Implantation IESN

Environnement de développement de logiciels

IG3 — Labo 5 — 2016-2017

Objectifs

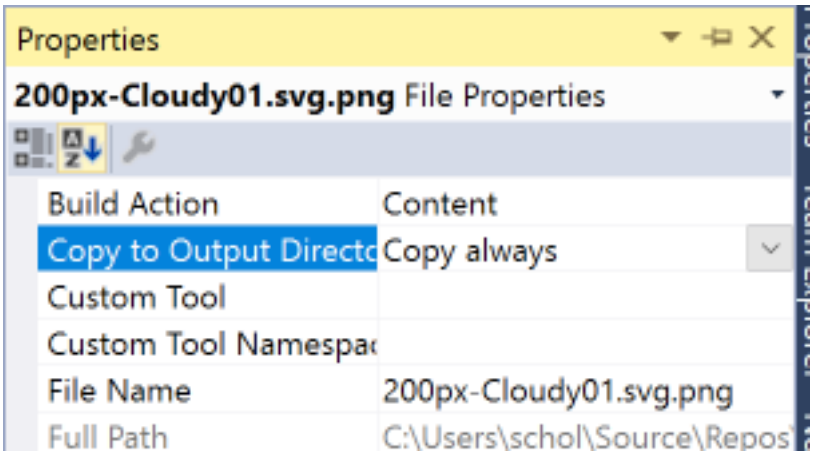
- Découpe en couches MVVM
- Recherche d'informations via un fichier Json

Description :

Vous allez dans le cadre de cet exercice réaliser des appels http à une API distante, à laquelle vous accèderez par Internet en http. Les données récupérées seront au format Json.
Le service en question retourne les prévisions météo pour une ville donnée en entrée.
Vous devrez formater les données récupérées et les afficher de manière user-friendly.

1. Afin d'utiliser le service de prévisions météo, il faut créer un compte qui nous donnera accès à l'API.
    a. Dans votre navigateur, ouvrez http://openweathermap.org/api. Créez votre compte (gratuit), puis dans votre profil, affichez la liste des API Keys. Vous devriez en avoir une par défaut. Recopiez sa valeur.
    b. Rendez-vous ensuite à nouveau sur la page http://openweathermap.org/api, sélectionnez « 5 day / 3 hour forecast » => api doc. Cherchez la documentation nécessaire pour réaliser un appel par le nom de la ville. Cherchez également comment récupérer les informations de manière localisée (en français, voir support multilingue) Il s'agit d'une URL qui contiendra les paramètres utiles. Recopiez cette URL, puis dans un autre onglet de votre navigateur, testez-la. A noter : vous devez utiliser votre API Key dans l'URL en question. Voir http://openweathermap.org/appid . Jouez avec les valeurs des paramètres afin de récupérer la météo de Namur.
2. Vous avez récupéré les prévisions météo sous forme de données structurées depuis une API distante à l'aide d'un client http (votre navigateur). Nous allons réaliser le même appel à l'aide d'un autre client http, intégré à une application UWP.

1. Créez une application universelle.
2. Allez sur Internet et recherchez deux images représentant une météo ensoleillée et une météo nuageuse. Ensuite, dans votre solution Visual Studio, créez un dossier Images dans lequel vous insérez ces dernières.
   Attention :



Ces images seront utilisées pour illustrer les prévisions météo.

3. Créez un dossier Model.
   a. Dans ce dernier, ajoutez une classe WeatherForecast.cs. Ce sera la classe qui donne le moule des données.
   Elle comprend plusieurs propriétés auto-implémentées : Date de type DateTime, MaxTemp, MinTemp et WindSpeed de type double et une chaîne de caractères WeatherDescription.
   b. Dans ce dossier, ajoutez une classe Forecast.cs.
   Elle comprend deux propriétés : Cityname de type string et une collection IEnumerable d'objets de type WeatherForecast, appelée WeatherForecasts.
   c. Toujours dans ce dossier[1], créez une classe WeatherService.cs. Cette dernière a pour but de rapatrier les données du service distant pour la ville de Namur. Elle comprendra donc la méthode indiquée plus loin.
   d. recopiez le code qui suit dans votre classe et modifiez le nom de la ville. Analysons le code de plus près :
      i. Un objet de type HttpClient (System.Net) est créé
      ii. A partir de cet objet, on adresse une requête (GET request) pour recevoir sous forme de chaîne (JSon) de caractères les données.
      On y a placé le mot await car le rapatriement des données demande du temps ; la méthode est alors déclarée async(hrone). Ce qui crée un thread secondaire. La méthode GetForecast ne bloque pas ainsi le thread principal, c'est-à-dire l'interface utilisateur dans notre cas[2].
      iii. Une fois les résultats obtenus, il faut les parser et en extraire des éléments en fonction des balises. Cette étape est réalisée en utilisant la librairie Newtonsoft.Json.

[1] Ou dans un autre dossier appelé DataAccessObject (DAO)
[2] Cf. module asynchrone

Téléchargez le package nuget de cette librairie et ajoutez-le à votre projet.

```
1 reference
public async Task<IEnumerable<WeatherForecast>> GetForecast()
{
    var wc = new HttpClient();
    var weather = await wc.GetStringAsync(new Uri("http://api.openweathermap.org/data
    var rawWeather = JObject.Parse(weather);
    var forecast = rawWeather["list"].Children().Select(d => new WeatherForecast()
        {
            Date = d["dt_txt"].Value<DateTime>(),
            MinTemp = d["main"]["temp_min"].Value<double>(),
            MaxTemp = d["main"]["temp_max"].Value<double>(),
            WeatherDescription = d["weather"].First["description"].Value<string>(),
            WindSpeed = d["wind"]["speed"].Value<double>()
        });
    return forecast;
}
```

iv. Attardez-vous à bien comprendre chacune des instructions ci-dessus. Comparez avec le retour obtenu lorsque vous avez réalisé le premier appel de test depuis le navigateur. Essayez de comprendre les classes et méthodes utilisées. Pour plus d'informations, voir http://www.newtonsoft.com/json/help/html/Introduction.htm

4. Dans un autre dossier Converter, écrivons deux classes de type IValueConverter : elles ont pour but de transformer des données rapatriées et de les préparer pour la vue.

   a. Nous voulons présenter la date sous une autre forme que celle rapatriée du fichier sous forme *DateTime*.
   Ajoutons une classe *StringFormatConverter* qui implémente l'interface IvalueConverter.

```
//Repris de http://blogs.msdn.com/b/africaapps/archive/2013/06/05/windows-8-getting-stringformat-functionality.aspx
4 references
public class StringFormatConverter : IValueConverter
{
    1 reference
    public object Convert(object value, Type targetType, object parameter, string language)
    {
        //No value provided
        if (value == null)
        {
            return null;
        }
        // No format provided.
        if (parameter == null)
        {
            return value;
        }

        return String.Format((String)parameter, value);
    }

    1 reference
    public object ConvertBack(object value, Type targetType, object parameter, string language)
    {
        return value;
    }
}
```

   b. La description du temps sera transformée en une image « nuageux » ou « ensoleillé ».
   Ajoutons une classe *WeatherDescriptionToImageValueConverter* qui implémente aussi l'interface IValueConverter.

```
4 references
public class WeatherDescriptionToImageValueConverter:IValueConverter
{
    1 reference
    public object Convert(object value, Type targetType, object parameter, string language)
    {
        var forecast = (string)value;
        if (forecast.Contains("nuageux"))
            return new BitmapImage(new Uri("ms-appx:/Images/cloudy.png"));
        else
            return new BitmapImage(new Uri("ms-appx:/Images/sunny.png"));
    }

    1 reference
    public object ConvertBack(object value, Type targetType, object parameter, string language)
    {
        throw new NotImplementedException();
    }
}
```

c. Nous associons ensuite deux clés à ces deux classes dans *App.xaml*. On ajoute deux ressources. Voyez l'exercice précédent pour un autre exemple d'utilisation des ressources. Vous devez comprendre leur utilité.

```
<Application.Resources>

    <local:WeatherDescriptionToImageValueConverter x:Key="WeatherImage"></local:WeatherDescriptionToImageValueConverter>
    <local:StringFormatConverter x:Key="StringConverter"></local:StringFormatConverter>
```

d. Créons un *DataTemplate* pour la vue.
Plaçons-nous dans App.xaml et créons une ressource *WeatherTemplate1.*

```
<DataTemplate x:Key="WeatherTemplate1">
    <Grid Margin="15">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"></ColumnDefinition>
            <ColumnDefinition Width="*"></ColumnDefinition>
        </Grid.ColumnDefinitions>

        <Image Width="50" Height="50" Source="{Binding WeatherDescription, Converter={StaticResource WeatherImage}}"></Image>
        <StackPanel Grid.Column="1" Margin="15,0,0,0" Width="200">
            <TextBlock HorizontalAlignment="Left" TextWrapping="Wrap" Text="{Binding Date, Converter={StaticResource StringConverter}, ConverterParameter='{}{0:dd MMM yyyy}'}" VerticalAli
            <TextBlock HorizontalAlignment="Left" TextWrapping="Wrap" Text="{Binding WeatherDescription}" VerticalAlignment="Top"/>
            <TextBlock HorizontalAlignment="Left" TextWrapping="Wrap" VerticalAlignment="Top">
            <Run Text="Min "></Run>
            <Run Text="{Binding MinTemp}"></Run>
            <Run Text="°"></Run>
            <Run Text="Max "></Run>
            <Run Text="{Binding MaxTemp}"></Run>
            <Run Text="°"></Run>
            </TextBlock>
        </StackPanel>
    </Grid>
</DataTemplate>
```

e. Comme dans le cours MVVM, recherchons dans NuGet les librairies MVVMLight.
Créons le dossier *ViewModel* qui comprendra *ViewModelLocator.cs* et *MainViewModel.cs*.
Une ressource supplémentaire doit être ajoutée dans *App.xaml* ; elle associe une clé *Locator* à la classe ViewModelLocator.

```
<Application.Resources>
    <vm:ViewModelLocator x:Key="Locator" xmlns:vm="using:FlickClient.ViewModel" />
```

f. Préparons la vue. Adaptons le code xaml dans *MainPage.xaml*. La vue est liée au *ViewModel* par le *DataContext*. La création d'un objet de type *ViewModel* est effectué par Main qui se trouve dans le *ViewModelLocator*.

```
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    DataContext="{Binding Source={StaticResource Locator}, Path=Main}"
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

    <Grid>
        <ListView ItemsSource="{Binding Forecast}" ItemTemplate="{StaticResource WeatherTemplate1}"/>
    </Grid>
</Page>
```

g. Dans la classe MainViewModel, nous devons trouver la propriété *Forecast*.
Dès lors, ajoutons ce qu'il faut dans *MainViewModel.cs*.
A savoir l'attribut *_forecast*, sa propriété *Forecast* où il faut signaler de prendre en compte le changement et une méthode *InitialiazeAsync* qui va recharger les données de la couche *Model*.

```
public class MainViewModel : ViewModelBase
{

    private ObservableCollection<WeatherForecast> _forecast = null;

    2 references
    public ObservableCollection<WeatherForecast> Forecast
    {
        get
        {
            return _forecast;
        }

        set
        {
            if (_forecast == value)
            {
                return;
            }
            _forecast = value;
            RaisePropertyChanged("Forecast");
        }
    }
```

```
public MainViewModel()
{
    if (IsInDesignMode)
    {
        var forecast = new Forecast() { CityName = "Namur" };
        var weatherForecasts = new List<WeatherForecast>();
        for (int i = 0; i < 40; i++)
        {
            weatherForecasts.Add(new WeatherForecast()
                {
                    Date = DateTime.Now.AddDays(i),
                    MaxTemp = 5 + i / 2,
                    MinTemp = 0 + i / 2,
                    WeatherDescription = "Peu de nuages",
                    WindSpeed = i % 7
                });
        }
        forecast.WeatherForecasts = weatherForecasts;
        Forecast = new ObservableCollection<WeatherForecast>(weatherForecasts);
    }
    else
    {
        InitializeAsync();
    }
}

1 reference
public async Task InitializeAsync()
{
    var service = new WeatherService();
    var forecast = await service.GetForecast();
    Forecast = new ObservableCollection<WeatherForecast>(forecast);
}
```

10. Lancez votre application. Votre affichage devrait être similaire au suivant, à deux nuances près :

- l'illustration ci-dessous affiche les heures de chaque prévision. Adaptez votre programme pour en faire de même.
- Les images ne sont pas nécessairement identiques aux vôtres.

09 Oct 2016 21:00
peu nuageux
Min 277.789 ° Max 277.79 °

10 Oct 2016 00:00
peu nuageux
Min 274.828 ° Max 274.83 °

10 Oct 2016 03:00
ensoleillé
Min 273.41 ° Max 273.414 °

10 Oct 2016 06:00
peu nuageux
Min 272.34 ° Max 272.341 °

10 Oct 2016 09:00
partiellement ensoleillé
Min 280.782 ° Max 280.782 °

10 Oct 2016 12:00
nuageux
Min 284.515 ° Max 284.515 °

Aller plus loin

Vous avez terminé ? Essayez l'exercice suivant.

Pour l'instant, la ville est hardcodée et c'est son nom qui est utilisé comme critère de recherche. OpenWeatherMap (OWM) recommande d'utiliser les coordonnées ou l'identifiant de la ville pour plus d'assurance dans les réponses reçues. Permettez à l'utilisateur de rechercher une ville reconnue par l'API OWM, mémorisez son id et modifiez votre URL pour récupérer les prévisions sur base de cet identifiant. Mémorisez la ville saisie par l'utilisateur. De sorte que ce dernier ne devra pas la saisir à nouveau lorsque l'application sera relancée.

Encore plus loin ? Permettre à l'utilisateur de spécifier un ensemble de villes pour lesquelles il doit pouvoir visualiser les prévisions météo.

Vous aurez donc besoin de plusieurs vues:

- recherche et ajout d'une ville, gestion de celles-ci (avec suppression)
- page d'accueil : affiche la prévision pour le lendemain pour chacune des villes choisies
- page de détail d'une ville : affiche pour une ville choisie, les prévisions disponibles pour cette ville (et éventuellement d'autres informations, reçues depuis d'autres sources de données par exemple)

Vous aurez besoin de maîtriser plusieurs aspects :

- Stockage local/roaming des villes choisies
- Navigation entre pages, passage de paramètres entre pages dans un contexte MVVM