

C# Mobile

Week 7

- Behaviors
- MVVM
- Commands
- CommunityToolkit.Mvvm



**DE HOGESCHOOL
MET HET NETWERK**

Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be



Doelstellingen

- De junior-collega kan met behulp van Behaviors extra functionaliteit toevoegen aan controls
- De junior-collega kent de verschillende componenten uit het MVVM patroon
- De junior-collega kan de basis concepten van MVVM toepassen en herkennen in eigen projecten
- De junior-collega kan een Command aanmaken en gebruiken in een MAUI applicatie
- De junior-collega kan de Community MVVM Toolkit installeren en gebruiken



Behaviors

Behaviors

- **Met behaviors voegen we extra functionaliteit toe aan de Controls uit .NET MAUI**
- **Niet in de code-behind**
- **Lijkt een beetje op een converter:**
 - C# klasse
 - Afgeleid van de Behavior-klasse of Behavior<T>-klasse
 - Heeft twee methodes OnAttachedTo en OnDetachingFrom
- **Bekijk ook de Community Toolkit: verschillende behaviors beschikbaar**

Behaviors

- We kunnen gedrag van Controls veranderen
- Bijvoorbeeld:
 - Entry rode tekst-kleur geven bij een foutieve input
- Toevoegen in XAML:

```
<Entry Text="{Binding Game.Rating}"
      VerticalOptions="CenterAndExpand"
      HorizontalOptions="FillAndExpand">
    <Entry.Behaviors>
      <behaviors:RatingBehavior></behaviors:RatingBehavior>
    </Entry.Behaviors>
</Entry>
```

=>

```
0 references
class RatingBehavior: Behavior<Entry>
{
    0 references
    protected override void OnAttachedTo(Entry entry)
    {
        entry.TextChanged += OnEntryTextChanged;
        base.OnAttachedTo(entry);
    }

    0 references
    protected override void OnDetachingFrom(Entry entry)
    {
        entry.TextChanged -= OnEntryTextChanged;
        base.OnDetachingFrom(entry);
    }
}

2 references
void OnEntryTextChanged(object sender, TextChangedEventArgs args)
{
    int result;
    bool isValid = int.TryParse(args.NewTextValue, out result);
    if (isValid)
    {
        if (result < 0 || result > 100)
        {
            isValid = false;
        }
    }
    ((Entry)sender).TextColor = isValid ? Color.Default : Color.Red;
}
```

Behaviors: MAUI Community toolkit

- Toolkit gemaakt (voornamelijk) door de community
- Bevat kant-en-klare behaviors, converters en andere uitbreidingen op MAUI
- <https://learn.microsoft.com/en-us/dotnet/communitytoolkit/maui/>



MVVM

Architectuur – MVVM

Model-View-ViewModel ≈ MVC (C# Web)

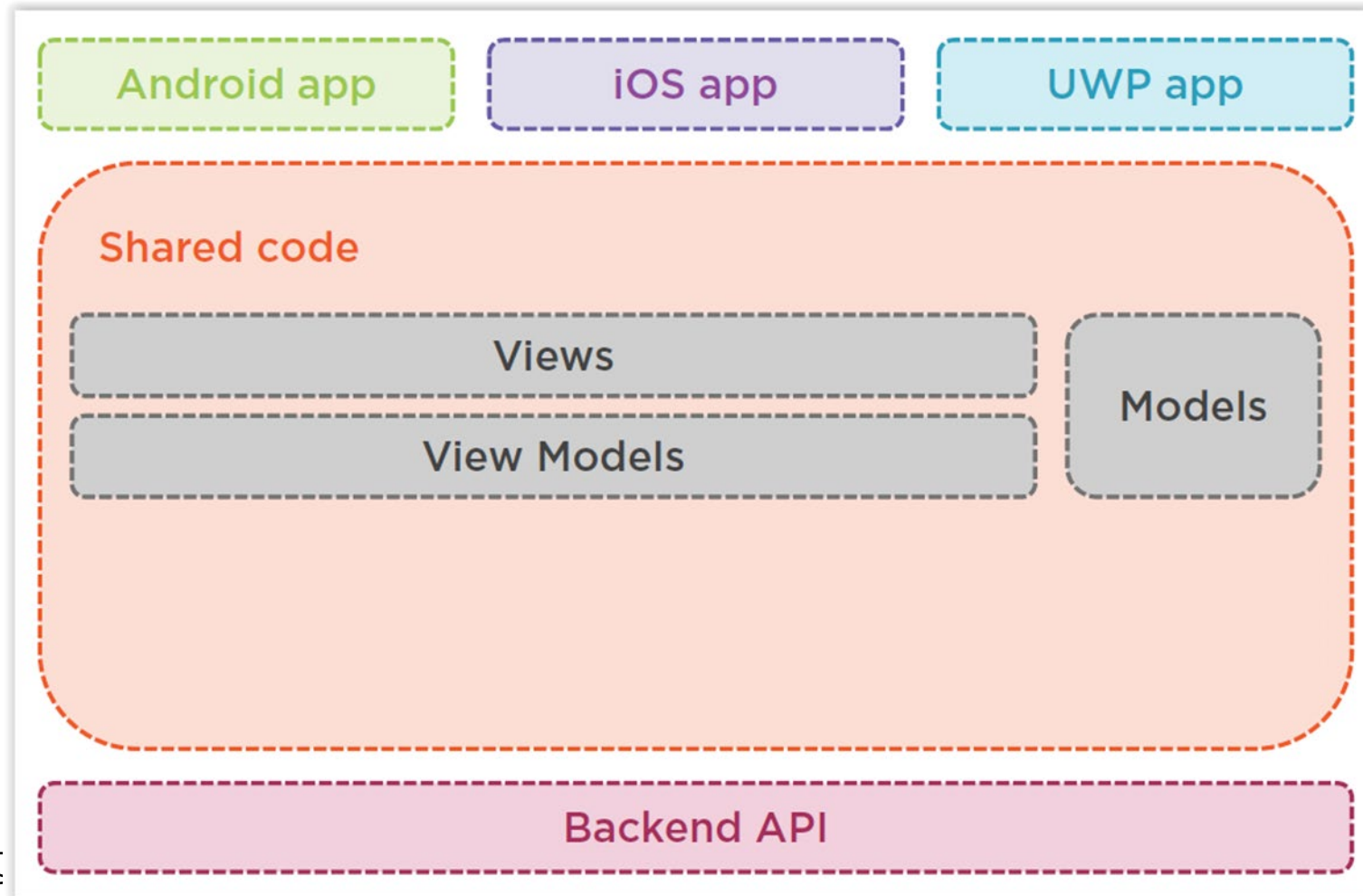
“Seperation of concerns”:
elk onderdeel doet slechts één ding

Architectuur – MVVM Doel



Seperation of concerns
Testability
Maintainability
Loose coupling
Code sharing

Architectuur



View

- UI: in .NET MAUI zijn dit onze content pages met hun view componenten
- Bevat GEEN logica, enkel via binding gekoppeld aan ViewModel

Model

- Object dat de data vertegenwoordigt

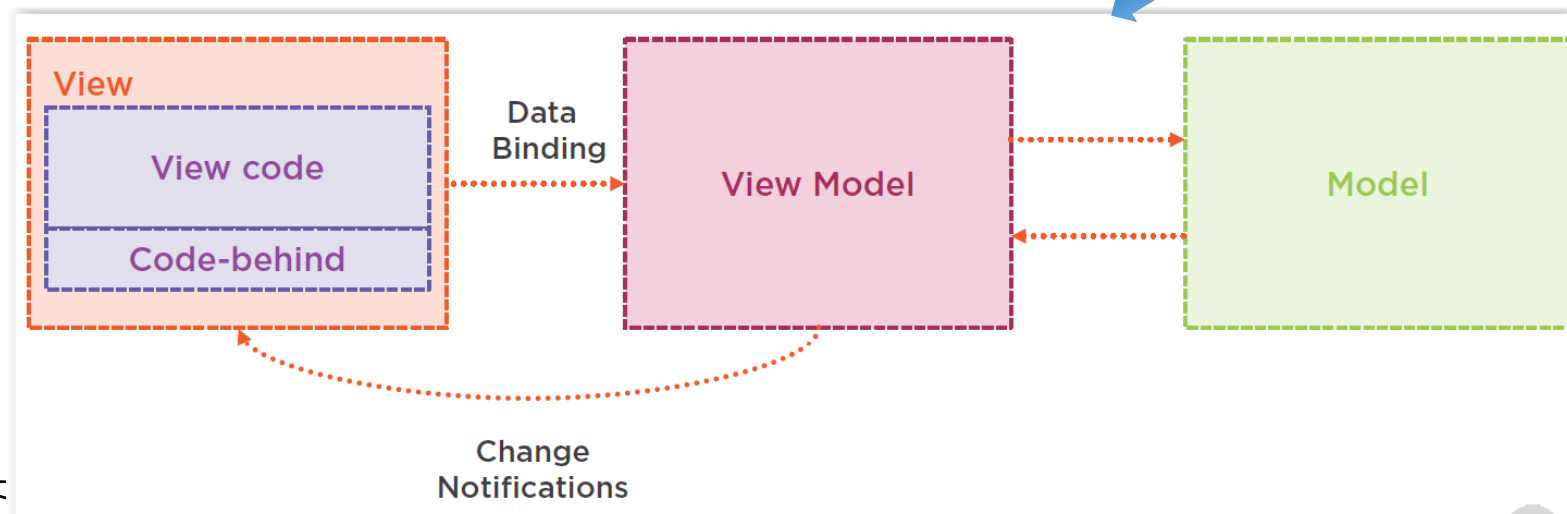
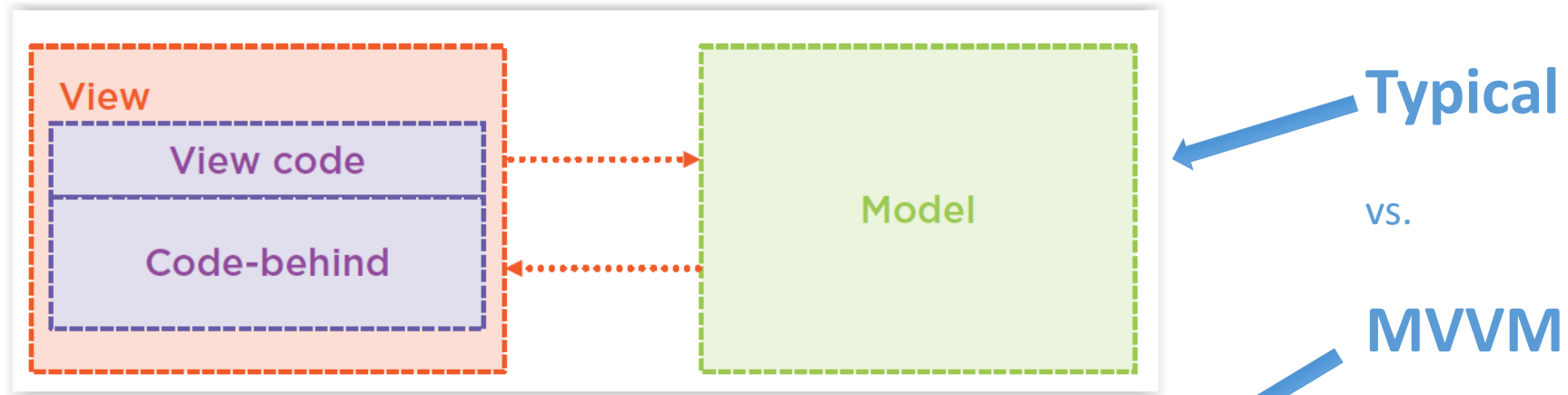
ViewModel

- Koppeling tussen de View en het Model
- Bevat de logica die nodig is om de View van data te voorzien en interacties te verwerken (commands)

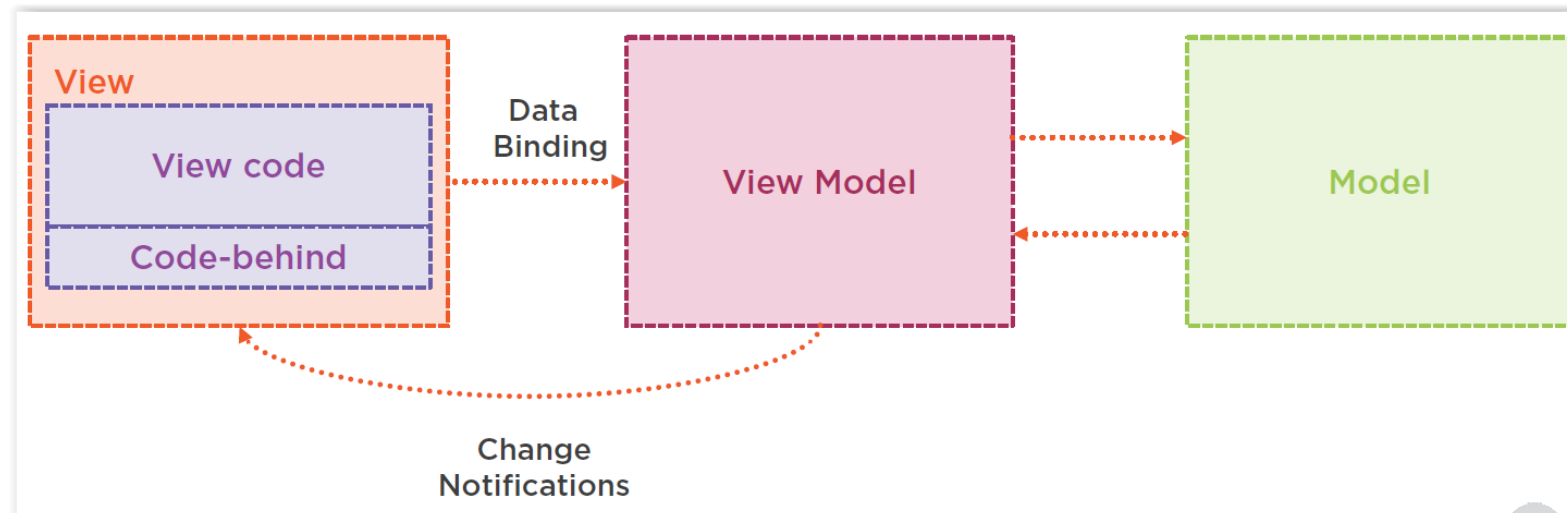
MVVM: Labo

- Open *Labo – MVVM* in de Blackboard cursus
- Accepteer de assignment
- Open je repository
- Maak een clone van je repository
- Open het README.md bestand
- Voltooi Deel 1

Traditional vs. MVVM

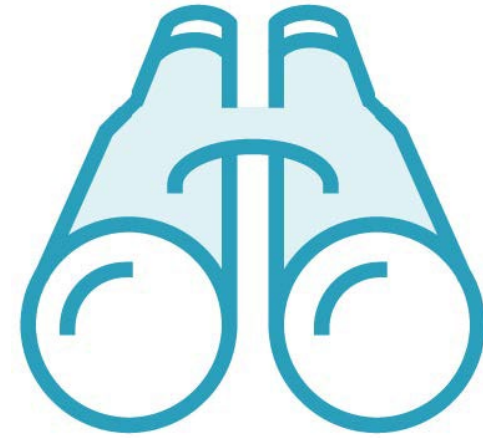


MVVM Pattern



MVVM - View

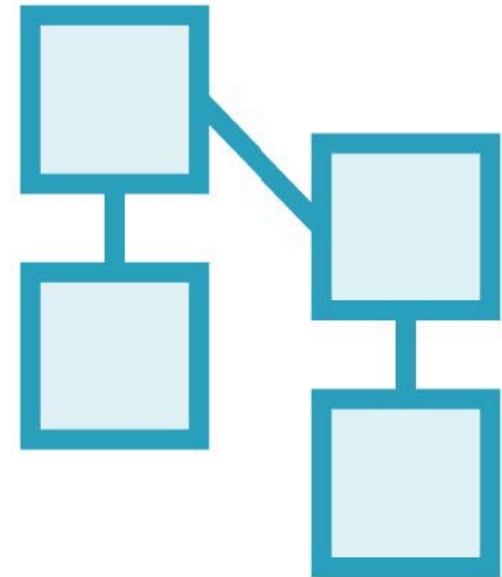
- User Interface
 - Eenvoudig, vooral XAML
- **Databinding** naar het **View-Model**
- Geen business logic!
=> Bijna geen code in de code-behind



```
public partial class GameListView : ContentPage
{
    public GameListView(GameListViewModel vm)
    {
        InitializeComponent();
        BindingContext = vm;
    }
}
```

MVVM - View-Model

- Link tussen de View en het Model
- Stelt *state* (properties) en *operations* (commands) ter beschikking
- “Manager” van de applicatie
- **Geen rechtstreekse link** naar UI-elementen!
- “Gewone” C# klasse
- INotifyPropertyChanged!



MVVM - View-Model

```
public class GamesViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler? PropertyChanged;

    protected void OnPropertyChanged([CallerMemberName] string name = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
    }

    private Game selectedGame;
    public Game SelectedGame
    {
        get => selectedGame;
        set
        {
            selectedGame = value;
            OnPropertyChanged();
        }
    }

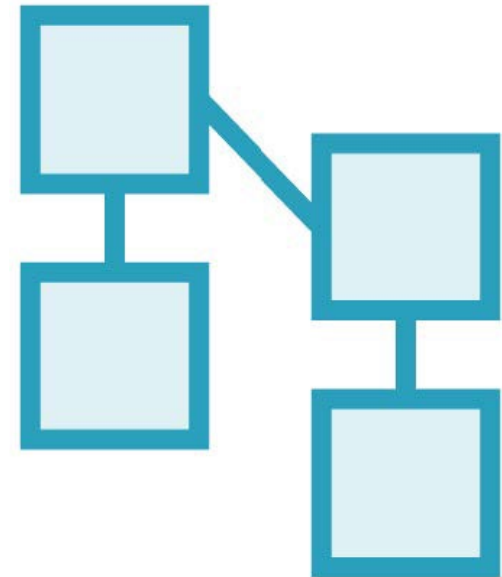
    public ObservableCollection<Game> AllGames { get; set; }

    public GamesViewModel()
    {
        this.AllGames = new ObservableCollection<Game>();
    }
}
```

MVVM - View-Model

- Commands:
 - Property van ViewModel met { get; private set; }
 - Initialiseren in de constructor
 - Opgeslagen in een ICommand reference
 - Methode meegeven die de actie uitvoert
 - Optioneel: private bool CanExecute()
- Binding met View via Command attribuut
- Optioneel: CommandParameter

```
<Button Text="Save" Command="{Binding SaveCommand}"></Button>
```



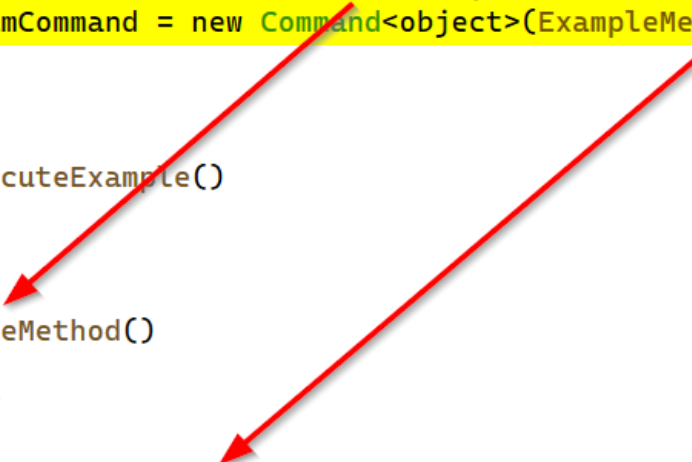
MVVM - View-Model

```
public class CommandsViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler? PropertyChanged;

    public ICommand ExampleCommand { get; private set; }
    public ICommand ExampleWithParamCommand { get; private set; }

    public CommandsViewModel()
    {
        ExampleCommand = new Command(ExampleMethod, CanExecuteExample);
        ExampleWithParamCommand = new Command<object>(ExampleMethodWithParam);
    }

    private bool CanExecuteExample()
    {
        return true;
    }
    private void ExampleMethod()
    {
        //Do some stuff
    }
    private void ExampleMethodWithParam(object value)
    {
        //Do some stuff with value
    }
}
```



MVVM - Model

- Data
 - Geeft de data waar het View-Model mee kan werken
 - (Vereenvoudigde) voorstelling van de data zoals deze zich in de database bevindt



MVVM - Model

```
public class Game : INotifyPropertyChanged
{
    private string name;
    private int rating;

    public string Name
    {
        get => name;
        set
        {
            name = value;
            OnPropertyChanged();
        }
    }

    public int Rating
    {
        get => rating;
        set
        {
            rating = value;
            OnPropertyChanged();
        }
    }

    public event PropertyChangedEventHandler? PropertyChanged;

    protected void OnPropertyChanged([CallerMemberName] string name = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
    }
}
```



MVVM: Labo

- Deel 2

<https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/data-binding/relative-bindings>



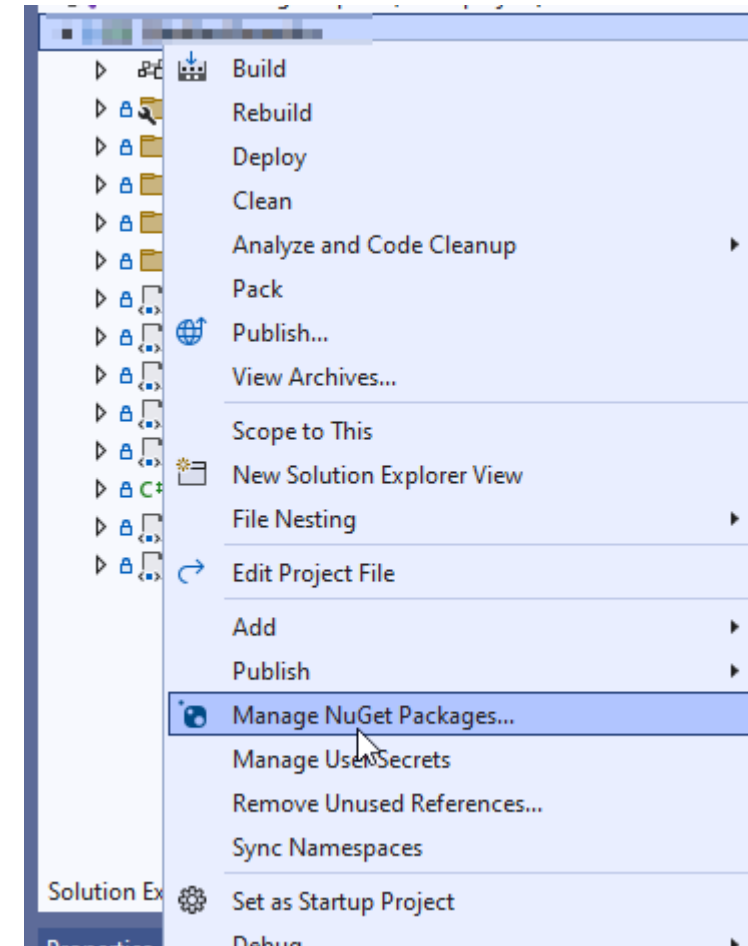
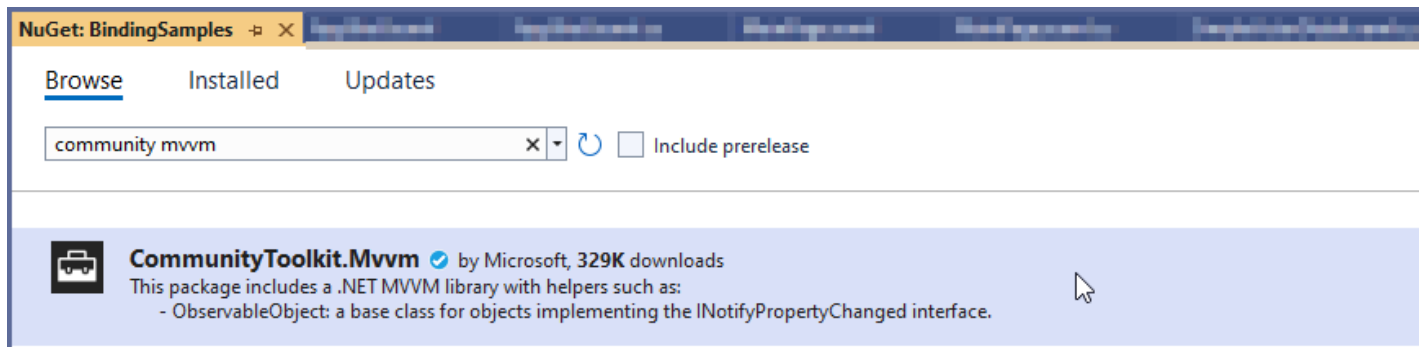
CommunityToolkit.Mvvm

CommunityToolkit.Mvvm

- MVVM Library
 - Documentatie: <https://learn.microsoft.com/en-us/dotnet/communitytoolkit/mvvm>
- Code generators
 - => Zelf minder code schrijven!
- Bijvoorbeeld:
 - ObservableObject: INotifyPropertyChanged wordt automatisch geïmplementeerd
 - [RelayCommand]: maakt een *command* van methods
- **Opgelet:** classes moeten als *Partial* geïmplementeerd worden
- Installeren via NuGet

CommunityToolkit.Mvvm

Nuget install:



CommunityToolkit.Mvvm

- **ObservableObject**


- O.a.: INotifyPropertyChanged wordt geïmplementeerd voor ons
- O.a.: SetProperty-method beschikbaar (kent value toe + vuurt event!)

0 references

```
public class ActorAdvanced : ObservableObject
{
    private string name;
    0 references
    public string Name {
        get => name;
        set => SetProperty(ref name, value);
    }
}
```

CommunityToolkit.Mvvm

- attribuut **[ObservableProperty]**
 - Gaat code genereren in onze klasse => onze klasse moet *partial* zijn!

 `class` CommunityToolkit.Mvvm.ComponentModel.ObservablePropertyAttribute (+ 1 overload)
An attribute that indicates that a given field should be wrapped by a generated observable property. In order to use this attribute, the containing type has to inherit from `ObservableObject`, or it must be using `ObservableObjectAttribute` or `INotifyPropertyChangedAttribute`. If the containing type also implements the `System.ComponentModel.INotifyPropertyChanged` (that is, if it either inherits from `ObservableObject` or is using `ObservableObjectAttribute`), then the generated code will also invoke `ObservableObject.OnPropertyChanged(System.ComponentModel.PropertyChangingEventArgs)` to signal that event.

This attribute can be used as follows:

```
partial class MyViewModel : ObservableObject
{
    [ObservableProperty]
    private string name;

    [ObservableProperty]
    private bool isEnabled;
}
```

And with this, code analogous to this will be generated:

```
partial class MyViewModel
{
    public string Name
    {
        get => name;
        set => SetProperty(ref name, value);
    }

    public bool IsEnabled
    {
        get => isEnabled;
        set => SetProperty(ref isEnabled, value);
    }
}
```

```

12 references
public class Actor : INotifyPropertyChanged
{
    private string name;
    6 references
    public string Name
    {
        get => name;
        set
        {
            name = value;
            RaiseEventPropChanged(nameof(Name));
        }
    }
    private string firstName;
    5 references
    public string FirstName { get => firstName; set
    {
        firstName = value;
        RaiseEventPropChanged(nameof(FirstName));
    }
    }
    private int birthYear;
    4 references
    public int BirthYear { get => birthYear; set
    {
        birthYear = value;
        RaiseEventPropChanged(nameof(BirthYear));
    }
    }
    private string profilePicture;
    4 references
    public string ProfilePicture { get => profilePicture; set
    {
        profilePicture = value;
        RaiseEventPropChanged(nameof(ProfilePicture));
    }
    }
    public event PropertyChangedEventHandler PropertyChanged;
    4 references
    public void RaiseEventPropChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}

```



```

7 references
public partial class ActorAdvanced : ObservableObject
{
    [ObservableProperty]
    private string name;
    [ObservableProperty]
    private string firstName;
    [ObservableProperty]
    private int birthYear;
    [ObservableProperty]
    private string profilePicture;
}

```


CommunityToolkit.Mvvm

Voorbeeld

```
public class BaseViewModel : INotifyPropertyChanged
{
    bool isBusy;
    string title;

    4 references
    public bool IsBusy
    {
        get => isBusy;
        set
        {
            if (isBusy == value)
                return;
            isBusy = value;
            OnPropertyChanged();

            OnPropertyChanged(nameof(IsNotBusy));
        }
    }

    1 reference
    public bool IsNotBusy => !IsBusy;
}
```



```
2 references
public partial class BaseViewModel : ObservableObject
{
    [ObservableProperty]
    [AlsoNotifyChangeFor(nameof(IsNotBusy))]
    bool isBusy;

    [ObservableProperty]
    string title;

    0 references
    public bool IsNotBusy => !IsBusy;
}
```

CommunityToolkit.Mvvm

- attribuut **[RelayCommand]**
 - Genereert code die een 'simpele' method omvormt naar een Command!

[RelayCommand]

2 references

```
public async void AddGame()
{
    Game newGame = new Game();
    newGame.Title = "New Game";
    AllGames.Add(newGame);
    await gameService.AddGame(newGame);
}
```

```
partial class MyViewModel
{
    [RelayCommand]
    private void GreetUser(User? user)
    {
        Console.WriteLine($"Hello {user.Name}!");
    }
}
```

And with this, code analogous to this will be generated:

```
partial class MyViewModel
{
    private RelayCommand? greetUserCommand;

    public IRelayCommand GreetUserCommand =>
        greetUserCommand ??= new RelayCommand(GreetUser);
}
```

MVVM: Labo

- Deel 3

<https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/data-binding/relative-bindings>



Relative bindings

Relative bindings

- Wordt gebruikt om bindings te maken tussen UI-elementen die zich op verschillende plekken in de visuele boom bevinden
- Bijvoorbeeld binding naar een Command van het ViewModel vanuit een CollectionView die als DataType het model heeft
- Syntax:

```
{Binding Source={RelativeSource AncestorType={x:Type <ParentViewModel>}},  
    Path=<TargetProperty>}
```

MVVM: Labo

- Deel 4

<https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/data-binding/relative-bindings>

Extra bronnen

- <https://learn.microsoft.com/en-us/dotnet/maui/xaml/fundamentals/mvvm>
- <https://www.youtube.com/watch?v=XmdBXuNPShs>
- <https://www.youtube.com/watch?v=5Qga2pniN78>