# MVVM: Model-View-ViewModel

Contents

[MVVM: Model-View-ViewModel 1](#_Toc432510294)

[Model- View tekortkomingen: probleemstelling 2](#_Toc432510295)

[Model - View – ViewModel oplossingen (MVVM) 2](#_Toc432510296)

[ViewModel properties op meerdere plaatsen binden 2](#_Toc432510297)

[MasterDetailVM 2](#_Toc432510298)

[MasterDetailVM en databinding 3](#_Toc432510299)

[ViewModel properties logisch op elkaar afstemmen 3](#_Toc432510300)

[Filter pagina (wireframe) 3](#_Toc432510301)

[FilterVM 4](#_Toc432510302)

[FilterVM DataBinding 4](#_Toc432510303)

[MVVM en asynchrone operaties 5](#_Toc432510304)

[asynchrone MVVM oplossing 5](#_Toc432510305)

[Asynchrone MVVM oplossing toegepast op StudentModulePunt 6](#_Toc432510306)

[Asynchrone MVVM, StudentModulePunt, Model klasses 6](#_Toc432510307)

[Asynchrone MVVM, StudentModulePunt, ViewModel 6](#_Toc432510308)

[Asynchrone MVVM, StudentModulePunt, Repositories 7](#_Toc432510309)

[MVVM StudentModulePunt goed toegepast? 7](#_Toc432510310)

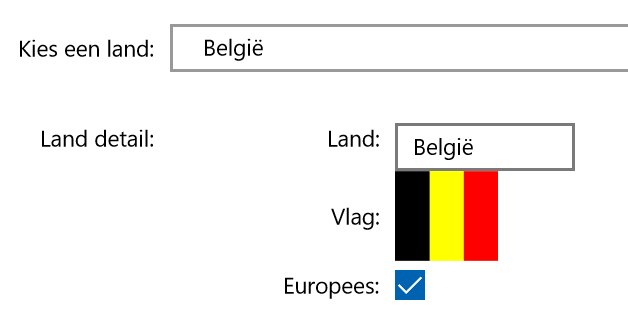
[Asynchrone MVVM oplossing toegepast op de Parking 8](#_Toc432510311)

[Round Up 8](#_Toc432510312)

## Model- View tekortkomingen: probleemstelling

De DataTemplates zoals geïntroduceerd in de voorbije les koppelen instanties van een Model- klasse aan de XAML- code van de View door middel van Bindings. De huidige benadering creeërt volgende situatie:

* Wijzigingen in de UI worden (indien TwoWay Binding) aan het Model object doorgegeven;
* Wijzigingen in het Model object worden (indien INotifyPropertyChanged) aan de UI doorgegeven.

Wat momenteel ontbreekt is een logische interactie tussen verschillende gegevens op dezelfde UI:

Het kiezen van het land België in de lijst toont automatisch de detail van het gekozen item. De huidige benadering realiseert dit door het opvangen van het SelectionChanged Event van de combobox, waarna het gepaste land gevisualiseerd wordt. In een MVVM benadering zullen dergelijke logische verbanden via het ViewModel gelegd worden!

## Model - View – ViewModel oplossingen (MVVM)

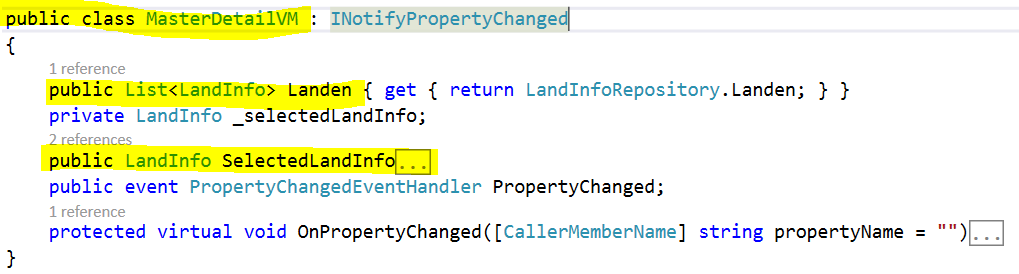
### ViewModel properties op meerdere plaatsen binden

We introduceren een aparte verzameling klasses, de ViewModels, die in hun eigen folder (ViewModels) geplaatst worden. Elke ViewModel klasse verzorgt het aanreiken van de nodige gegevens van een Page (View), en implementeert de nodige algoritmes om een logische samenhang tussen de gegevens te waarborgen. Per ViewModel maken we exact één application wide static resource van deze klasse. Deze zal als datacontext van de overeenkomstige View (Page) ingesteld worden

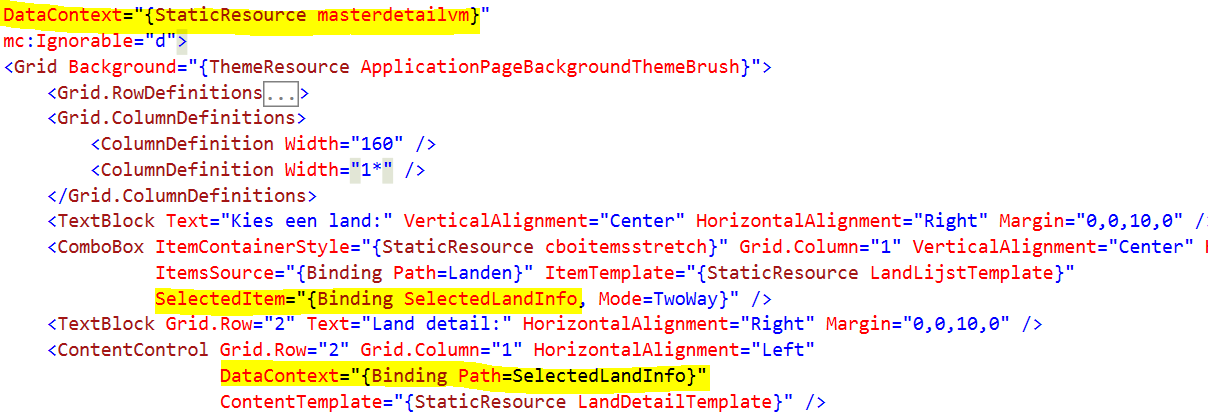
#### MasterDetailVM

Voor bovenstaande situatie ontwikkelen we een ViewModel met volgende properties:

* Landen : List<LandInfo>: een property die de volledige lijst landen teruggeeft. Deze zal gebruikt worden om de inhoud van de ComboBox te vullen
* GeselecteerdLandInfo: LandInfo: een property die één LandInfo teruggeeft. Deze property wordt TwoWay gebind aan de SelectedItem property van de ComboBox. Het kiezen van een land in de ComboBox zal de property in het ViewModel aanpassen. Verder is deze property ook gebind aan de DataContext property van de ContentControl die de detail zal tonen.



#### MasterDetailVM en databinding



De DataContext van de Page wordt ingesteld op masterdetailvm, een StaticResource van de overeenkomstige ViewModel klasse MasterDetailVM.

Bemerk dat bovenstaande eenvoudige situatie ook bekomen kan worden door de DataContext van de ContentControl te binden aan de SelectedItem property van de ComboBox (dit omzeilt de interactie met het ViewModel).

In een normaal ontwikkelproces zullen we eerst een WireFrame uitdenken, en op basis daarvan een ViewModel opzetten.

### ViewModel properties logisch op elkaar afstemmen

Algemener dan de vorige situatie, waarbij éénzelfde property op meerdere plaatsen werd gebind, is het geval waarbij één of meerdere properies worden beïnvloedt door het wijzigen van een andere property. Indien beide properties gebind zijn hoeven we geen extra C# code te schrijven om het onderlinge verband tussen beide properties te visualiseren.

#### Filter pagina (wireframe)

We voorzien twee filter checkboxen:

* Worden de landen in Europa getoond?
* Worden de landen niet in Europa getoond?

Beide of geeneen kunnen aangevinkt zijn (vandaar de CheckBoxen en geen RadioButtons)

Onder de filters komt een ListBox waarin de gefilterede landen worden geplaatst. Ze worden gevisualiseerd met dezelfde LandTemplate als deze van de vorige topic.

#### FilterVM

De property GefilterdeLijst levert de landen die aan de huidige filters voldoen. Het wijzigen van de Filter properties (IsEuropaFilter en IsNietEuropaFilter) moet dan ook het wijzigen van GefilterdeLijst tot gevolg hebben. Dit wordt geïmplementeerd door in de setter van de filter properties OnPropertyChanged(“GefilterdeLijst”) op te roepen.

In de code aan de rechterkant moet IsNietEuropaFilter natuurlijk verder uitgewerkt worden, gelijkaardig aan de code van IsEuropaFilter.

#### FilterVM DataBinding



Bemerk dat enkel de Properties die door de gebruiker kunnen aangepast worden gebind moeten zijn met Mode=TwoWay



Een tekortkoming van bovenstaande situatie is dat een land zichzelf niet onmiddellijk kan wegfilteren door de filtervoorwaarde aan te passen: indien we de IsEuropees checkbox van België afzetten terwijl de Europese landen worden gefilterd wordt België nog altijd getoond, tot de filtervoorwaarden worden aangepast:

## MVVM en asynchrone operaties

In een real life scenario wordt informatie asynchroon uit een database of van een webservice opgehaald. Omdat properties niet asynchroon gedefinieerd kunnen worden veroorzaakt dit een programmeer mismatch: de View wil binden aan een property wiens gegevens nog asynchroon door een andere methode opgehaald moeten worden.

### asynchrone MVVM oplossing

Vooraleer de oplossing in detail op te bouwen moeten een aantal zaken duidelijk gesteld worden:

* De Model- klasses zijn onwetend (agnostic) omtrent de situatie waarin ze gebruikt zullen worden. Het is dan ook niet de bedoeling om in de Model klasses features te voorzien om de asynchrone problematiek op te lossen;
* De views binden zich met properties van hun ViewModel, niet met properties van de repositories (zoals vorige week, voor het MVVM verhaal wel nog het geval was);
* De repositories zijn verantwoordelijk voor de asynchrone datamanipulaties. Asynchrone operaties kunnen geen properties zijn!

De werkwijze die we zullen volgen om een asynchroon MVVM pattern uit te bouwen is de volgende:

* De Views binden hun properties aan de ViewModel properties, waar nodig met TwoWay binding;
* De ViewModel klasses implementeren de INotifyPropertyChanged interface om de Bindings te verwittigen indien een propertywaarde wijzigt;
* De properties van het ViewModel worden ondersteunt door een backing field (een field waarin de waarde van de property wordt bijgehouden) indien deze property door de gebruiker kan gewijzigd worden (indien TwoWay gebind);
  + Op deze manier kunnen we een setter met extra logica programmeren
* Indien de waarde van een property asynchroon opgehaald wordt gebeurt dit in een aparte methode, die de overeenkomstige repository oproepen zal awaiten (deze ViewModel methode is dan ook een async methode);
  + De verzamelde gegevens worden bewaard in de properties;
  + Indien een property waarde hierdoor wijzigt zal de OnPropertyChanged methode met de juiste propertynaam als parameter worden opgeroepen. Dit kan eventueel ook in de setter van de property gebeuren, en geniet mijn voorkeur indien er een backing field gedefinieerd werd (omwille van het éénmalige coderen op de juiste plaats ☺)

Opmerkingen omtrent de nieuwe oplossing:

* We hebben geen nood meer aan repository static resources
* We hebben nu wel nood aan ViewModel static resources. Deze definiëren we ook best via de App.xaml file.

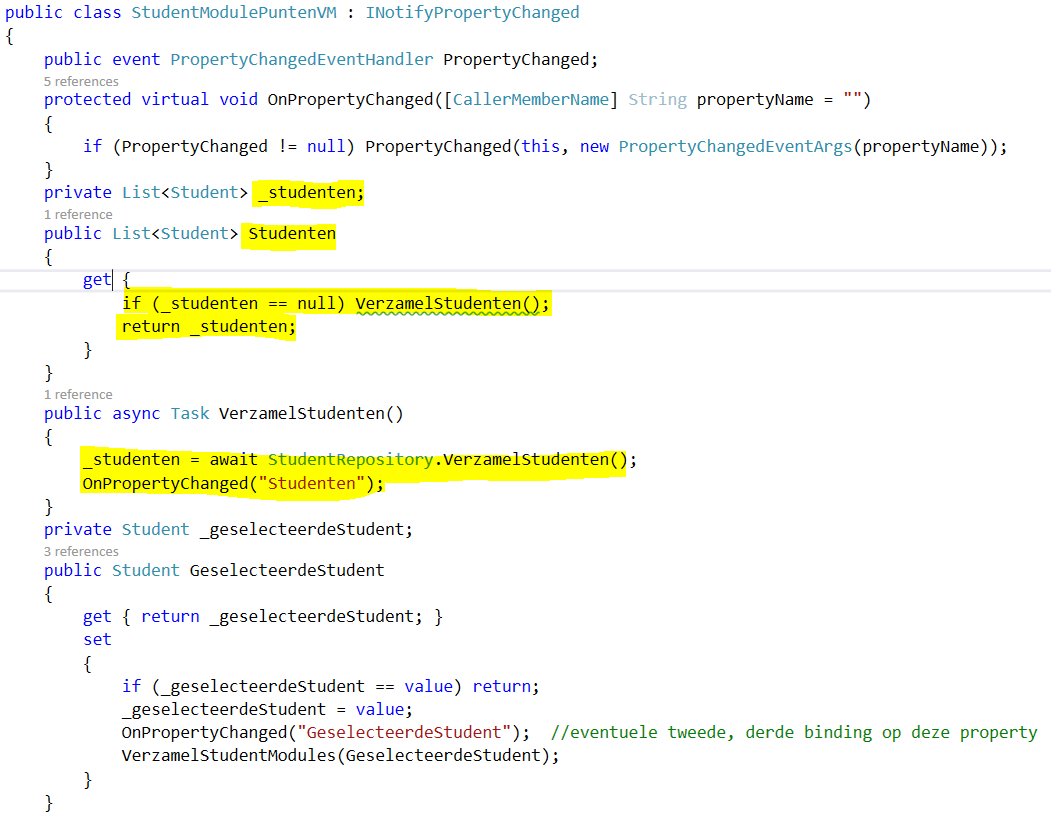
### Asynchrone MVVM oplossing toegepast op StudentModulePunt

#### Asynchrone MVVM, StudentModulePunt, Model klasses

* We wijzigen niets aan de Model Klasses van de vorige oplossing: Student, StudentModulePunt en (optioneel) ModuleStatistiek

#### Asynchrone MVVM, StudentModulePunt, ViewModel

Het eerste stuk van de ViewModel klasse werd hieronder afgedrukt:



Op basis van de gekende UI voorzien we in de StudentModulePuntVM klasse volgende properties:

* Studenten: List<Student>: de nu gekende lijst van studenten:
* Async VerzamelStudenten():
  + Verzamelt de studenten in de property Studenten;
  + Na het verzamelen wordt OnPropertyChanged(“Studenten”) opgeroepen om aan te geven dat de Studenten property werd gewijzigd
  + Deze methode zullen we oproepen de eerste keer dat de Studenten worden opgehaald;
* GeselecteerdeStudent: Student: deze property zal TwoWay gebind worden aan de SelectedItem property van de studentkeuze control;
  + Ook de eigen PropertyChanged wijziging moet worden doorgegeven, voor eventuele andere bindingen op dezelfde GeselecteerdeStudent property
  + Bij het wijzigen van deze property moeten ook StudentModulePunten van de geselecteerde student worden aangepast

Het tweede stuk van de klasse is gelijkaardig opgebouwd:



Terugverwijzend naar de Asynchrone MVVM oplossing kunnen we dus stellen:

* Het verzamelen van informatie via asynchrone methodes zal op het einde van de uitvoering, via OnPropertyChanged oproepen, de gerelateerde Bindings verwittigen dat de UI aangepast moet worden.

#### Asynchrone MVVM, StudentModulePunt, Repositories

* StudentRepository.VerzamelStudenten : List<Student>: deze asynchrone methode verzamelt alle studenten
* StudentModulePuntRepository.VerzamelStudentStudentModulePunten(Student student): deze asynchrone methode verzamelt alle StudentModulePunt objecten van een student;

De repository klasses dienen om de ModelView klasses te ondersteunen.

#### MVVM StudentModulePunt goed toegepast?

Hoe meer MVVM correct werd toegepast, hoe minder C# code in de code- behind files wordt gevonden. Voor de hier gepresenteerde StudentModulePunt oplossing is de code behind leeg, dus meer MVVM kunnen we hier niet toepassen.

### Asynchrone MVVM oplossing toegepast op de Parking

Volledig gelijkaardig ☺

## Round Up

