock Text="{Binding Greeting}" HorizontalAlignment="Center" VerticalAlignment="Center"/>

вставить следующий код:

<ContentControl Content="{Binding Content}"/>

Выглядеть это должно так:

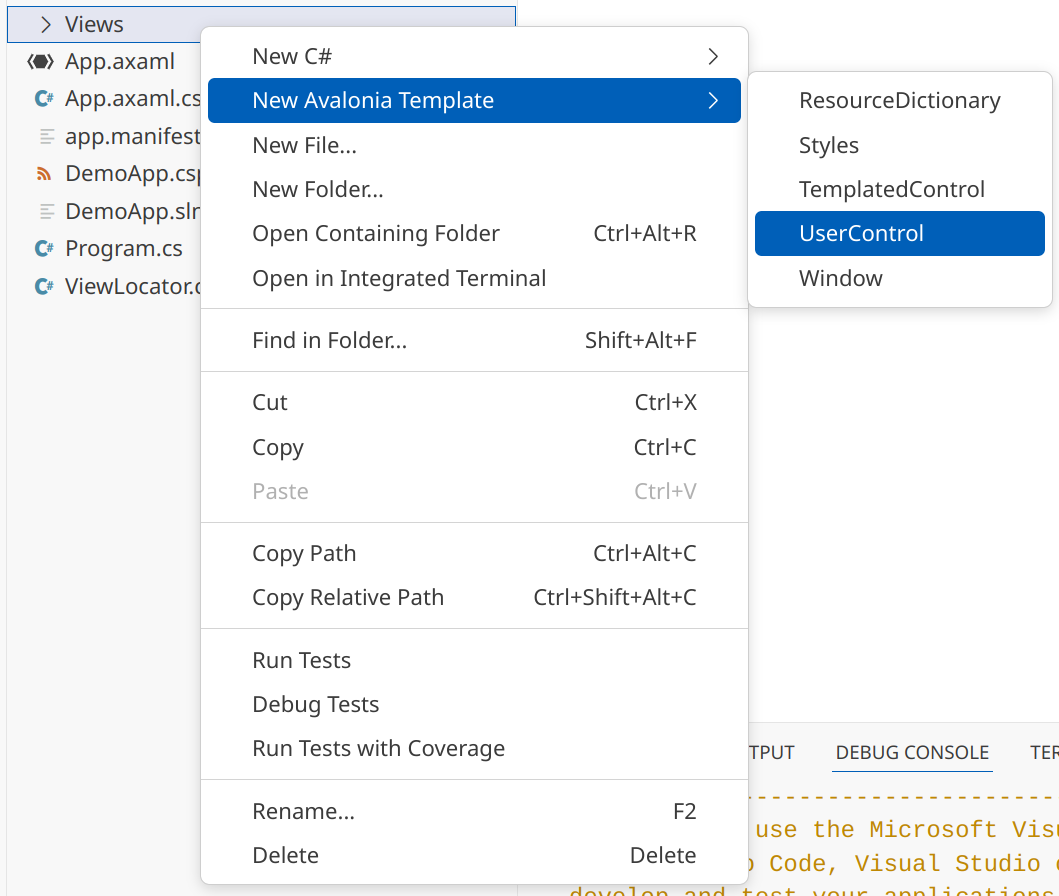


Попробуйте запустить приложение, оно должно запуститься!

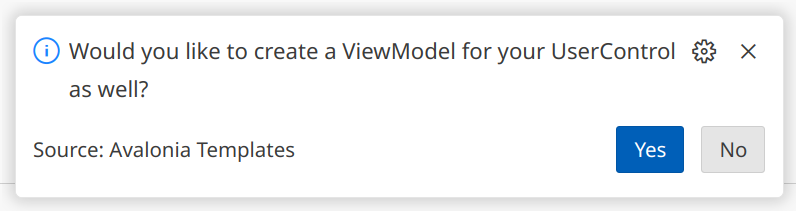
# СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ЭЛЕМЕНТА УПРАВЛЕНИЯ

UserControl — это ContentControl, представляющий собой многократно используемый набор элементов управления в заданном макете.

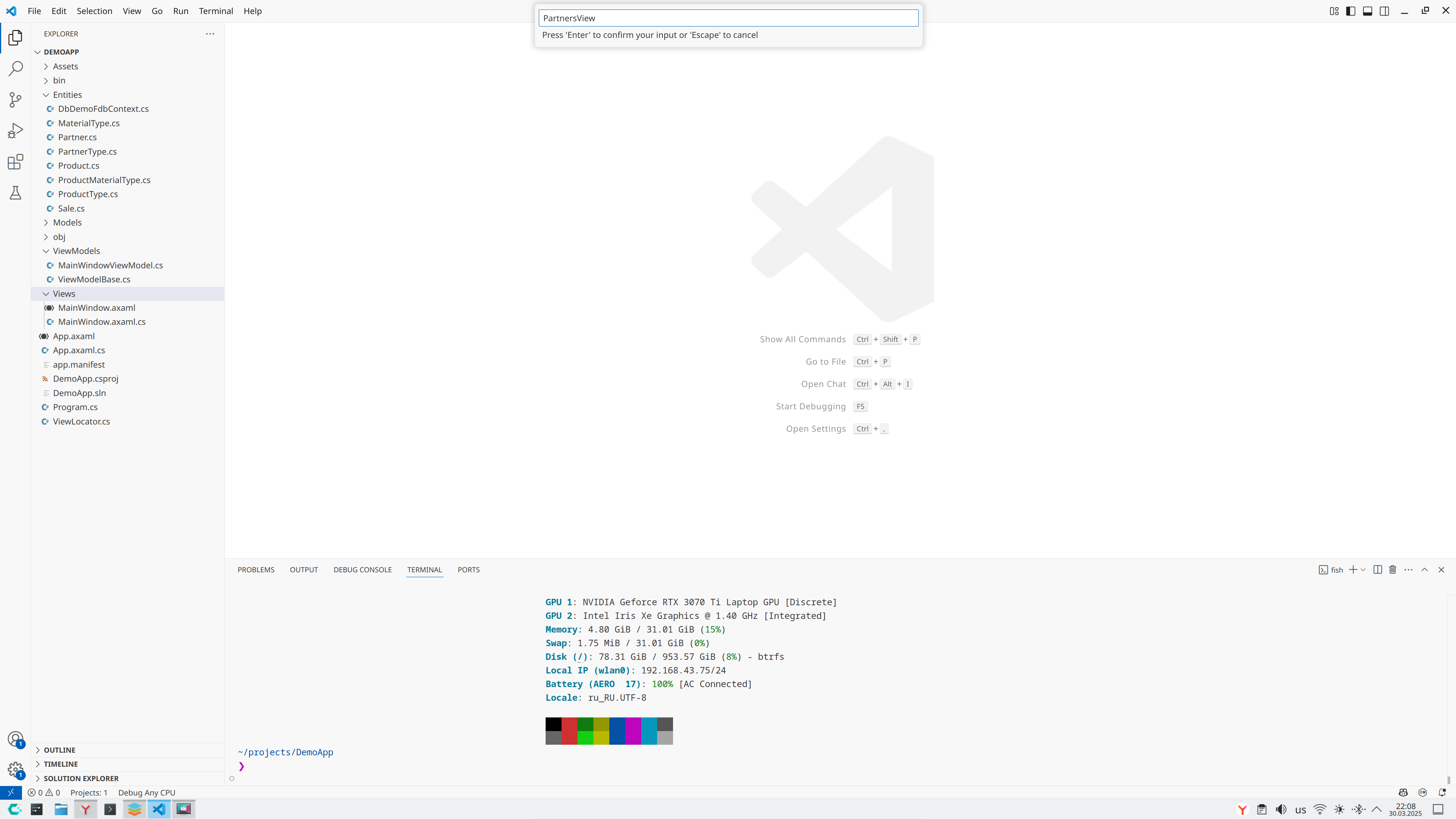
В каталоге Views необходимо добавить новый UserControl при помощи контекстного меню:



При выборе пункта меню UserControl, вам будет предложено создать ViewModel, вам необходимо отказаться:



В появившемся окне необходимо указать следующее название **PartnersView**:



Первое что нам необходимо это исправить x:Class с:

x:Class="/Views.PartnersView"

на:

x:Class="DemoApp.Views.PartnersView"

Также необходимо открыть файл PartnersView.axaml.cs и изменить его с:

namespace /Views;

на:

namespace DemoApp.Views;

Следующим шагом будет разметка нашего контрола для этого вместо:

Welcome to Avalonia!

необходимо вставить:

<Grid Margin="20">

<StackPanel Spacing="10">

<TextBlock Text="Список партнеров" FontSize="24" FontWeight="Bold" />

<StackPanel Orientation="Horizontal" Spacing="10">

<Button Content="Добавить партнера" Command="{Binding AddPartnerCommand}" />

<Button Content="Редактировать" Command="{Binding EditPartnerCommand}" />

</StackPanel>

<ListBox>

<!-- здесь будет разметка для списка партнеров -->

</ListBox>

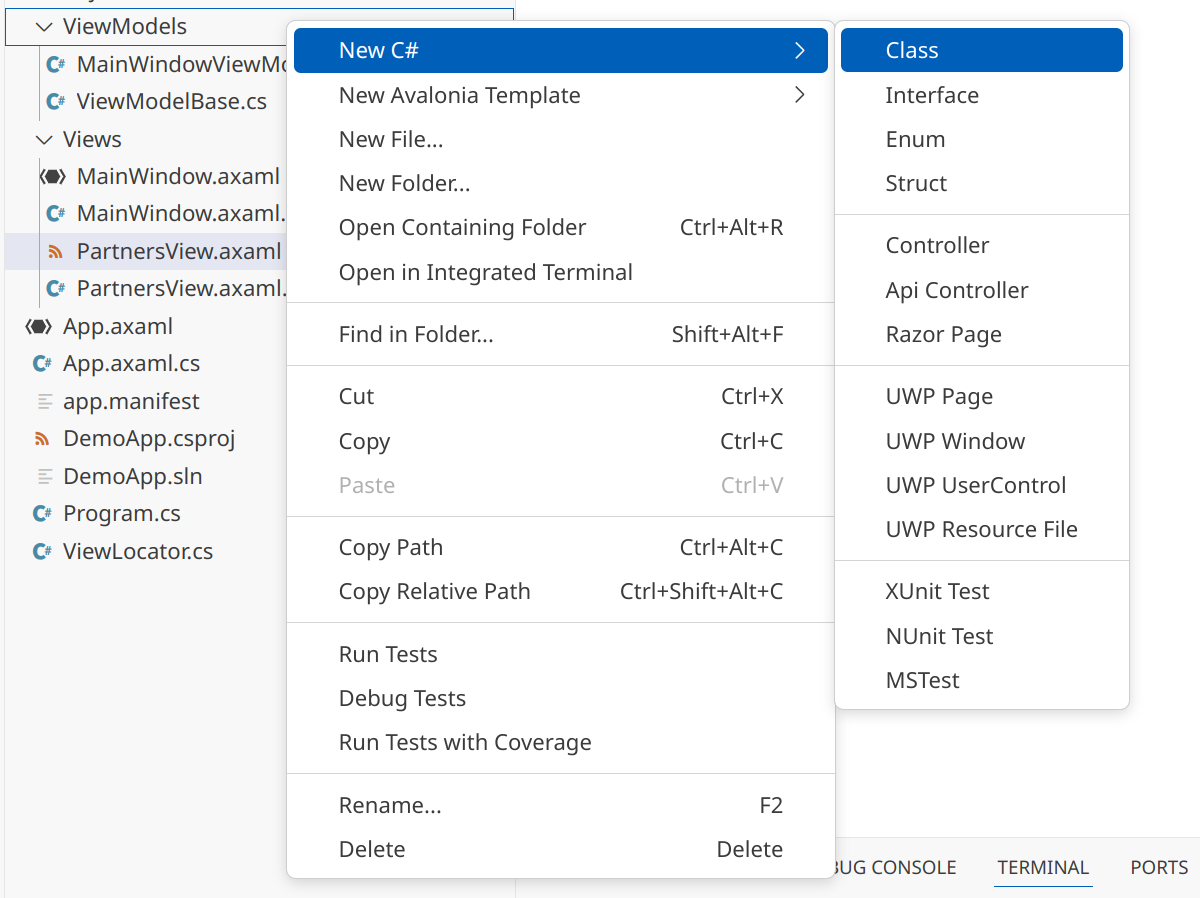
</StackPanel>

</Grid>

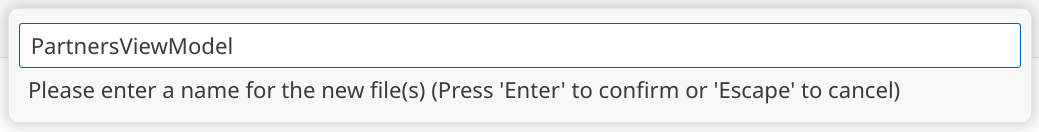
В результате наших действий разметка выглядит следующим образом:



Теперь нам необходимо создать ViewModel для нашего UserControl. В каталоге ViewModels необходимо добавить новый Class, при помощи контекстного меню:



В появившемся окне необходимо указать следующее название **PartnersViewModel**:



Изменим наш файл следующим образом

using CommunityToolkit.Mvvm.ComponentModel;

using CommunityToolkit.Mvvm.Input;

namespace DemoApp.ViewModels

{

public partial class PartnersViewModel : ObservableObject

{

public PartnersViewModel()

{

//отображение партнера

}

[RelayCommand]

private void EditPartner()

{

//изменение партнера

}

[RelayCommand]

private void AddPartner()

{

//добавление партнера

}

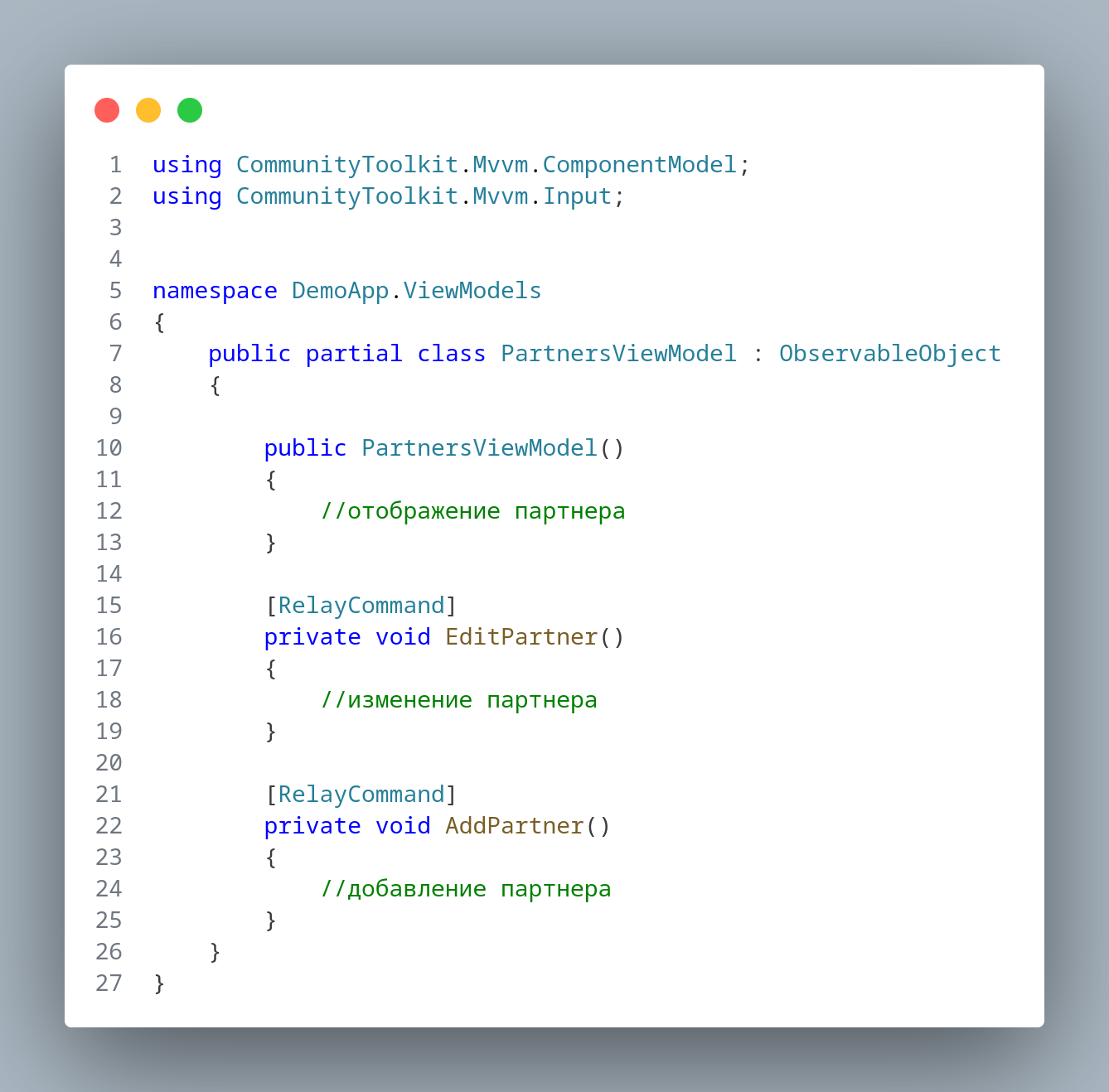
}

}

Было:



Стало:



Следующим шагом нам необходимо связать наш пользовательский элемент управления (View) и ViewModel. Для этого необходимо открыть файл PartnersView.axaml и изменить файл

<UserControl xmlns="https://github.com/avaloniaui"

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"

mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="450"

x:Class="DemoApp.Views.PartnersView"

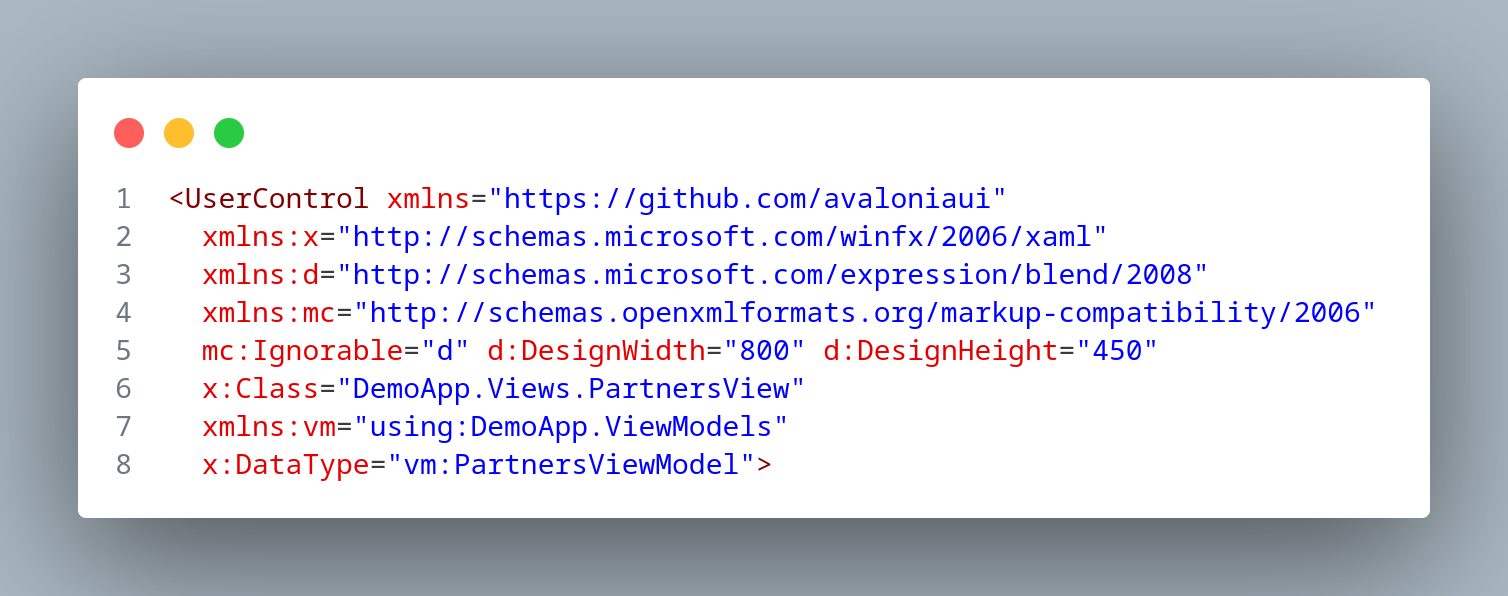
xmlns:vm="using:DemoApp.ViewModels"

x:DataType="vm:PartnersViewModel">

Было:



Стало:



Отобразим наш пользовательский элемент управления в главном окне, для этого необходимо изменить файл MainWindowViewModel.cs

Подключить пространство имен

using DemoApp.Views;

Изменить метод отображения партнеров и добавить конструктор

public void ShowPartnersList()

{

Content = new PartnersView

{

};

}

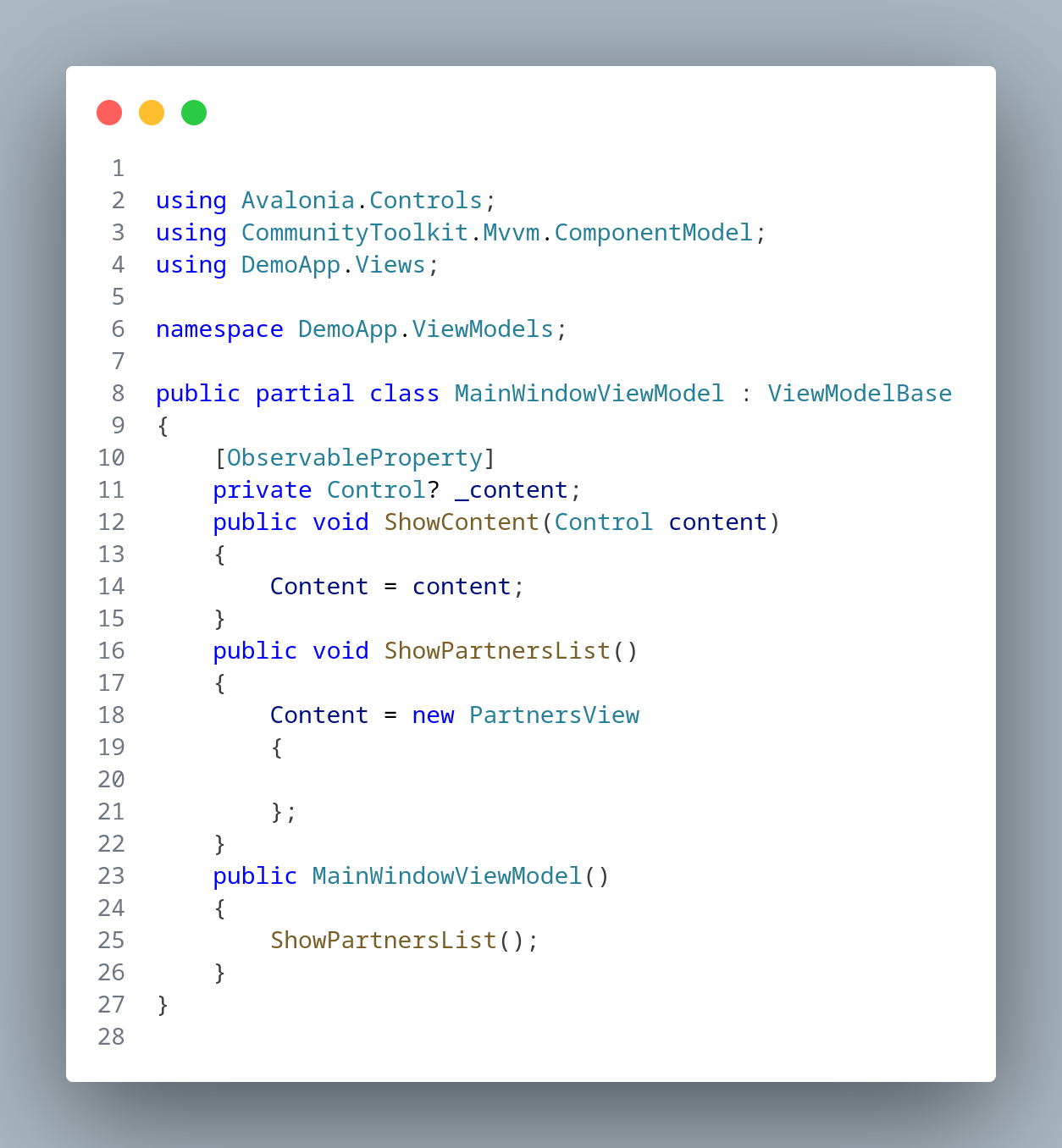
public MainWindowViewModel()

{

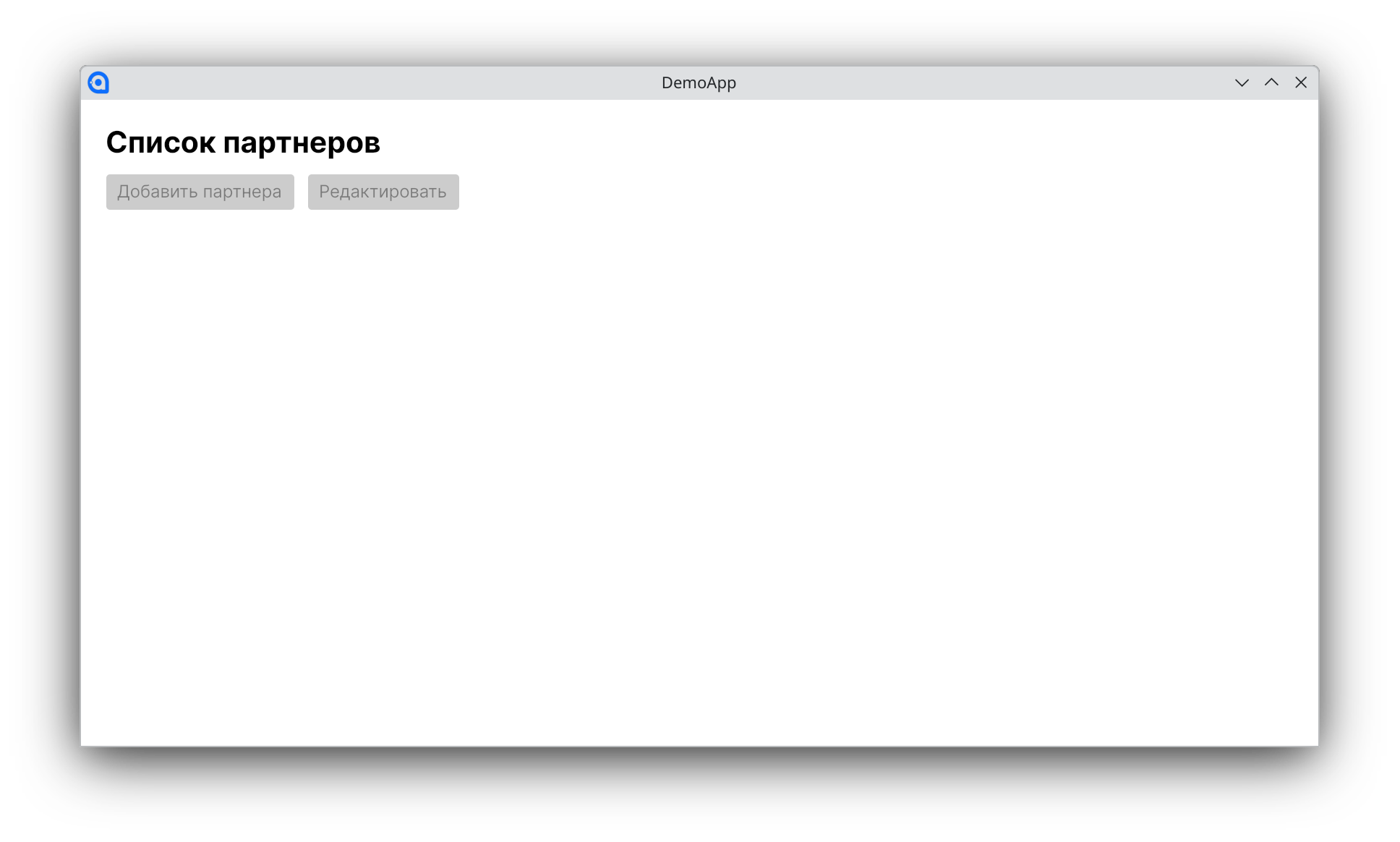
ShowPartnersList();

}

Наш файл будет выглядеть следующим образом:

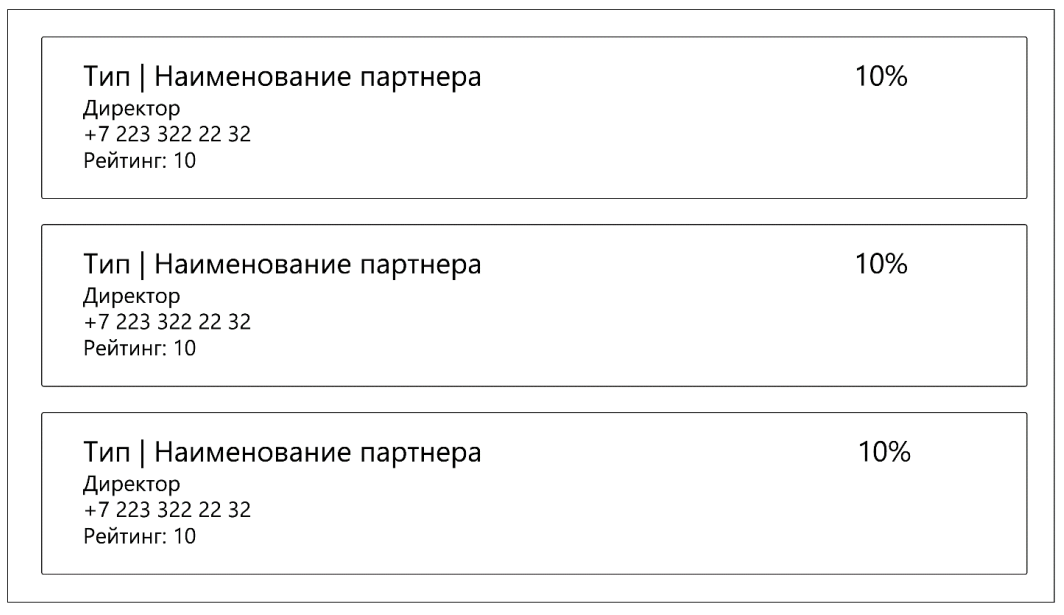


Если мы сейчас запустим наше приложение, то увидим



## НАСТРОЙКА LISTBOX

По заданию отображение партнеров должно быть организовано следующим образом:



Изменим наш ListBox следующим образом:

<ListBox ItemsSource="{Binding Partners}" SelectedItem="{Binding SelectedPartner}"

MaxHeight="800">

<ListBox.ItemTemplate>

<DataTemplate>

<StackPanel Orientation="Horizontal" Margin="5" HorizontalAlignment="Left">

<StackPanel Orientation="Vertical" Margin="5" HorizontalAlignment="Left">

<StackPanel Orientation="Horizontal" HorizontalAlignment="Left">

<TextBlock Text="Тип партнера" TextWrapping="Wrap" Width="150" />

<TextBlock Text=" | " TextWrapping="Wrap" />

<TextBlock Text="Наименование партнера" TextWrapping="Wrap" Width="250" />

</StackPanel>

<TextBlock Text="Директор" TextWrapping="Wrap" Width="150" HorizontalAlignment="Left" />

<TextBlock Text="Телефон партнера" TextWrapping="Wrap" Width="150" HorizontalAlignment="Left" />

<TextBlock Text="Рейтинг" TextWrapping="Wrap" Width="150" HorizontalAlignment="Left" />

</StackPanel>

<StackPanel Orientation="Vertical" Spacing="10">

<TextBlock Text="Скидка" TextWrapping="Wrap" Width="150" />

<Button Content="История продаж" />

</StackPanel>

</StackPanel>

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>

Наш ListBox будет выглядеть следующим образом:



Давайте добавим в наш ViewModel два свойства Partners и SelectedPartner.

Но прежде чем мы их туда добавим, нам необходимо создать модель партнера в каталоге Models. В каталоге Models необходимо добавить новый Class, при помощи контекстного меню с именем **Partner**

Далее необходимо изменить этот класс

using System.Linq;

namespace DemoApp.Models

{

public class Partner : Entities.Partner

{

public Entities.Partner \_partner { get; set; }

public Partner(Entities.Partner partner)

{

\_partner = partner;

}

public string? PartnerTypeDescription => \_partner.PartnerTypeNavigation?.PartnerTypeName;

public string ContactInfo => $"{\_partner.DirectorPhone} | {\_partner.DirectorMail}";

public string RatingDisplay => $"Рейтинг: {\_partner.PartnerRating ?? 0}";

public int TotalProductsSold => \_partner.Sales?.Sum(s => s.ProductCount) ?? 0;

public decimal Discount => CalculatePartnerDiscount(TotalProductsSold);

public string DiscountDisplay => $"Скидка: {Discount:P0}";

public static decimal CalculatePartnerDiscount(int totalProductsSold)

{

return totalProductsSold switch

{

< 10000 => 0,

< 50000 => 0.05m,

< 300000 => 0.10m,

\_ => 0.15m

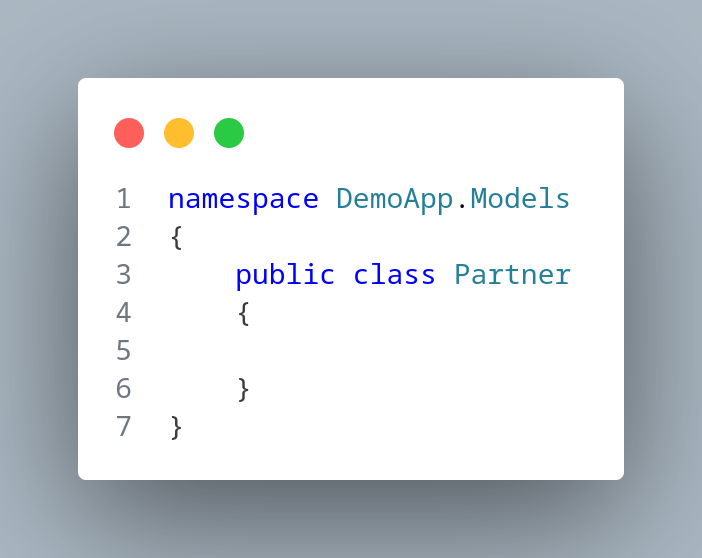
};

}

}

}

Было:



Стало:



Теперь отредактируем наш класс PartnersViewModel.cs и добавим туда наши два свойства

[ObservableProperty]

private ObservableCollection<Models.Partner> \_partners = new();

[ObservableProperty]

private Models.Partner? \_selectedPartner;

Изменим наш конструктор, чтобы он принимал в себя два: параметра контекст базы данных и ModelView главного окна, и вызовем в нем загрузку списка партнеров

public PartnersViewModel(DbDemoFdbContext context, MainWindowViewModel mainViewModel)

{

\_context = context;

\_mainViewModel = mainViewModel;

LoadPartners();

}

Также создадим под них свойства.

private readonly DbDemoFdbContext \_context;

internal readonly MainWindowViewModel \_mainViewModel;

Кроме свойств нам понадобится метод для загрузки списка партнеров

public void LoadPartners()

{

var entities = \_context.Partners.Include(p => p.Sales).Include(p => p.PartnerTypeNavigation).ToList();

Partners = new ObservableCollection<Models.Partner>(

entities.Select(e => new Models.Partner(e))

);

}

Также нам необходимо создать заглушку для просмотра истории продаж

[RelayCommand]

private void ViewSalesHistory(Partner? partner)

{

//история продаж

}

Необходимо отредактировать пространство имен про пространство имен, вверху файла должны быть подключены следующие пространства имен:

using System.Collections.ObjectModel;

using System.Linq;

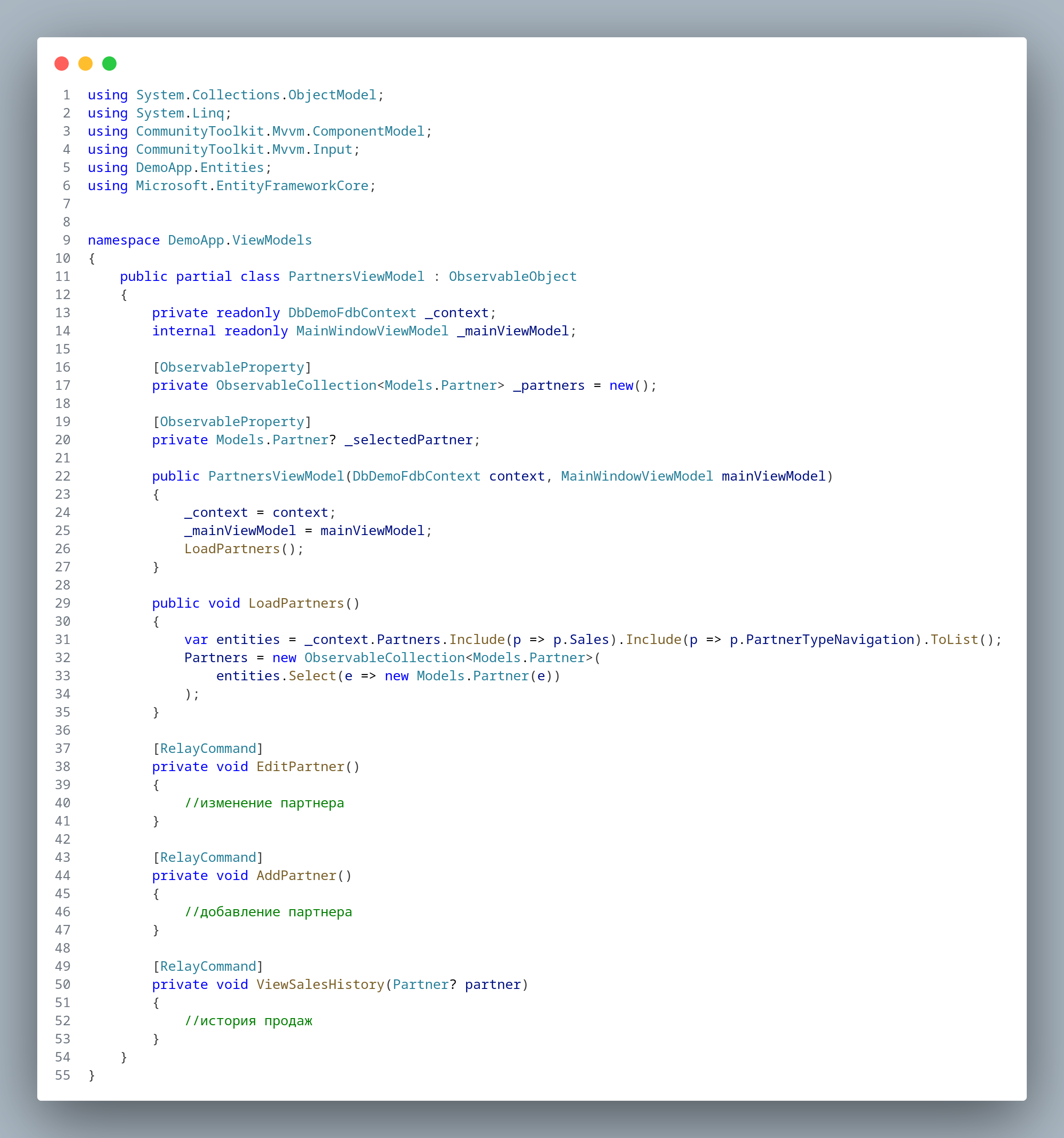
using CommunityToolkit.Mvvm.ComponentModel;

using CommunityToolkit.Mvvm.Input;

using DemoApp.Entities;

using Microsoft.EntityFrameworkCore;

Наш файл будет выглядеть следующим образом:



Полный код файла:

using System.Collections.ObjectModel;

using System.Linq;

using CommunityToolkit.Mvvm.ComponentModel;

using CommunityToolkit.Mvvm.Input;

using DemoApp.Entities;

using Microsoft.EntityFrameworkCore;

namespace DemoApp.ViewModels

{

public partial class PartnersViewModel : ObservableObject

{

private readonly DbDemoFdbContext \_context;

internal readonly MainWindowViewModel \_mainViewModel;

[ObservableProperty]

private ObservableCollection<Models.Partner> \_partners = new();

[ObservableProperty]

private Models.Partner? \_selectedPartner;

public PartnersViewModel(DbDemoFdbContext context, MainWindowViewModel mainViewModel)

{

\_context = context;

\_mainViewModel = mainViewModel;

LoadPartners();

}

public void LoadPartners()

{

var entities = \_context.Partners.Include(p => p.Sales).Include(p => p.PartnerTypeNavigation).ToList();

Partners = new ObservableCollection<Models.Partner>(

entities.Select(e => new Models.Partner(e))

);

}

[RelayCommand]

private void EditPartner()

{

//изменение партнера

}

[RelayCommand]

private void AddPartner()

{

//добавление партнера

}

[RelayCommand]

private void ViewSalesHistory(Partner? partner)

{

//история продаж

}

}

}

Следующим этапом станет изменение вызова данного ViewModel.

Открываем файл MainWindowViewModel.cs

Необходимо добавить свойство для работы с базой данных

private readonly DbDemoFdbContext \_context;

Далее редактируем конструктор

public MainWindowViewModel()

{

\_context = new DbDemoFdbContext();

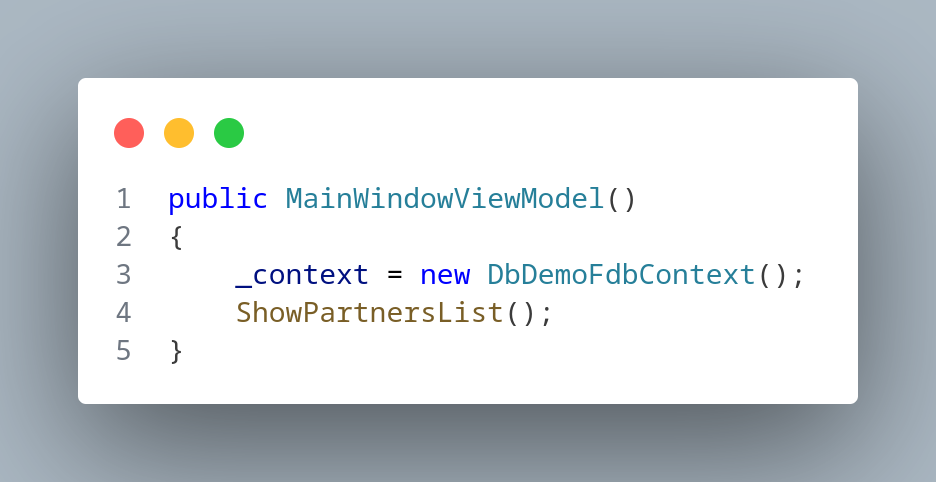
ShowPartnersList();

}

Было:



Стало:



Далее редактируем метод ShowPartnersList

public void ShowPartnersList()

{

Content = new PartnersView

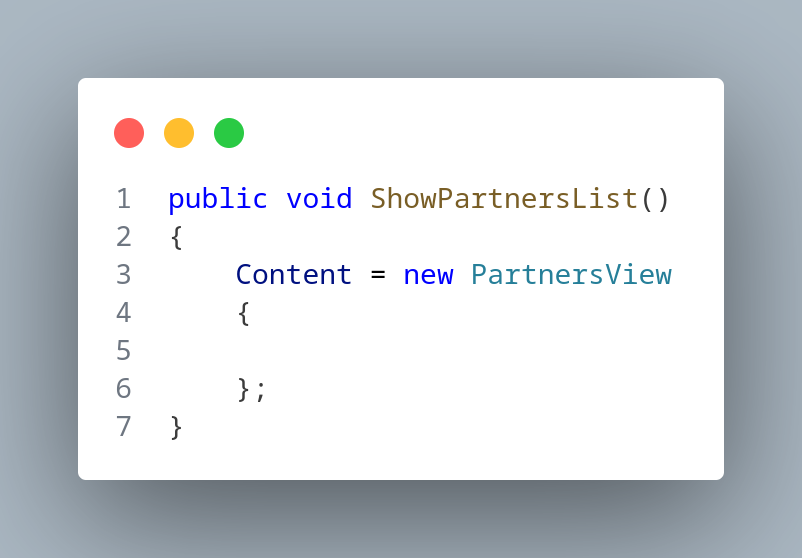
{

DataContext = new PartnersViewModel(\_context, this)

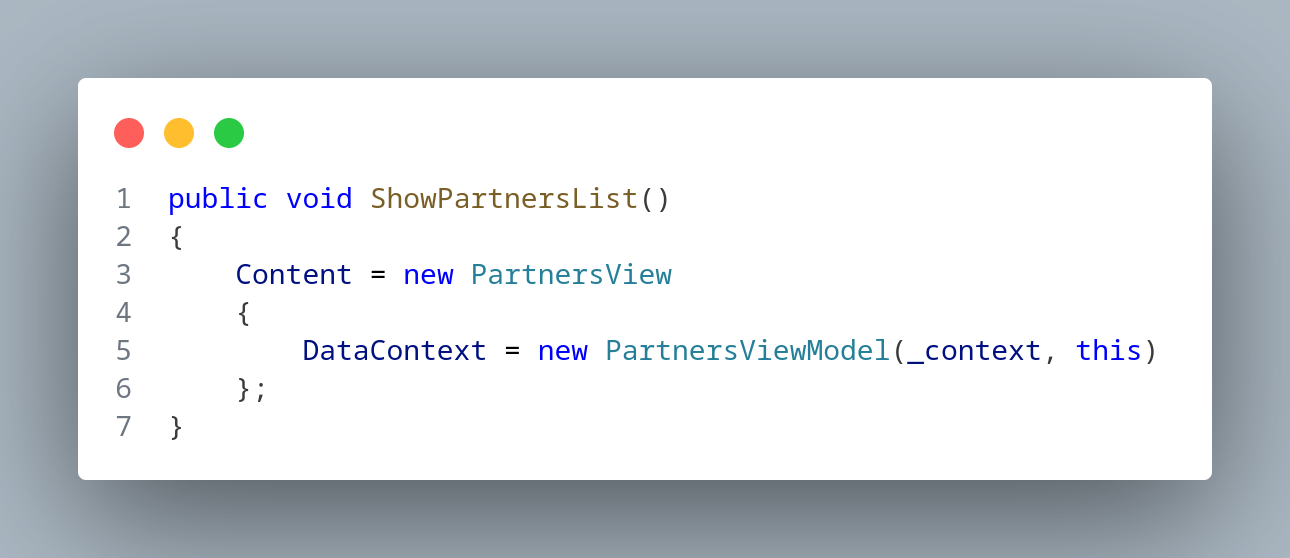
};

}

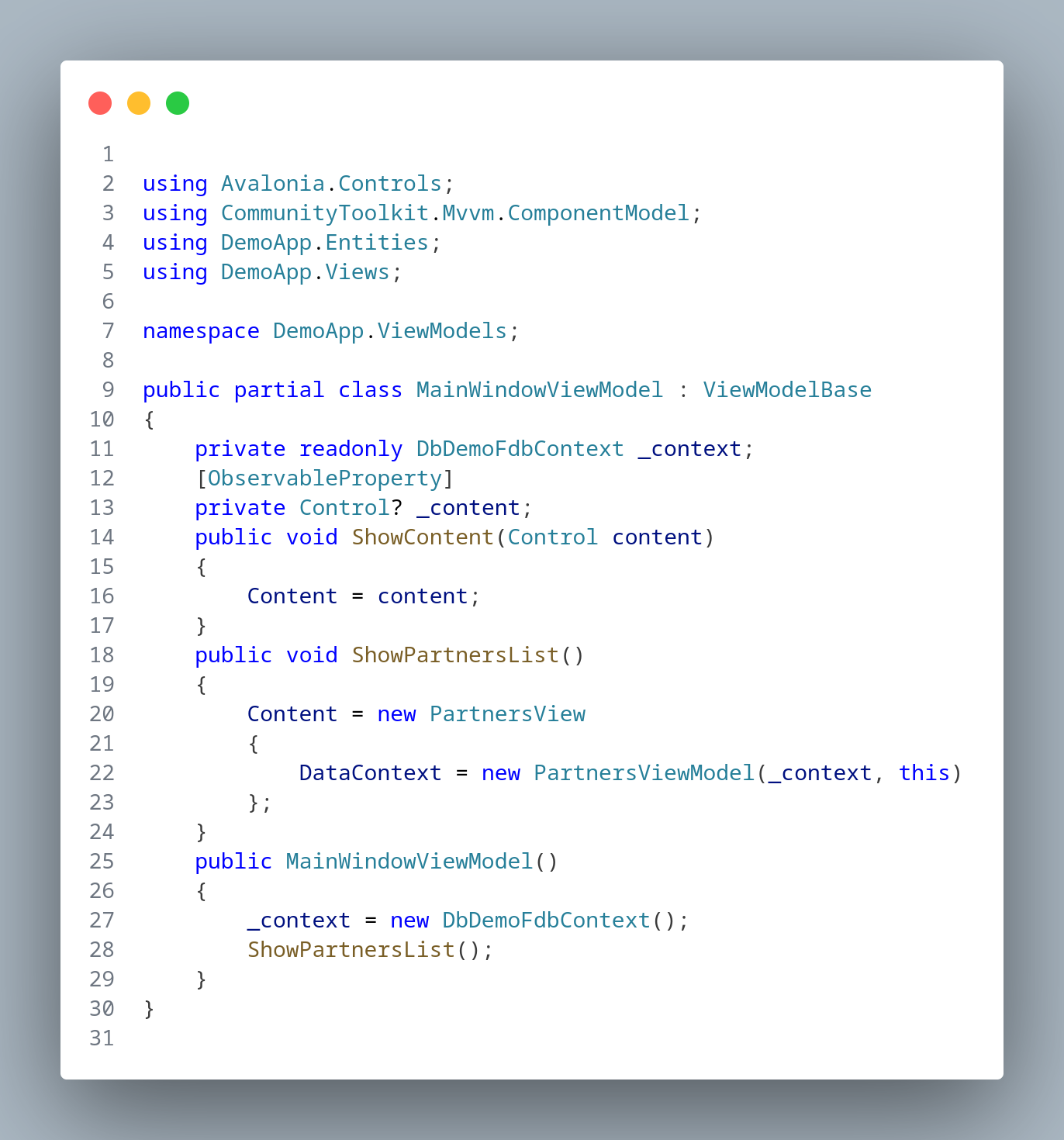
Было:



Стало:



Наш файл будет выглядеть следующим образом:



Полный код файла:

using Avalonia.Controls;

using CommunityToolkit.Mvvm.ComponentModel;

using DemoApp.Entities;

using DemoApp.Views;

namespace DemoApp.ViewModels;

public partial class MainWindowViewModel : ViewModelBase

{

private readonly DbDemoFdbContext \_context;

[ObservableProperty]

private Control? \_content;

public void ShowContent(Control content)

{

Content = content;

}

public void ShowPartnersList()

{

Content = new PartnersView

{

DataContext = new PartnersViewModel(\_context, this)

};

}

public MainWindowViewModel()

{

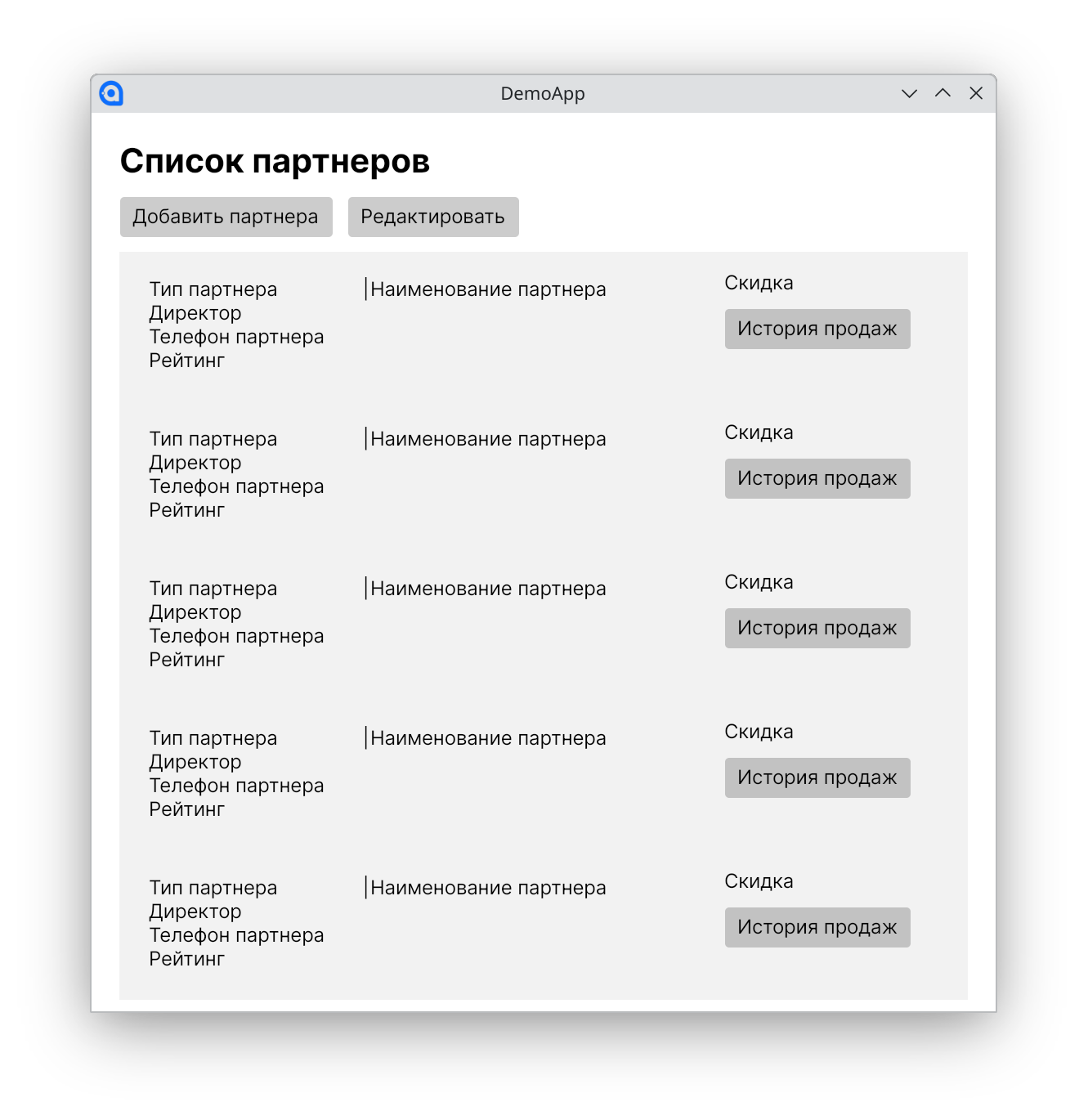
\_context = new DbDemoFdbContext();

ShowPartnersList();

}

}

Давайте запустим наше приложение и посмотрим как теперь оно выглядит



Мы видим текст, но не данные!!! Необходимо изменить разметку ListBox и забиндить туда данные!

**Binding** (привязка) в C# — инструмент программирования, который позволяет синхронизировать значения свойств двух разных объектов.

Привязка подразумевает взаимодействие двух объектов: источника и приёмника. Объект-приёмник создаёт привязку к определённому свойству объекта-источника. Если объект-источник модифицируется, объект-приёмник также будет модифицирован.

Изменим ListBox следующим образом:

<ListBox ItemsSource="{Binding Partners}" SelectedItem="{Binding SelectedPartner}"

MaxHeight="800">

<ListBox.ItemTemplate>

<DataTemplate>

<StackPanel Orientation="Horizontal" Margin="5" HorizontalAlignment="Left">

<StackPanel Orientation="Vertical" Margin="5" HorizontalAlignment="Left">

<StackPanel Orientation="Horizontal" HorizontalAlignment="Left">

<TextBlock Text="{Binding PartnerTypeDescription}" TextWrapping="Wrap" />

<TextBlock Text=" | " TextWrapping="Wrap" />

<TextBlock Text="{Binding \_partner.PartnerName}" TextWrapping="Wrap" Width="250" />

</StackPanel>

<TextBlock Text="{Binding \_partner.Director}" TextWrapping="Wrap" Width="250" HorizontalAlignment="Left" />

<TextBlock Text="{Binding \_partner.DirectorPhone}" TextWrapping="Wrap" Width="250" HorizontalAlignment="Left" />

<TextBlock Text="{Binding RatingDisplay}" TextWrapping="Wrap" Width="250" HorizontalAlignment="Left" />

</StackPanel>

<StackPanel Orientation="Vertical" Spacing="10">

<TextBlock Text="{Binding DiscountDisplay}" TextWrapping="Wrap" Width="150" />

<Button Content="История продаж" />

</StackPanel>

</StackPanel>

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>

Наш ListBox будет выглядеть следующим образом:



Наш файл будет выглядеть следующим образом:



Полный код файла:

<UserControl xmlns="https://github.com/avaloniaui"

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"

mc:Ignorable="d" d:DesignWidth="800" d:DesignHeight="450"

x:Class="DemoApp.Views.PartnersView"

xmlns:vm="using:DemoApp.ViewModels"

x:DataType="vm:PartnersViewModel">

<Grid Margin="20">

<StackPanel Spacing="10">

<TextBlock Text="Список партнеров" FontSize="24" FontWeight="Bold" />

<StackPanel Orientation="Horizontal" Spacing="10">

<Button Content="Добавить партнера" Command="{Binding AddPartnerCommand}" />

<Button Content="Редактировать" Command="{Binding EditPartnerCommand}" />

</StackPanel>

<ListBox ItemsSource="{Binding Partners}" SelectedItem="{Binding SelectedPartner}"

MaxHeight="800">

<ListBox.ItemTemplate>

<DataTemplate>

<StackPanel Orientation="Horizontal" Margin="5" HorizontalAlignment="Left">

<StackPanel Orientation="Vertical" Margin="5" HorizontalAlignment="Left">

<StackPanel Orientation="Horizontal" HorizontalAlignment="Left">

<TextBlock Text="{Binding PartnerTypeDescription}" TextWrapping="Wrap" />

<TextBlock Text=" | " TextWrapping="Wrap" />

<TextBlock Text="{Binding \_partner.PartnerName}" TextWrapping="Wrap" Width="250" />

</StackPanel>

<TextBlock Text="{Binding \_partner.Director}" TextWrapping="Wrap" Width="250" HorizontalAlignment="Left" />

<TextBlock Text="{Binding \_partner.DirectorPhone}" TextWrapping="Wrap" Width="250" HorizontalAlignment="Left" />

<TextBlock Text="{Binding RatingDisplay}" TextWrapping="Wrap" Width="250" HorizontalAlignment="Left" />

</StackPanel>

<StackPanel Orientation="Vertical" Spacing="10">

<TextBlock Text="{Binding DiscountDisplay}" TextWrapping="Wrap" Width="150" />

<Button Content="История продаж" />

</StackPanel>

</StackPanel>

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>

</StackPanel>

</Grid>

</UserControl>

Давайте запустим наше приложение и посмотрим, как теперь оно выглядит:

