WishLister

Introductie

In deze oefening voorzie je een view voor een lijst van cadeautips die je gedurende het jaar kan bewaren. Handig wanneer iemand vraagt wat je wil hebben op kerst. Bekijk de website van wishlistr om een idee te krijgen.

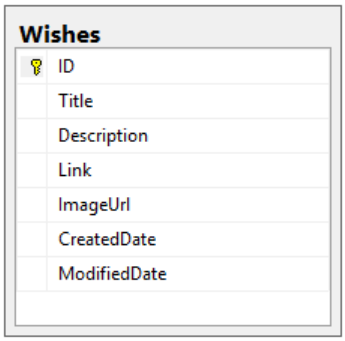
Je krijgt van ons het server gedeelte cadeau. Bedoeling is dat je de database en client kan zelf uitwerkt.

In deze oefening maak je zelf een UI en voorzie je zelf waar stijlen en datatemplates gebruikt moeten worden. **Gebruik ook de slides en labodocumenten van voorgaande lessen.**

Dit document is met opzet zo geschreven dat jullie met behulp van de voorgaande lessen oplossing moeten voorzien.

Database

De uiteindelijke data of 'wishes' moeten in een database terecht komen. Maak hiervoor een nieuwe database aan in je lokale SQL Server. Deze tabel moet volgende kolommen bevatten:



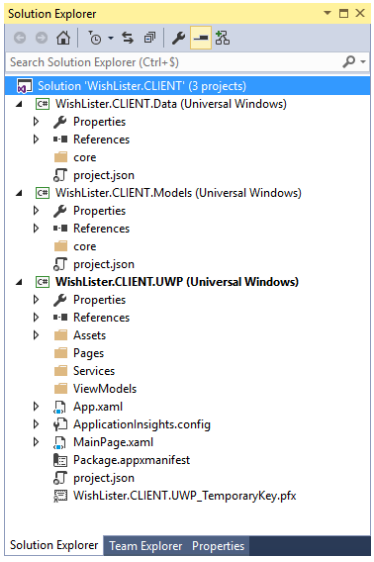
- ID (key veld en auto increment)
- Title
- Description
- Link
- ImageUrl
- CreatedDate
- ModifiedDate

In het gegeven WEB API project passen jullie de web.config indien nodig aan zodat deze mapt op jullie SQL Server installatie.

Client

Solution opzet

De bedoeling van de applicatie is dat je data via een REST Service / API ophaalt. Omdat we scheiding der componenten hoog in het vaandel willen houden splitsen we alles op in verschillende projecten:



- UI of UWP project
- Models project
- Data project

Models project

Onze modellen verzamelen we in een **Universal Windows Class Library**. Maak daarin alvast een core mapje aan waar alle basisfunctionaliteiten van een model kan geplaatst worden.

Onze modellen zijn een representatie van onze data objecten. Ons Wish model zal dan ook dezelfde eigenschappen bevatten die we terug vinden in de tabel die we gemaakt hebben in de database.

Data project

In dit project, wat net zoals het Models project een **Univeral Windows Class Library** is, maken we de verschillende classes aan die verantwoordelijk zijn voor de communicatie met onze API. Ook hier maken we een core mapje aan.

Het formaat dat zal gebruik worden om te communiceren met de API is JSON. Installeer hiervoor de juiste package om objecten in JSON formaat om te zetten en omgekeerd.

Deze classes zullen gebruik maken van de modellen. Leg een referentie naar het Models project.

UI of UWP Project

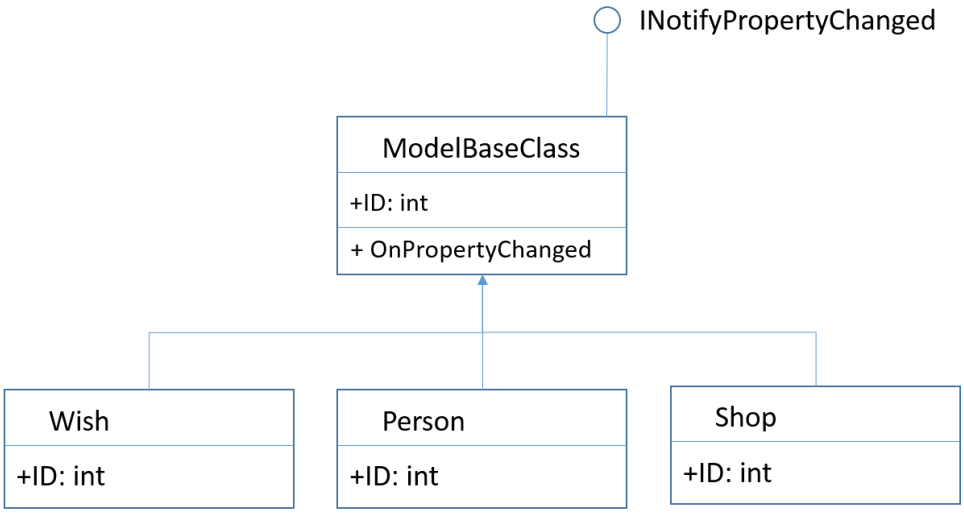
Hiervoor kiest u een **Blank Application**. Leg de gepaste referentie naar de andere projecten. In dit project zullen we ook gebruik maken van het MVVM en 'Inversion of Control' pattern. Installeer hiervoor het juiste pakket.

Client Models

ModelBaseClass

In deze oefening werken we maar met 1 object. Later kunnen hier eventueel nog andere objecten aangemaakt worden.

- Daarom maken we een BaseModelClass aan waarin we alle gemeenschappelijke properties, methodes, events, ... aanmaken.



Zorg ervoor dat deze basisklasse **niet geinstantiëerd** kan worden. We implementeren ook de **INotifyPropertyChanged** zodat geïnteresseerden op de hoogte gebracht worden wanneer een property gewijzigd wordt.

Wish

- Maak een Wish class aan die van deze basis klasse overerft.
- Voorzie de nodige properties

Repositories

De repositories worden gebruikt als poort naar onze API. Ook hier weten we niet op voorhand hoeveel repositories er kunnen gemaakt worden en is het voor onze UI niet belangrijk waar de repository zijn data vandaan haalt:

- Ons eigen API
- Amazon API
- …

Toch dient ons client project te weten welke methodes er uitgevoerd moeten worden:

```
wishes = await wishesRepo.Get();
wishesRepo.Put(NewWish);
```

We kunnen dit afdwingen door een interface te maken die een class verplicht een bepaalde signatuur te hebben.

- Maak een interface aan met de naam **IRepository** waarin je bovenstaande methodes voorziet. Baseer u op de repository interface van vorige labo's.

Interfaces

Interfaces zijn zeer belangrijk in het hedendaagse programmeren. Beschouw onderstaand voorbeeld:

"Zorg er voor dat een Schilder een lijst met objecten kan schilderen".

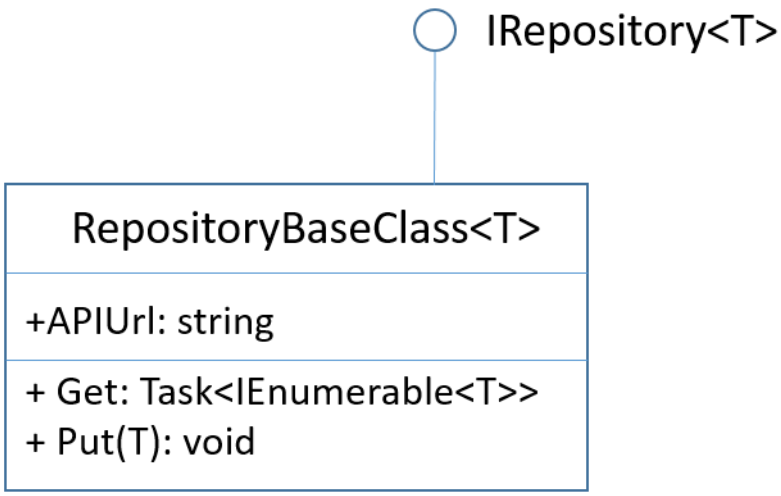
```
public class Schilder
{
    private List<IPaintable> paintableObjects;

    public Schilder()
    {
        paintableObjects = new List<IPaintable>();
    }

    public void paintAllObjects()
    {
        foreach (IPaintable paintable in paintableObjects)
        {
            paintable.paint();
        }
    }
}

public interface IPaintable
{
    void paint();
}
```

- *Bovenstaande code hoeft niet in jullie project geschreven te worden, enkel ter illustratie.*
- Zorg er ook voor dat de interface zo geschreven wordt dat deze voor elk type geldt. Zo kunnen we onze interface herbruiken voor repositories van verschillende types. We gebruiken hiervoor een generic interface.



- Omdat de implementatie van deze repository altijd hetzelfde is, krijgen jullie de code cadeau (zorg dat je deze code snapt):

```
2 references
public async Task<IEnumerable<T>> Get()
{
    using (var client = new HttpClient())
    {
        client.DefaultRequestHeaders.Accept.Clear();
        client.DefaultRequestHeaders.Accept.Add(
            new System.Net.Http.Headers.MediaTypeWithQualityHeaderValue("application/json"));

        HttpResponseMessage response = await client.GetAsync(ApiUrl) ;
        if (response.IsSuccessStatusCode)
        {
            String s = await response.Content.ReadAsStringAsync();
            List<T> result = JsonConvert.DeserializeObject<List<T>>(s);
            return result;
        }
        else
            return null;
    }
}

2 references
public void Put(T item) {
    using (HttpClient client = new HttpClient()) {
        client.BaseAddress = new Uri(ApiUrl);
        client.DefaultRequestHeaders.Accept.Clear();
        client.DefaultRequestHeaders.Accept.Add(
            new System.Net.Http.Headers.MediaTypeWithQualityHeaderValue("application/json"));
        var jsonContent = new StringContent(Newtonsoft.Json.JsonConvert.SerializeObject(item),
                                            Encoding.UTF8,
                                            "application/json");
        HttpResponseMessage response = client.PutAsync(ApiUrl, jsonContent).Result;

    }
}
```

- Maak nu een repository aan die van deze basis klasse overerft en de juiste url gebruikt. In deze repository bepaal je het type die zal gebruikt worden. We kiezen voor het Wish Type. Noem deze repository WishesRepository.

```
public class WishesRepository: RepositoryBaseClass<Wish>
{
```

- In tegenstelling tot de voorgaande oefeningen gebruiken we enkel ApiUrl

```
4 references
public override string ApiUrl
{
    get
    {
        return "http://localhost:4009/api/wishes";
    }
}
```

Extra

Wanneer je klaar bent met de volledige oefening probeer je eens een repository te schrijven om de amazon task api te bevragen. Als je alles goed geprogrammeerd hebt, dan hoef je enkel een nieuwe repository te instantiëren in je project zonder andere code aan te passen.

UWP Project

Maak volgende mappen aan met bijhorende classes

- **Services**
  - IoCContainer.cs
- **ViewModels**
  - **core**
    - BaseViewModel.cs
  - ApplicationViewModel.cs
  - AllWishesViewModel.cs
  - WishDetailsViewModel.cs
- **Pages**
  - AllWishesPage.xaml
  - WishDetailsPage.xaml

Zoals je kan zien maar we voor elke page een ViewModel aan.

Dependency Injection en IoCContainer

Omdat we niet op voorhand weten welke repository zal gebruikt worden dienen we deze te injecteren in onze ViewModels.

```
0 references
public AllWishesViewModel(IRepository<Wish> wishesRepo)
{
    this.wishesRepo = wishesRepo;
}
```

- Maak een IoCContainer klasse aan (zie voorgaande theorie lessen). Deze instantieren we dan in de app.xaml en voorzien we een property in de IoCContainer klasse zelf

```
public static IoCContainer IoC
{
    get { return App.Current.Resources["ioc"] as IoCContainer; }
}
```

- Deze container zal de instanties van de repositories, viewmodels en pages beheren.

MainPage

Op de MainPage voorzie je een SplitView control. Deze control bestaat uit 2 delen. Enerzijds uit een pane waarin we navigatie items kunnen plaatsen en anderzijds uit een content gedeelte waarin we onze frame plaatsen waarin verschillende pagina's geladen worden.

- Schrijf volgende code in de mainpage.xaml

```
<SplitView x:Name="MySplitView" DisplayMode="CompactOverlay"  IsPaneOpen="{Binding IsPaneOpen, Mode=TwoWay}"
           CompactPaneLength="50" OpenPaneLength="150">
    <SplitView.Pane>
        <StackPanel Background="Gray"...>
    </SplitView.Pane>
    <SplitView.Content>
        <Grid>
            <Frame Content="{Binding CurrentPage}" />
        </Grid>
    </SplitView.Content>
</SplitView>
```

- De code om de buttons te plaatsen kan je op LEHO vinden
- Voorzie een ApplicationViewModel in de ViewModels map waar je volgende zaken voorziet
    - Constructor waarin je de huidige pagina plaatst, dit zal de AllWishesPage worden
    - CurrentPage Property
    - IsPaneOpen Property
    - Navigate methode
    - ShowMenuCommand
- Zorg dat deze ViewModel gekoppeld is aan je MainPage.xaml.
    - Zorg ervoor dat een instantie van dit ViewModel in de IocContainer gemaakt wordt
    - Voorzie een property naar dit ViewModel

```
0 references
static IoCContainer() {
   // ... code ...
    SimpleIoc.Default.Register<ApplicationViewModel>(false);
    // ... code ...
}

1 reference
public ApplicationViewModel ApplicationViewModel
{
    get { return SimpleIoc.Default.GetInstance<ApplicationViewModel>(); }
}
```

- Verbind de IsPaneOpen property van de SplitView met de property van de ApplicationViewModel

- Zorg ervoor dat wanneer de "hamburger button" gebruikt wordt, de pane open en toe kan gaan. Hiervoor verbind je deze button met de ShowMenuCommand.

AllWishes Page

- Maak een AllWishesPage aan in de Pages folder
  - Listbox
  - Velden om een nieuwe Wish aan te maken
  - Button om de nieuwe Wish op te slaan (gekoppeld aan het AddCommand)
- Zorg dat deze getoond wordt wanneer de applicatie start
- Voorzie de bijhorende ViewModel met volgende zaken
  - Wishes Property
  - GetAllWishes methode die de wishes list gaat opvullen
  - AddCommand om een nieuwe wish toe te voegen
  - DetailsCommand om de details van een wish op te vragen
- Instantieer het model en de page in de IoCContainer en voorzie een Properties naar deze instanties
- Voorzie de nodige xaml, styles en datatemplates om hiervoor een geschikte UI te maken

DetailsCommand

Wanneer men een wish in de listbox selecteert, laad je een nieuwe pagina in. Dit is de WishDetailsPage.

- Open blend en voorzie een command op de listbox (die afgevuurd wordt bij het SelectionChangedEvent)
- Koppel dit event aan de DetailsCommand beschikbaar in je AllWishesViewModel
  - Via dit event kan je navigeren naar een nieuwe Page

```
0 references
private void executeDetailsCommand()
{
    IoCContainer.IoC.EditWishViewModel.Wish = SelectedWish;
    IoCContainer.IoC.ApplicationViewModel.Navigate(typeof(Pages.WishDetailsPage));
}
```

- Maak een WishDetailsPage aan met bijhorende ViewModel
  - Voorzie een Wish property waarin je de geselecteerde Wish kan plaatsen. Deze wish wordt ingevuld voor het navigeren (zie bovenstaande code)
- Instantieer en voorzie properties in de IoCContainer

Resources

IoC

https://msdn.microsoft.com/en-us/library/ff921087.aspx

MVVM

https://channel9.msdn.com/Events/TechEd/NorthAmerica/2012/DEV216