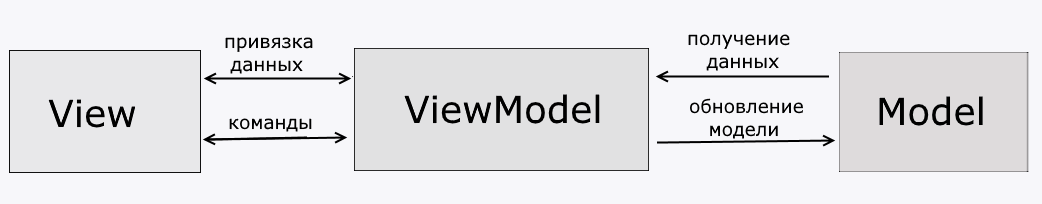
1.Шаблон проектирования — повторяемая архитектурная конструкция в сфере проектирования программного обеспечения, предлагающая решение проблемы проектирования.

* MVVM
* MVC

2.

3.View в MVVM отвечает за представление (окна и их элементы)

4.ViewModels в MVVM отвечает за взаимодействие модели и представления

5. Models в MVVM отвечает за модель (объекты языка или базы данных). SSMS

6."Server=(localdb)\mssqllocaldb;Database=helloappdb;Trusted\_Connection=True;"

7. Интерфейс INotifyPropertyChanged используется для уведомления клиентов (обычно привязывая клиентов) об изменении значения свойства.

8. По функциональности коллекция ObservableCollection похожа на список List за тем исключением, что позволяет известить внешние объекты о том, что коллекция была изменена.

9. Для установки источника или контекста данных в элементах управления WPF предусмотрено свойство DataContext.

<Window x:Class="MyApp.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="My Application">

<Window.DataContext>

<local:MainWindowViewModel />

</Window.DataContext>

*<!-- Остальной код окна -->*

</Window>

В данном примере мы устанавливаем контекст данных (DataContext) для окна Window на экземпляр класса MainWindowViewModel. Это позволяет использовать свойства и методы этого класса из XAML-разметки окна.

10. Команды представляют механизм выполнения какой-нибудь задачи.

|  |  |
| --- | --- |
| [CanExecute(Object)](https://learn.microsoft.com/ru-ru/dotnet/api/system.windows.input.icommand.canexecute?view=net-7.0#system-windows-input-icommand-canexecute(system-object)) | Определяет метод, который определяет, может ли данная команда выполняться в ее текущем состоянии. |
| [Execute(Object)](https://learn.microsoft.com/ru-ru/dotnet/api/system.windows.input.icommand.execute?view=net-7.0#system-windows-input-icommand-execute(system-object)) | Определяет метод, вызываемый при вызове данной команды. |

11. public class RelayCommand : ICommand

{

private Action<object> execute;

private Func<object, bool> canExecute;

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

public RelayCommand(Action<object> execute, Func<object, bool> canExecute = null)

{

this.execute = execute;

this.canExecute = canExecute;

}

public bool CanExecute(object parameter)

{

return this.canExecute == null || this.canExecute(parameter);

}

public void Execute(object parameter)

{

this.execute(parameter);

}

}

Пример использования RelayCommand на стороне модели представления:

public class MainViewModel : INotifyPropertyChanged

{

private string \_displayText;

public string DisplayText

{

get { return \_displayText; }

set

{

\_displayText = value;

OnPropertyChanged(nameof(DisplayText));

}

}

public ICommand SayHelloCommand { get; set; }

public MainViewModel()

{

SayHelloCommand = new RelayCommand(SayHello);

}

private void SayHello(object obj)

{

DisplayText = "Hello, world!";

}

}

Пример привязки команды к кнопке в WPF:

<Button Content="Say Hello" Command="{Binding SayHelloCommand}" />

12. Валидация в рамках паттерна MVVM может быть выполнена несколькими способами:

1. Использование интерфейса INotifyDataErrorInfo

INotifyDataErrorInfo - это интрефейс, который позволяет оповестить приложение о наличии ошибок ввода и их описаниях. Для того чтобы воспользоваться этим интерфейсом, нужно создать класс валидатора и реализовать его.

1. Использование библиотек валидации

Существуют множество библиотек валидации для разных платформ, таких как FluentValidation для .NET или Validator.js для JavaScript. Эти библиотеки предоставляют готовые решения для валидации данных в модели представления.

В WPF ValidationRule - это класс, который позволяет выполнять пользовательскую валидацию данных в элементах управления.

Пример создания класса ValidationRule для проверки количества товара:

csharpCopy Code

public class PositiveIntValidationRule : ValidationRule

{

public override ValidationResult Validate(object value, CultureInfo cultureInfo)

{

int intValue;

if (!int.TryParse(value.ToString(), out intValue) || intValue < 0)

{

return new ValidationResult(false, "Value must be a positive integer");

}

return ValidationResult.ValidResult;

}

}

Здесь мы создали класс PositiveIntValidationRule наследующий от базового класса ValidationRule. Метод Validate принимает два параметра: значение, которое нужно проверить и информацию о культуре.

В методе Validate мы преобразуем переданное значение к целочисленному типу и проверяем, является ли оно положительным числом. Если это не так, то возвращаем объект ValidationResult с флагом false и сообщением об ошибке.

Далее мы можем использовать этот класс как ValidationRule для любого элемента управления в WPF. Например, можно использовать его для TextBox следующим образом:

Copy Code

<TextBox>

<TextBox.Text>

<Binding Path="Quantity" UpdateSourceTrigger="PropertyChanged">

<Binding.ValidationRules>

<local:PositiveIntValidationRule />

</Binding.ValidationRules>

</Binding>

</TextBox.Text>

</TextBox>

Здесь мы связываем свойство Text элемента TextBox с свойством Quantity в модели представления. Также мы добавляем правило валидации PositiveIntValidationRule для проверки корректности ввода количества товара. Если значение не проходит валидацию, то будет показано сообщение об ошибке.

Для проверки сложности пароля можно использовать интерфейс IDataErrorInfo вместе с атрибутами валидации. Вот пример кода на языке C#, который показывает, как это можно сделать:

csharpCopy Code

public class User : IDataErrorInfo

{

public string Password { get; set; }

public string this[string columnName]

{

get

{

if (columnName == "Password")

{

if (string.IsNullOrEmpty(Password))

{

return "Поле Пароль обязательно для заполнения.";

}

else if (Password.Length < 8)

{

return "Пароль должен содержать не менее 8 символов.";

}

else if (!Regex.IsMatch(Password, @"\d+"))

{

return "Пароль должен содержать хотя бы одну цифру.";

}

else if (!Regex.IsMatch(Password, "[A-Z]"))

{

return "Пароль должен содержать хотя бы одну заглавную букву.";

}

else if (!Regex.IsMatch(Password, "[a-z]"))

{

return "Пароль должен содержать хотя бы одну строчную букву.";

}

}

return null;

}

}

public string Error

{

get { return null; }

}

}

Обратите внимание, что здесь используются регулярные выражения для проверки наличия цифры, заглавной и строчной буквы в пароле. Если данные критерии не выполняются, возвращается ошибка соответствующего типа. Если все критерии выполнены, то метод возвращает null, что означает, что пароль прошел проверку и может быть сохранен.

13.

Достоинства:

* Разделение логики приложения и пользовательского интерфейса, что делает код более чистым и поддерживаемым;
* Упрощение тестирования, поскольку модели представления можно протестировать без необходимости отображения пользовательского интерфейса;
* Уменьшение связанности между компонентами приложения, что улучшает его гибкость и возможность внесения изменений;
* Поддержка более легковесной архитектуры приложения за счет использования привязок данных.

Недостатки:

* Необходимость дополнительной работы по созданию классов моделей представления, что может привести к более сложной архитектуре приложения;
* Сложность использования в простых приложениях, что может быть избыточным и занимать больше времени на разработку;
* Нагрузка на систему, которая может отрицательно сказаться на производительности приложения.

Если проект требует управляемый пользовательский интерфейс, то MVVM может быть хорошим выбором. Особенно если архитектура приложения должна быть масштабируемой и поддерживаемой в долгосрочной перспективе.

С другой стороны, обработчики событий могут быть более простым решением для менее сложных проектов или когда нужен быстрый и простой способ разработки. Но при этом следует помнить, что такой подход может усложнить тестирование и поддержку кода в будущем. Поэтому, если проект предполагает большой объем кода и время жизни проекта длительное, более предпочтительным будет использование MVVM.