Пример MVVM

Последнее обновление: 28.08.2016

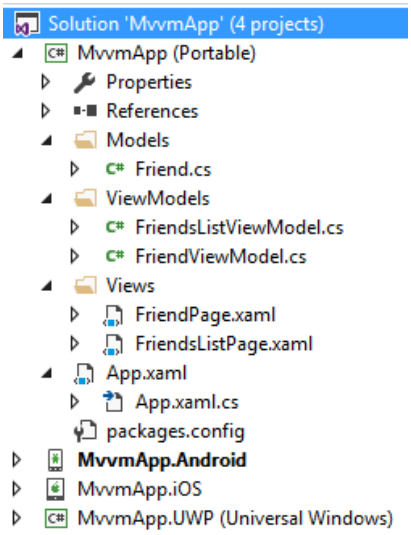
Рассмотрим на конкретном примере применение паттерна MVVM. Итак, создадим новое приложение, которое назовем **MvvmApp**. Задача нашего приложения будет заключаться в создании списка друзей, а также их добавлении, редактировании и удалении. Вобщем управление списком друзей.

Вначале добавим в главный проект три новых папки для каждого из компонентов паттерна:

- Папка Models
- Папка ViewModels
- Папка Views

Страницу MainPage.xaml, если она имется в проекте по умолчанию, можно удалить.

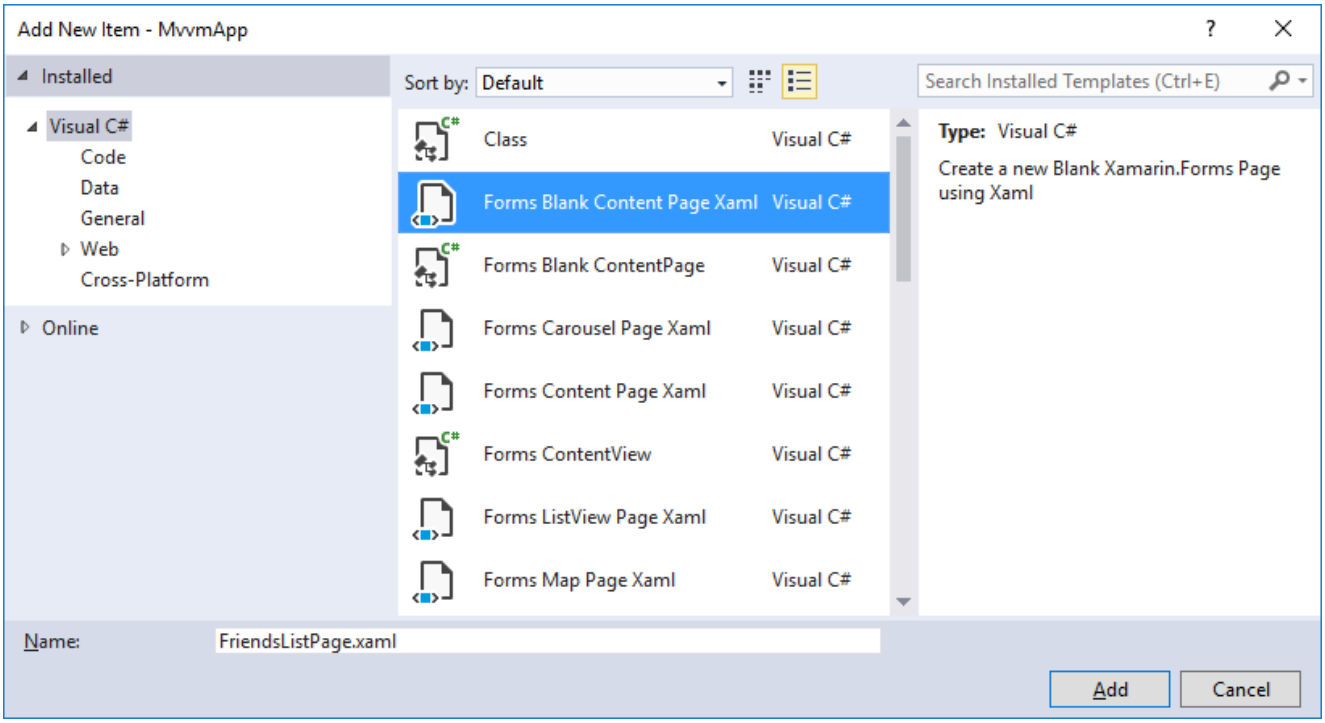
Финальная структура проекта будет выглядеть следующим образом:



Вначале определим в папке Models модель, которая будет представлять данные - класс Friend:

```
namespace MvvmApp.Models
{
    public class Friend
    {
        public string Name { get; set; }
        public string Email { get; set; }
        public string Phone { get; set; }
    }
}
```

Чтобы работать с данными этой модели добавим в папку Views две страницы XAML по типу **Content Page**: **FriendsListPage** (для вывода списка друзей) и **FriendPage** (для управления одним другом).



Далее добавим в другую папку *ViewModels* класс **FriendViewModel**:

```
using System.ComponentModel;
using MvvmApp.Models;

namespace MvvmApp.ViewModels
{
    public class FriendViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        FriendsListViewModel lvm;

        public Friend Friend { get; private set; }

        public FriendViewModel()
        {
            Friend = new Friend();
        }

        public FriendsListViewModel ListViewModel
        {
            get { return lvm; }
            set
            {
                if (lvm != value)
                {
                    lvm = value;
                    OnPropertyChanged("ListViewModel");
                }
            }
        }
        public string Name
        {
            get { return Friend.Name; }
            set
            {
                if (Friend.Name != value)
                {
                    Friend.Name = value;
                    OnPropertyChanged("Name");
                }
            }
        }
        public string Email
        {
            get { return Friend.Email; }
            set
            {
                if(Friend.Email != value)
                {
                    Friend.Email = value;
```

```
                    OnPropertyChanged("Email");
                }
            }
        }
        public string Phone
        {
            get { return Friend.Phone; }
            set
            {
                if (Friend.Phone != value)
                {
                    Friend.Phone = value;
                    OnPropertyChanged("Phone");
                }
            }
        }

        public bool IsValid
        {
            get
            {
                return ((!string.IsNullOrEmpty(Name.Trim())) ||
                    (!string.IsNullOrEmpty(Phone.Trim())) ||
                    (!string.IsNullOrEmpty(Email.Trim())));
            }
        }
        protected void OnPropertyChanged(string propName)
        {
            if (PropertyChanged != null)
                PropertyChanged(this, new PropertyChangedEventArgs(propName));
        }
    }
}
```

Фактически этот класс является надстройкой над объектом Friend.

Затем добавим в папку ViewModels новый класс, который будет представлять список друзей и который будет использоваться на странице FriendsListPage - класс **FriendsListViewModel**:

```
using System.Collections.ObjectModel;
using System.Windows.Input;
using Xamarin.Forms;
using System.ComponentModel;
using MvvmApp.Views;

namespace MvvmApp.ViewModels
{
    public class FriendsListViewModel : INotifyPropertyChanged
    {
        public ObservableCollection<FriendViewModel> Friends { get; set; }

        public event PropertyChangedEventHandler PropertyChanged;

        public ICommand CreateFriendCommand { protected set; get; }
        public ICommand DeleteFriendCommand { protected set; get; }
        public ICommand SaveFriendCommand { protected set; get; }
        public ICommand BackCommand { protected set; get; }
        FriendViewModel selectedFriend;

        public INavigation Navigation { get; set;}

        public FriendsListViewModel()
        {
            Friends = new ObservableCollection<FriendViewModel>();
            CreateFriendCommand = new Command(CreateFriend);
            DeleteFriendCommand = new Command(DeleteFriend);
            SaveFriendCommand = new Command(SaveFriend);
            BackCommand = new Command(Back);
        }

        public FriendViewModel SelectedFriend
        {
            get { return selectedFriend; }
            set
            {
                if (selectedFriend != value)
                {
                    FriendViewModel tempFriend = value;
                    selectedFriend = null;
                    OnPropertyChanged("SelectedFriend");
```

```
                    Navigation.PushAsync(new FriendPage(tempFriend));
                }
            }
        }
        protected void OnPropertyChanged(string propName)
        {
            if (PropertyChanged != null)
                PropertyChanged(this, new PropertyChangedEventArgs(propName));
        }

        private void CreateFriend()
        {
            Navigation.PushAsync(new FriendPage(new FriendViewModel() { ListViewModel = this }));
        }
        private void Back()
        {
            Navigation.PopAsync();
        }
        private void SaveFriend(object friendObject)
        {
            FriendViewModel friend = friendObject as FriendViewModel;
            if (friend != null && friend.IsValid)
            {
                Friends.Add(friend);
            }
            Back();
        }
        private void DeleteFriend(object friendObject)
        {
            FriendViewModel friend = friendObject as FriendViewModel;
            if (friend != null)
            {
                Friends.Remove(friend);
            }
            Back();
        }
    }
}
```

Для хранения списка друзей, который будет выводится на страницу, здесь определена коллекция `Friends`.

Навигация также является частью логики приложения, которая не относится к визуальной части, поэтому для хранения сервиса навигации здесь определено свойство `Navigation`. В дальнейшем через это свойство будет производиться переход к FriendPage.

Для управлением списком друзей в классе определено четыре команды. Команда добавления нового друга приводит в действие метод `CreateFriend()`, в котором производится переход к FriendPage. В конструктор FriendPage передается текущий объект FriendPageViewModel, который мы далее создадим.

По команде возвращения назад выполняется метод `Back()`, который производит переход назад.

Команда сохранения объекта выполняет метод `SaveFriend()`. В этом методе новый объект добавляется в коллекцию Friends.

По команде удаления вызывается метод `DeleteFriend`, который удаляет объект из списка.

Теперь изменим код страниц из папки Views. В коде C# страницы FriendsListPage пропишем следующее содержимое:

```
using Xamarin.Forms;
using MvvmApp.ViewModels;

namespace MvvmApp.Views
{
    public partial class FriendsListPage : ContentPage
    {
        public FriendsListPage()
        {
            InitializeComponent();
            BindingContext = new FriendsListViewModel() { Navigation = this.Navigation };
        }
    }
}
```

Здесь создается объект FriendsListViewModel, который устанавливается в качестве контекста страницы.

В коде XAML у этой страницы пропишем выражения привязки к этому объекту:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```

```
             x:Class="MvvmApp.Views.FriendsListPage" Title="Список друзей">
  <StackLayout>
    <Button Text="Добавить" Command="{Binding CreateFriendCommand}" />
    <ListView x:Name="booksList" ItemsSource="{Binding Friends}"
              SelectedItem="{Binding SelectedFriend, Mode=TwoWay}" HasUnevenRows="True">
      <ListView.ItemTemplate>
        <DataTemplate>
          <ViewCell>
            <ViewCell.View>
              <StackLayout>
                <Label Text="{Binding Name}" FontSize="Medium" />
                <Label Text="{Binding Email}" FontSize="Small" />
                <Label Text="{Binding Phone}" FontSize="Small" />
              </StackLayout>
            </ViewCell.View>
          </ViewCell>
        </DataTemplate>
      </ListView.ItemTemplate>
    </ListView>
  </StackLayout>
</ContentPage>
```

Несмотря на то, что тут определена кнопка для добавления нового объекта, код страницы не содержит никаких обработчиков, потому что всю обработку будет выполнять команда CreateFriendCommand, определенная в FriendsListViewModel.

Затем изменим код c# у страницы FriendPage:

```
using MvvmApp.ViewModels;
using Xamarin.Forms;

namespace MvvmApp.Views
{
    public partial class FriendPage : ContentPage
    {
        public FriendViewModel ViewModel { get; private set; }
        public FriendPage(FriendViewModel vm)
        {
            InitializeComponent();
            ViewModel = vm;
            this.BindingContext = ViewModel;
        }
    }
}
```

Теперь страница в качестве параметра в конструкторе принимает объект FriendViewModel, устанавливает его в качестве контекста и также не содержит никакой другой логики.

И в коде xaml у страницы определим выражения привязки:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MvvmApp.Views.FriendPage" Title="Информация о друге">
  <StackLayout>
    <StackLayout x:Name="friendStack">
      <Label Text="Имя" />
      <Entry Text="{Binding Name}" FontSize="Medium" />
      <Label Text="Электронная почта" />
      <Entry Text="{Binding Email}" FontSize="Medium" />
      <Label Text="Телефон" />
      <Entry Text="{Binding Phone}" FontSize="Medium" />
    </StackLayout>
    <StackLayout Orientation="Horizontal" HorizontalOptions="CenterAndExpand">
      <Button Text="Добавить" Command="{Binding ListViewModel.SaveFriendCommand}" CommandParameter="{Binding}" />
      <Button Text="Удалить" Command="{Binding ListViewModel.DeleteFriendCommand}" CommandParameter="{Binding}" />
      <Button Text="Назад" Command="{Binding Path=ListViewModel.BackCommand}" />
    </StackLayout>
  </StackLayout>
</ContentPage>
```

И при нажатии на кнопки будет вызываться соответствующая команда. Кроме того, первые две кнопки передают в команды параметр, который представляет привязанный объект - то есть тот объект FriendViewModel, который передан в конструктор и установлен в качестве контекста страницы. Выражение `{Binding}` без указания свойства объекта выполняет привязку к объекту, который является контекстом для данного элемента управления. Таким образом, команды в FriendsListViewModel получат добавляемый или удаляемый объект FriendViewModel.

При этому для редактирования в данном случае не надо нажимать никакую кнопку, так как при изменении значений в текстовых полях сразу же изменяться значения у этого объекта в списке. И после изменения значений достаточно нажать на

кнопку Назад для возврата на главную страницу. Правда, такое поведение не всегда бывает удобно. В этом случае мы можем добавить к объекту Friend/FriendViewModel какой-нибудь идентификатор. А при редактировании передавать не объект из списка, а его копию. Затем при сохранении данных в зависимости от Id выполнять либо добавление (если Id не установлен), либо редактирование. Удаление в этом случае также бы производилось бы по Id.

И в конце в классе App установим страницу FriendsListPage в качестве главной:

```
using Xamarin.Forms;
using MvvmApp.Views;

namespace MvvmApp
{
    public partial class App : Application
    {
        public App()
        {
            MainPage = new NavigationPage(new FriendsListPage());
        }

        protected override void OnStart()
        { }

        protected override void OnSleep()
        { }

        protected override void OnResume()
        { }
    }
}
```

Запустим приложение и добавим какой-нибудь объект.

И после добавления мы увидим этот объект в списке на главной странице:

Назад **Содержание** **Вперед**

марк • 4 месяца назад

встал вопрос, а как отредактировать друга? напишите пожалуйста

Ответить • Поделиться ›

марк ➚ марк • 4 месяца назад

если вдруг у кого тоже такая проблема, вот как я ее решил, но думаю можно и лучше, я там добавил по одному конструктору в класс который принимает имена чтобы их сразу записывать в лабелы и эмпти, и да, сразу все не получится отредактировать тк нужен 1 параметр по поиску старой ячейки

```
private void EditItem(object ItemObject)
{
ItemViewModel item = ItemObject as ItemViewModel;
Navigation.PushAsync(new EditItemPage(new ItemViewModel(item.Name, item.Count) { ListViewModel = this }));
}

private void AcceptEditItem(object ItemObject)
{
ItemViewModel item = ItemObject as ItemViewModel;
if (item.IsValid)
{
var temp = Items.FirstOrDefault(it => it.Name == item.Name);
Items[Items.IndexOf(temp)].Count = item.Count;
}
Back();
}
```

Ответить • Поделиться ›

марк • 4 месяца назад

у всех не работает кнопка удаления или только у меня?

Ответить • Поделиться ›

Metanit Модератор ➚ марк • 4 месяца назад

Если делать все статье, то удаление должно работать. На всякий случай даже перепроверил код.

Ответить • Поделиться ›

Павел Махнёв ➚ Metanit • 3 месяца назад

У меня тоже не работало ни удаление не редактирование, когда я пробовал дебажить\запускать в Xamarin Live Player (уже доступен не в превью версии VS), после билда apk все заработало на том же устройстве. Возможно, кому-то будет полезна эта информация. Хоть пока он [Xamarin Live Player], очевидно, и не подходит для серьезной работы, но все равно порекомендую для тестирования интерфейса на лету.

P.S. Огромное спасибо за материалы на сайте, уже очень многое почерпнул для себя на нем.

Ответить • Поделиться ›

Metanit Модератор ➚ Павел Махнёв • 3 месяца назад

я все примеры тестирую на реальных устройствах, возможно, на эмуляторах что-то будет отличаться или работать не так, но в любом случае лучше отдавать предпочтение реальным устройствам, чем эмуляторам.

Ответить • Поделиться ›

марк ➚ Metanit • 4 месяца назад

спасибо за ответ)

Ответить • Поделиться ›

марк ➚ марк • 4 месяца назад

кароче просто привязал эту кнопку к контекстному меню, где не надо ничего вводить и тогда всё пашет, так что да, траблы что не эквиваленты

• Ответить • Поделиться ›

марк ➚ марк • 4 месяца назад

сделал так но кажется что это бред и можно лучше)

```
private void DeleteItem(object ItemObject)
{
ItemViewModel item = ItemObject as ItemViewModel;
if (item != null)
{
foreach (var it in Items)
{
if (item.Name != it.Name || item.Count != it.Count) continue;
Items.Remove(it);
break;
}
}
Back();
}
```

я друга заменил на предмет и его количество, все остальное такое же

• Ответить • Поделиться ›

марк ➚ марк • 4 месяца назад

я так понимаю что объект который передается не эквивалентен объекту коллекции?

• Ответить • Поделиться ›

Denis Lohachov • 6 месяцев назад

использую паттерн MVVM, есть такая ViewModel

```
#region Fields
private string _login;
private string _password;
#endregion

#region Propertys
public string Login
{
get { return _login; }
set
{
if(_login != value)
{
_login = value;
OnPropertyChanged("Login");
}
}
```

показать больше

• Ответить • Поделиться ›

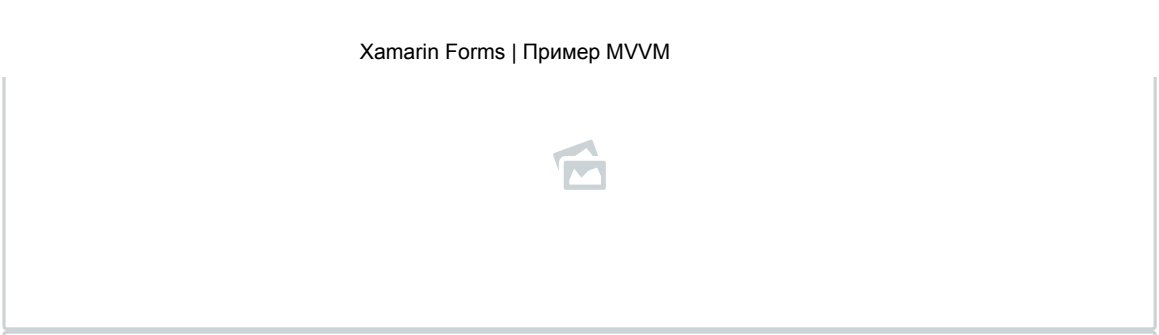
Metanit Модератор ➚ Denis Lohachov • 6 месяцев назад

тут не очень понятно, какой именно объект здесь равен null, вы смотрели через отладку?

• Ответить • Поделиться ›

Denis Lohachov ➚ Metanit • 6 месяцев назад

да , вот что показывает



• Ответить • Поделиться ›

Denis Lohachov ➚ Denis Lohachov • 6 месяцев назад

нашел решение,
заменил Navigation.PushAsync(new ChoiseRecipiantView(), true);
на Application.Current.MainPage.Navigation.PushAsync(new ChoiseRecipiantView(), true);
но в чем проблема так и не понял. С ViewModel нельзя обратится к самой странице?

• Ответить • Поделиться ›

Metanit Модератор ➚ Denis Lohachov • 6 месяцев назад

у вас видимо просто объект Navigation не установлен
обратите внимание как он устанавливается в коде странице
BindingContext = new FriendsListViewModel() { Navigation = this.Navigation };

• Ответить • Поделиться ›

Denis Lohachov ➚ Metanit • 6 месяцев назад

Да вы правы, это же очевидно. как такое сразу не заметил(((Спасибо!

• Ответить • Поделиться ›

Anton Maslii • год назад

Для того чтоб при изменении текста в Entry оно автоматически не сохранялось можно сделать все намного проще. Нужно в xaml указать способ привязки. Например <entry text="{Binding Name, Mode=OneWay}"/> указывает, что привязка выполняется только от источника к цели. По умолчанию в Entry установлен режим Mode=TwoWay, поэтому и работает двухсторонняя привязка.

• Ответить • Поделиться ›