# Architectuur oefening

De voorbije lessen werd MVVM technisch geïntroduceerd. We waren vooral geïnteresseerd om de Binding technieken (zowel data als command) onder de knie te krijgen, zonder veel aandacht te besteden aan de architectuur van onze oplossing. Deze les zullen we de aangeleerde technieken (ook Object Oriented Programming) in een goede architectuur kaderen.

Het bijhorende theorie document beschrijft het proces om van een oplossing zoals we deze maken in de voorbije lessen naar de eind architectuur toe te werken. Dit verklaart alle stappen (Ok, daar leren we van) maar brengt flink wat extra werk en tijdverlies met zich mee. Het stappenplan van deze oefening werkt onmiddellijk toe naar de correcte architectuur.

## Architectuur oefening: GeoPhotoApp

### GeoPhotoApp: features

Deze app toont in een overzichtscherm foto’s die uit de Flickr service werden opgehaald met behulp van een tag. De tags worden gekozen op basis van geografische termen (continent, land, stad) die door de WorldWS webservice worden aangeleverd. De toepassing laat toe om een gekozen foto in detail te tonen en daarna terug te keren naar het overzichtscherm. Uiteraard zouden we ook extra features kunnen toevoegen aan het detail scherm: het bewaren van foto’s, ...

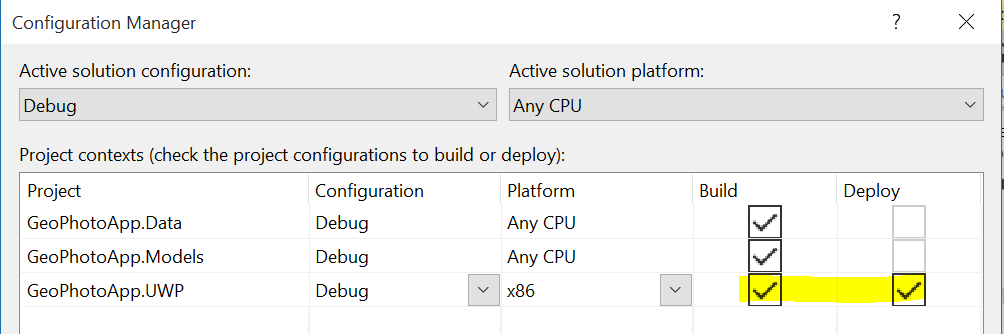
### GeoPhotoApp: solution opzet

* Maak een solution met 3 projecten (twee class libraries en één Blank app). U vindt deze types onder ‘Windows.Universal’. Leg de gepaste referenties tussen de drie projecten en voorzie reeds de folders die u nodig zal hebben. Indien u folders vergeet zal u ze later gemakkelijk kunnen toevoegen.
* In het Data project zullen we JSON tekst tot objecten deserialiseren. Installeer hiervoor het juiste package
* In het UWP project zullen we MVVM toepassen. Installeer het juiste package dat hievoor ondersteuning biedt

### GeoPhotoApp: Flickr code

* Onderzoek de flickr api (u vindt de Url in het word document), test wat met de mogeljkheden (via dezelfde url) en maak op basis van de bekomen resultaten en de json2csharp website klasses voor de bekomen json resultaten. De code van deze klasses komt in de Flickr folder van de models library
  + Dit zijn de eerste models die we maken (welk project? In welke folder?). Deze moeten allemaal erven van de ModelsBaseClass die we in de Core folder plaatsen (verwijder de laatste qualifier)
  + Breidt de Photo class uit met de property PhotoUrl (zie cursus tekst) omdat we deze property zullen binden met de Source property van een Image
  + De RootObject class renamen we tot FlickrData (indien u wenst dat de namen kloppen met de voorbeeldcode in de cursustekst)
* Codeer een Flickr repository class (welk project? In welke folder?). De code vindt u in de cursustekst. Zoals toegelicht in de cursustekst is dit een stand- alone class. De **andere** repositories die we ontwikkelen zullen erven van een base class.
  + Om een Flickr repository op te zetten zal u een Api key nodig hebben. In de cursustekst wordt de url vermeld (zoek op **app garden**) waar u een Api key kan aanvragen

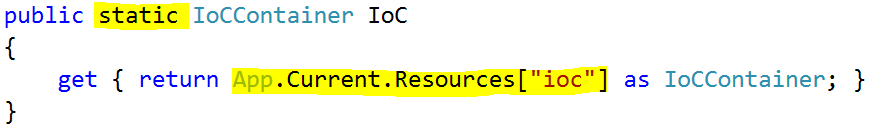
### GeoPhotoApp: flickr test code

* Maak een page PhotoOverviewPage en bijhorend ViewModel PhotoOverviewViewModel. Het ViewModel heeft een Photos property die een Ienumerable<Photo> aanlevert (gebruik de repository klasse om deze op te vragen (async pattern gebruiken)). Plaats op de page een GridView property en bind diens ItemsSource aan de Photos property van een StaticResource van het bijhorende viewmodel (we maken hiervoor een StaticResource in App.xaml). U zal uiteraard ook een DataTemplate moet maken voor de GridView (in te stellen via ItemTemplate, die bevat een Image wiens Source gebind wordt aan de PhotoUrl property van de getoonde Photo (hint: in het overzicht wens je een template die de foto’s klein toont te gebruiken, beperk de Width en Height van de Grid in de DataTemplate tot 100 (pixels))).
  + Maak een BaseViewModel klasse die de INPC interface implementeert. Al onze ViewModels moeten hiervan erven.
* Plaats op de MainPage een Frame (in de Grid), geef die een naam en in het Loaded event van de page zullen we de Content van dit frame instellen op een nieuw object van de klasse PhotoOverviewPage (een niet MVVM manier van werken, enkel om te testen).
* Verifieer dat Visual Studio weet welke projecten gebuild en gedeployed moeten worden: Build.Configuration Manager (verifieer ook dat het UWP project als startup project gedefinieerd is)
* 

Deze toepassing zou momenteel moeten werken

### Dependency Injection en IoCContainer

We zullen onmiddellijk de puntjes op de i plaatsen: een viewmodel zal zijn repositories geinjecteerd krijgen via de constructor:

* Maak een interface voor de FlickrRepository klasse (gebruik Visual Studio) en voorzie een parameter (in de ctor van het ViewModel) en een field (of property) in het ViewModel om deze te bewaren. Vervang de code waar de Repository klasse wordt gebruikt door code die het field gebruikt.
  + Nu kan er geen StaticResource meer van deze klasse gemaakt worden, wat op dit moment een compiler fout genereert (in App.xaml)
* Terwijl we toch bezig zijn: maak ook een interface voor het ViewModel
* Implementeer een IoCContainer klasse (u vindt voorbeeld code in het theorie document) en registreer beide zonet gemaakte interfaces met hun ondersteunende klasse. U hoeft niet onmiddellijk een object aan te maken.
  + Voorzie in de IoCContainer class een static property die een IoC resource teruggeeft. Deze property zullen we gebruiken om in C# snel toegang tot de IoCContainer resource te krijgen: we kunnen **IoCContainer.IoC** tikken  
    
  + Voorzie voor elk type dat de interface registreert (interface of klasse) een property met dezelfde naam die het object van dit type teruggeeft (voorbeeld code vindt u in de cursus).
* Verwijder het ViewModel static resource (dit compileert toch niet meer) en voeg een IoCContainer resource toe met als key “ioc”
* Vergeet niet in de eerder gemaakte page de DataContext nu te binden met de ioc- static resource en volg het pad naar de juiste property (er werd een property gemaakt voor elk type).

Start de toepassing terug op om te testen of alles nog werkt. Indien dit zo is hebben we nu de basis infrastructuur gerealiseerd waarbinnen we verder zullen werken.

### PhotoDetailPage en PhotoDetailViewModel

Ontwikkel een PhotoDetailPage en zijn bijhorende ViewModel:

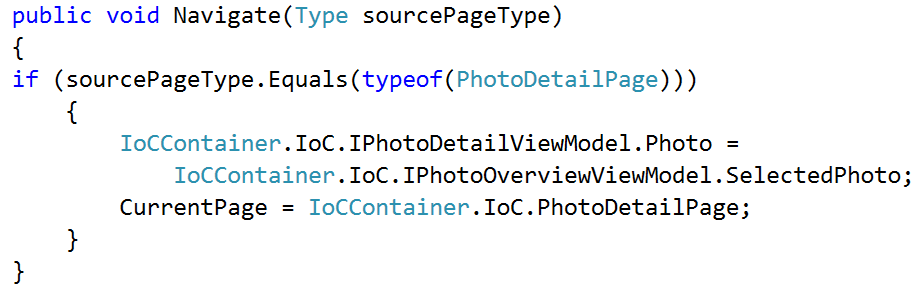
* De DetailPage toont gegevens van één Photo. Deze Photo is dan ook een property op het ViewModel. De DetailPage toont de foto groot, en zet onderaan de titel van die Photo. Ontwikkel een Page met de juiste bindings voor de Photo property van het ViewModel
  + Vergeet niet voor het viewmodel een interface op te zetten
  + Vergeet niet die interface te registreren in de IoCContainer
  + Vergeet niet een property voor deze interface aan te maken in de IoCContainer
* Koppel de DataContext van de page aan de juiste property van de StaticResource ioc. Indien u nu pas de bindings legt (en eerst de solution rebuild) kan u gebruik maken van de intellisense.

### Navigatie tussen PhotoOverview en PhotoDetail

Indien de gebruiker op een overview foto klikt wensen we te navigeren naar een detail scherm voor de geselecteerde foto:

* Registreer beide pages in de IoCContainer. Het is mogelijk om voor de pages een interface te ontwikkelen (indien verschillende implementaties van deze pages gewenst zijn. Ikzelf meen dat dit in deze toepassing niet nodig zal zijn, ik registreer dan ook enkel de Page klasses.
* Voorzie in het OverviewViewModel een DetailCMD RelayCommand (ShowDetailCMD) met één parameter (type Object). Het is de bedoeling dat deze parameter de gekozen Photo zal bevatten.
  + vergeet niet de interface opnieuw aan te maken om deze met de nieuwe properties rekening te laten houden (wie niet goed oplet maakt een nieuwe interface in een andere file en wijzigt de bestaande interface niet!).
* Bind de het PointerReleased event van de image in de DataTemplate met dit commando. In de cursus tekst zoeken op **grote foto’s tonen** beschrijft hoe u dit in Blend kan realiseren (we zoeken het commando in de IphotoOverviewViewModel van de static resource ioc. Het commando voert de methode **ShowDetail(Photo photo)** uit (die we nog moeten uitwerken, voorzie nu al de hoofding met een lege body)
  + Vooral het stukje over de CommandParameter is interessant: we willen de getemplate Photo doorgeven aan het commando! (zelfs indien blend een verkeerd type voor de DataContext weergeeft werkt de manier van werken toch: gewoon Custom path aanduiden).
* Zet een breakpoint op de lege body van ShowDetail en test uw applicatie: start en klik op een detail photo. Het breakpoint moet bereikt worden, en het argument photo moet ingevuld zijn.
* Voorzie op het ViewModel een property SelectedPhoto waarin de Photo van de ShowDetail methode wordt getoond (voor het navigeren naar een andere pagina). Bij het navigeren zal deze property van het viewmodel gebruikt worden om de Photo van het detailviewmodel in te stellen.

Zoals in de cursus tekst werd toegelicht zullen we navigatie tussen pagina’s opzetten door middel van een ApplicationViewModel.

* Ontwikkel een ApplicationViewModel class (erf van zijn base- class) en voorzie een CurrentPage property.
  + De default CurrentPage (in te stellen in de ctor van ApplicationViewModel) is het overzichtscherm PhotoOverviewPage (op te vragen aan de IoCContainer.ioc resource)
* Wijzig de MainPage implementatie:
  + Bind de DataContext van de MainPage aan de ApplicationViewModel property van de StaticResource ioc
  + Bind de Content property van het frame aan de CurrentPage property (van de DataContext)
  + Verwijder de naam van het frame en de Loaded code in de code behind
* Voorzie de klasse ApplicationViewModel van een Navigate methode. Onderstaande screenshot is wellicht gemakkelijker te begrijpen dan de screenshot in het theorie document (het theorie document gebruikt Dependency Injection om de verschillende pages te bekomen, onderstaande screenshot vraagt de pages aan de ioc static resource)
* Omdat de navigatie soms meer inhoudt dan het wijzigen van de CurrentPage (in dit geval moet ook een Photo doorgegeven worden naar het ViewModel van de pagina waarnaartoe we navigeren) is het niet voldoende om enkel de CurrentPage property in te stellen vanuit het ViewModel.

### Navigatie uitbreiding

Voorzie op het detail scherm de mogelijkheid om terug te navigeren naar het overzichtscherm (een Back knop):

* Voorzie BackCMD op het PhotoDetailViewModel (dit commando behoeft geen parameter);
  + U zal uiteraard de navigatie methode in ApplicationViewModel moeten uitbreiden
* Koppel dit commando aan het klikken op de image.

Test deze methode.

Opmerking: navigatie en back- navigatie komt later uitgebreider terug. Dit wordt ondersteund door de UWP omgeving (en is de reden waarom we navigeren binnen een Frame).

### Designtime versus runtime objecten

De Dependency Injection benadering werd ingevoerd om op basis van voorwaarden verschillende klasses te gebruiken om een functionaliteit aan te bieden. Wijzig uw toepassing zodat de IphotoOverviewViewModel implementatie verschillend is at design time en at run time. De gemakkelijkste manier om dit te bekomen is als volgt:

* In PhotoOverviewViewModel voorzien we een virtual (overschrijfbare) methode Tags (type string) die in deze klasse de waarde “dog” teruggeeft.
* Erf van deze klasse en maak de klasse PhotoOverviewDesignViewModel. Overschrijf deze methode en geef de waarde ‘cat’ terug.
  + Belangrijk: u zal ook een constructor met de nodige parameters moeten voorzien om de base constructor te kunnen oproepen
* Pas de IoCContainer aan zodat at design time de design versie wordt gebruikt en at run time de originele versie.
* Hercompileer uw programma en kijk na of at design time de fotos van de overview page wel degelijk katten tonen, en de runtime versie (na het opstarten) honden.

Omdat de property Tags nergens buiten deze twee klasses gebruikt wordt is het geen probleem dat deze niet in de interface is opgenomen.

## WorldWS uitbreiding

Op leho vindt u de broncode voor de WorldWS webservice (de data werd overgenomen uit de tutorial database bij een MySQL installatie). In het theorie document vindt u voorbeeld aanroepen om Continent, Country en City informatie te bekomen.

### World Repositories

* Maak Model klasses voor elk van de types
  + Gebruik json2csharp!
  + Vergeet niet te erven van de base modelclass
* Maak repository klasses voor de drie types. Deze moeten erven van een gemeenschappelijke base class. Het theorie document beschrijft de stappen om deze base class aan te maken.
* Maak voor elke repository klasse een interface (extract interface) en registreer deze interface in de IoCContainer en voorzie een property voor elk type.

### PhotoOverviewViewModel World uitbreiding

Het is finaal de bedoeling om op PhotoOverviewPage drie comboboxen te plaatsen,ingevuld met de continenten, de landen en de steden. Een keuze voor een continent beperkt de mogelijke landen, een keuze voor een land beperkt de mogelijke steden (dit wordt gecodeerd in het viewmodel):

* U moet een lijst property hebben voor elk type (Continenten, ..);
* U moet een Selected property hebben voor elk type (SelectedContinent, ..);

Vergeet niet de interface aan te passen! Vergeet niet in de ervende klasse de ctor aan te passen (extra parameters!)

Gebruik dependency injection in PhotoOverviewViewModel om, net zoals de IFlickrRepository de (interface) repositories van de World klasses te bekomen.

Indien een Continent, Land of Stad gekozen wordt dan wordt deze de nieuwe tag voor de flickr toepassing en moeten de fotos terug opgehaald worden!

## ArcGis uitbreiding

Voor wie echt graag programmeert: in het theorie document staat een verwijzing naar de ArcGis service. Deze kan gebruikt worden om voor een naam GIS coordinaten op te vragen. De Flickr service kan met deze extra informatie accuratere foto’s ophalen.

Succes!