XAML ALAPÚ FEJLESZTÉS – MVVM MINTA HASZNÁLATA

Segédlet a Kliens oldali technológiák c. tárgyhoz

Tóth Tibor
toth.tibor@aut.bme.hu
2020

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

1117 Budapest, Magyar tudósok körútja 2.
Q. ép. B. szárny 207. • www.aut.bme.hu
Telefon: 463-2870 • Fax: 463-2871

Bevezetés

Célkitűzés

A labor során megismerjük a kliens oldali MVVM minta alkalmazását az UWP XAML platformján szemléltetve.

Előfeltételek

A labor elvégzéséhez szükséges eszközök:

- Microsoft Visual Studio 2017
  - Universal Windows Platform min v14393 SDK
- Kiinduló project (`Cookbook_start.zip`)

Amit érdemes átnézned

- XAML nyelv alapjai, adatkötés (Kliens oldali technológiák jegyzet, előző labor)
- MVVM minta XAML kontextusban (Kliens oldali technológiák jegyzet)

Lebonyolítás

A gyakorlat anyagát távolléti oktatás esetén önállóan, jelenléti oktatás esetében számítógépes laborban, gyakorlatvezetői útmutatással kell megoldani.

Beadás

Amennyiben elkészültél a megoldással, távolíts el belőle a fordítási binárisokat és fordítási segédmappákat („bin” mappa, „obj” mappa, „Visual Studio” mappa, esetleges nuget/node csomagok, majd az így elkészült anyagot egy zip fájlba becsomagolva töltsd fel a Moodle rendszerbe. Amennyiben a zip fájl mérete 10 MB fölött van, valószínűleg nem töröltél ki belőle mindent a korábbi felsorolásból. Jegyzőkönyv készítése nem szükséges, azonban amennyiben beadott kódoddal kapcsolatban kérdések merülnek fel, megkérhetünk arra, hogy működésének egyes részleteit utólag is magyarázd el.

Összefoglalás

A labor során egy receptkönyv alkalmazás néhány funkcióját valósítjuk meg MVVM minta mentén. A minta egyszerűbb megvalósításához egy MVVM keretrendszert használunk a Template10-t. A receptek adatai egy webszolgáltatáson keresztül érhetőek el, ami HTTP kéréseken keresztül JSON formátumban szolgáltatja nekünk az adatokat.

Áttekintés

Template10

A tanszéki portálról letöltött kiinduló projectünk a Template10 keretrendszerre épül, ami többek között az MVVM funkciók támogatására is ad előre elkészített megoldásokat. A Template10 mellett több más keretrendszert is használhattunk volna pl.: Prism, MVVMLight, Caliburn.Micro, ReactiveUI.

A kiinduló projektben mappákba rendszerezve találjuk a különböző komponenseket:

- `Views`: XAML oldalak nézetek
  - `MainPage.xaml`: A receptkönyv alkalmazásunk főoldala, ami most tartalmazza a főoldal nézetének főbb elemét a GridView-t
- `ViewModels`: View Model osztályok
  - `MainPageViewModel`: Üres osztály, csak ránk vár.
    - `ViewModelBase`-ből származik, ami megvalósítja az `INotifyPropertyChanged` interfészt és támogatást ad a navigációs életciklusesemények lekezelésére, és a navigációra. (`OnNavigatedToAsync`, `NavigationService`, stb.)
    - Az adatkötés működése miatt a `MainPage` oldal `DataContext` tulajdonságának egy `MainPageViewModel` példány van megadva a xaml-ben.
- `Models`: üres mappa ide kerülnek majd az adatosztályaink
- `Services`: üres mappa ide kerülnek majd a szolgáltatásosztályaink

Szolgáltatás

Az alkalmazásunk elkészítése során egy REST-es webszolgáltatást szeretnénk használni, amit a http://bmecookbook.azurewebsites.net oldalon érhetünk el. Itt Az API menüpontot választva áttekinthetjük a webszolgáltatás interfészét. Esetünkben 3 kérés lesz érdekes:

- Főoldalon a `GET api/Recipes/Groups` kérés eredményét akarjuk megjeleníteni
- Recept részletes oldalon a `GET api/Recipes/{id}` kéréssel a recept részletes adatait kérdezzük le
- Illetve a recept részletes oldalon új kommentet hozunk létre a `POST api/Recipes/{id}/Comments` kéréssel.

Főoldal (1. feladat)

Modell és szolgáltatás

Készítsük el az első kérést a főoldal számára. Tekintsük át milyen adatstruktúrával tér vissza a `GET api/Recipes/Groups` kérés. Figyeljük meg, hogy ez egy `RecipeGroup` listával tér vissza, egy `ReceptGroup` pedig többek között tartalmazza a hozzá tartozó `RecipeHeader` listát.

1. A `Models` mappába hozzunk létre egy osztályt `RecipeGroup` néven.
2. Töröljük ki a generált osztályt, az üres névtér maradjon meg.

3. A `GET api/Recipes/Groups` kérés dokumentációjának példa JSON-éből másoljunk ki a vágólapra pontosan egy `RecipeGroup` objektumot (első kapcsos zárójeltől a lezáróig, a vessző már nincs benne)

```
{
    "Id": "sample string 1",
    "Title": "sample string 2",
    "Recipes": [
      {
        "Id": 1,
        "Title": "sample string 2",
        "BackgroundImage": "sample string 3",
        "TileImage": "sample string 4"
      },
      {
        "Id": 1,
        "Title": "sample string 2",
        "BackgroundImage": "sample string 3",
        "TileImage": "sample string 4"
      }
   ]
}
```

4. Ezt a `RecipeGroup.cs` fájlba az Edit / Paste special / Paste JSON as Classes menüponttal illesszük be.
5. Javítsuk ki az elnevezéseket és a típusokat:
   a. `RootObject` ➔ `RecipeGroup`
   b. `Recipe` ➔ `RecipeHeader`
   c. `RecipeHeader[]` ➔ `List<RecipeHeader>`
6. Vegyünk fel egy szolgáltatás osztályt a `Services` mappába `CookbookService` néven
7. Készítsünk egy olyan generikus metódust, ami a `System.Net.HttpClient` osztály segítségével végrehajt egy GET kérést aszinkron módon, majd az eredményt a kapott JSON-ból visszasorosítja.

Megj.: A JSON sorosításhoz hozzá kellene adni a projekthez a NewtonSoft.Json (Json.net) NuGet csomagot, de esetünkben a Template10 már behivatkozta azt.

```
private async Task<T> GetAsync<T>(Uri uri)
{
    using (var client = new HttpClient())
    {
        var response = await client.GetAsync(uri);
        var json = await response.Content.ReadAsStringAsync();
        T result = JsonConvert.DeserializeObject<T>(json);
        return result;
    }
}
```

Az `async / await` egy C# nyelvi elem az aszinkron, háttérszálon történő utasítás végrehajtás támogatására, hogy ne a UI szálat terheljük hosszan tartó műveletekkel (ne fagyjon be az alkalmazás). `async / await`-et az alábbi megkötések mentén lehet használni:

- `await`-et hívni (bevárni egy háttérben futó műveletet) `Task` vagy `Task<T>` objektumon lehet.
- `await`-et hívni csak `async` metódusban lehet
- `async` metódus csak `void`, `Task` vagy `Task<T>` (vagy Task szerű objektum – itt nem fejtjük ki) visszatérési típusú lehet.
    - `void` esetében nem lehet kívülről bevárni
    - `async Task, async Task<T>` metódus esetében a `return` utasítással nem `Task`-ot kell visszaadjunk, hanem a taszk eredményét (semmit vagy `T`-t).

8. Készítsünk olyan metódust, ami lekéri az összes recept csoportot.

```
private readonly Uri serverUrl = new Uri("https://bmecookbook.azurewebsites.net");

public async Task<List<RecipeGroup>> GetRecipeGroupsAsync()
{
    return await GetAsync<List<RecipeGroup>>(new Uri(serverUrl, "api/Recipes/Groups"));
}
```

ViewModel

A `MainPage`-hez tartozó ViewModel-ben definiáljuk felül az `OnNavigatedToAsync` metódust és kérjük le a recept csoportokat, egy `ObservableCollection`-be.

9. Hozzuk létre a kollekciót `RecipeGroups` néven:

```
public ObservableCollection<RecipeGroup> RecipeGroups { get; set; } =
    new ObservableCollection<RecipeGroup>();
```

10. Az `OnNavigatedToAsync` függvényben kérjük le a `CookbookService`-en keresztül a recept csoportokat, és az eredményt adjuk hozzá a kollekciónkhoz.

```
public override async Task OnNavigatedToAsync(
    object parameter, NavigationMode mode, IDictionary<string, object> state)
{
    var service = new CookbookService();
    var recipeGroups = await service.GetRecipeGroupsAsync();
    foreach (var item in recipeGroups)
    {
        RecipeGroups.Add(item);
    }

    await base.OnNavigatedToAsync(parameter, mode, state);
}
```

Mivel itt is `await`-et hívni csak `async` metódusban lehet vegyük fel az `async` kulcsszót a metódus fejlécébe. Viszont ilyenkor a `return base` hívás már nem jó szintaktika, változtassuk meg azt `await`-re.

Nézet

A `MainPage.xaml` oldalon szerencsére már elkészítették nekünk a nézetet, a mi feladatunk ezt adatokkal feltölteni adatkötésen keresztül a View Modelből. Figyeljük meg, hogy a felületet készítő

fejlesztő csoportosított `GridView`-t használt, amit ha adatokkal szeretnénk feltölteni akkor először egy `CollectionViewSource` objektumon keresztül csoportosítanunk kell a kollekciónkat.

11. Hozzunk létre az oldal erőforrásai között egy `CollectionViewSource`-t aminek a forrás adatait kössük a `RecipeGroups` tulajdonságra és kapcsoljuk be a csoportosítást a `RecipeGroup`-on belül található `Recipes` kollekcióra.

```
<Page.Resources>
    <CollectionViewSource x:Key="RecipesViewSource"
                          IsSourceGrouped="True"
                          Source="{Binding RecipeGroups}"
                          ItemsPath="Recipes" />
</Page.Resources>
```

12. Vegyük fel az adatkötéseket az oldalon

```
<GridView Grid.Row="1"
          Padding="18"
          ItemsSource="{Binding Source={StaticResource RecipesViewSource}}"
          IsItemClickEnabled="True"
          ItemClick="RecipeGroups_OnItemClick">
    <GridView.ItemTemplate>
        <DataTemplate>
            <Grid Width="250" Height="250">
                <Image Source="{Binding BackgroundImage}"
                       Stretch="UniformToFill" VerticalAlignment="Center" />
                <Border VerticalAlignment="Bottom" Background="#AA000000">
                    <TextBlock Text="{Binding Title}" Margin="12" Foreground="White"/>
                </Border>
            </Grid>
        </DataTemplate>
    </GridView.ItemTemplate>
    <GridView.GroupStyle>
        <GroupStyle>
            <GroupStyle.HeaderTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding Title}" Margin="-12,0,0,0"/>
                </DataTemplate>
            </GroupStyle.HeaderTemplate>
        </GroupStyle>
    </GridView.GroupStyle>
</GridView>
```

Próbáljuk ki!

Részletes oldal (2. feladat)

Egy receptre kattintva navigáljunk el a recept részletes oldalára, ahol a kommentek funkciót fogjuk csak megvalósítani.

Inicializáció

13. Hozzunk létre a `Views` mappába egy új Blank Page-et `DetailsPage` néven.

14. Hozzuk létre a hozzá tartozó View Model osztályt a `ViewModels` mappába `DetailsPageViewModel` néven. Ez legyen publikus és származzon a `ViewModelBase` absztrakt osztályból.

```
public class DetailsPageViewModel : ViewModelBase
{
}
```

15. Állítsuk be a `DetailsPage.xaml`-ben az oldal `DataContext`-ének egy új `DetailsPageViewModel` példányt az adatkötés számára.

```
xmlns:vm="using:Cookbook.ViewModels"

<Page.DataContext>
    <vm:DetailsPageViewModel />
</Page.DataContext>
```

Főoldalról történő navigáció

A főoldal code behind-jában van számunkra egy előre elkészített eseménykezelő, amit a `GridView`-n történő elemkattintás süt el. Ezt a funkciót a View Modelben implementáljuk le, amit most az eseménykezelő hív majd meg. A navigáció történjen a View Modelben lévő `NavigationService` objektum felhasználásával. A navigáció során a `DetailsPage`-nek adjuk át a kattintott recept azonosítóját, ami alapján már a részletes oldal le fogja tudni kérni az adatokat.

16. Navigáció a `MainPageViewModel`-ben:

```
public void NavigateToDetails(int recipeId)
{
    NavigationService.Navigate(typeof(DetailsPage), recipeId);
}
```

17. Eseménykezelő a `MainPage.xaml.cs`-ben: (Használjuk fel a xaml-ben létrehozott View Modelt, amit `x:Name` el van nevezve, hogy elérjük a code behindból is.)

```
private void RecipeGroups_OnItemClick(object sender, ItemClickEventArgs e)
{
    var recipeHeader = (RecipeHeader)e.ClickedItem;
    ViewModel.NavigateToDetails(recipeHeader.Id);
}
```

Megj.: Ez nem a legszebb módja az MVVM eseménykezelésnek, mert szeretjük a xaml.cs állományt tisztán tartani. Ez áthidalható lenne a Behaviors SDK használatával. Most a laboron erre nem térünk ki külön.

Próbáljuk ki a navigációt!

Modell és szolgáltatás

Készítsük el a modellosztályokat a recept részletes oldal számára. Itt a `GET api/Recipes/{id}` kérés az érdekes, ahol az `{id}` helyére a lekérendő recept azonosítóját kell megadnunk.

18. Hozzunk létre egy osztályt a `Models` mappába `Recipe` néven.
19. Illesszük be a teljes példa json-t C# osztályként

20. Javítsuk át az osztály nevét: `RootObject` ➔ `Recipe` (most ne törődjünk a tömb vs lista problémával)
21. Illesszük be a recept részletes adatit letöltő metódust a szolgáltatásba (használjuk a **bit.ly/klienslabor3** gistet):

```
public async Task<Recipe> GetRecipeAsync(int id)
{
    return await GetAsync<Recipe>(new Uri(serverUrl, $"api/Recipes/{id}"));
}
```

bit.ly/klienslabor3

Részletek letöltése a View Modelben

22. Hozzunk létre egy változásértesítést is támogató tulajdonságot, ami a recept részletes adatait fogja eltárolni.

*Tipp: használjuk **a propfull** snippetet és egészítsük ki a settert a Set hívással*

```
private Recipe _recipe;
public Recipe Recipe
{
    get { return _recipe; }
    set { Set(ref _recipe, value); }
}
```

Itt az ősből használt `Set` metódus süti el az előző órán megismert `INotifyPropertyChanged` interfészben lévő `PropertyChanged` eseményt, így megvalósítva a változásértesítést az adatkötés felé.

23. Az `OnNavigatedToAsync`-ot definiáljuk felül és kérjük le a recept részletes adatait. Emlékezzünk vissza, hogy a navigáció során a recept azonosítóját adtuk át a részletes oldalnak.

```
public override async Task OnNavigatedToAsync(
    object parameter, NavigationMode mode, IDictionary<string, object> state)
{
    var recipeId = (int)parameter;

    var service = new CookbookService();
    Recipe = await service.GetRecipeAsync(recipeId);

    await base.OnNavigatedToAsync(parameter, mode, state);
}
```

Nézet és adatkötés

24. Másoljuk be a `DetailsPage.xaml` oldal előre elkészített tartalmát. Vegyük fel benne az adatkötéseket. **bit.ly/klienslabor3**

```
xmlns:c="using:Template10.Controls"

<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
```

```
    </Grid.RowDefinitions>

    <c:PageHeader Grid.Row="0" Content="{Binding Recipe.Title}" />
    <StackPanel Grid.Row="1" Margin="18" Orientation="Vertical" MaxWidth="400">
        <TextBlock Style="{StaticResource SubheaderTextBlockStyle}"
                   Text="Comments" Margin="0,0,0,12" />

        <TextBox PlaceholderText="Name" Margin="0,0,0,10" />
        <TextBox PlaceholderText="Comment" Margin="0,0,0,10" />

        <Button Content="Send" Margin="0,0,0,10" />

        <ItemsControl ItemsSource="{Binding Recipe.Comments}">
            <ItemsControl.ItemTemplate>
                <DataTemplate>
                    <StackPanel Orientation="Vertical">
                        <TextBlock FontWeight="Bold" TextWrapping="Wrap"
                                   Text="{Binding Name}" />
                        <TextBlock Text="{Binding Text}" />
                    </StackPanel>
                </DataTemplate>
            </ItemsControl.ItemTemplate>
        </ItemsControl>
    </StackPanel>
</Grid>
```

bit.ly/klienslabor3

Próbáljuk ki!

KOMMENTELÉS FUNKCIÓ, COMMAND

Valósítsuk meg a kommentelési funkciót a View Modelben. Ehhez szükségünk van két változásértesítést is támogató tulajdonságra a VM-ben, amit kétirányú adatkötéssel a `TextBox`-ok `Text` tulajdonságához kötünk majd.

25. `Name`, `Text` tulajdonságok a `DetailsPageViewModel`-ben: **bit.ly/klienslabor3**

```
private string _newComment;
public string NewComment
{
    get { return _newComment; }
    set { Set(ref _newComment, value); }
}

private string _name;
public string Name
{
    get { return _name; }
    set { Set(ref _name, value); }
}
```

bit.ly/klienslabor3

26. Adatkötések a `DetailsPage.xaml`-ben:

```
<TextBox PlaceholderText="Name" Margin="0,0,0,10"
         Text="{Binding Name, Mode=TwoWay}" />
<TextBox PlaceholderText="Comment" Margin="0,0,0,10"
         Text="{Binding NewComment, Mode=TwoWay}" />
```

A küldés gomb kattintásának lekezelését ne eseménykezelővel, hanem a gomb által támogatott command minta segítségével.

27. Hozzunk létre egy publikus `DelegateCommand` típusú tulajdonságot a `DetailsPageViewModel`-ben, ami egy privát függvényt hívjon meg.

```
public DetailsPageViewModel()
{
    SendCommentCommand = new DelegateCommand(SendComment);
}

public DelegateCommand SendCommentCommand { get; }

private void SendComment()
{
}
```

28. Kössük be ezt a nézeten a gomb commandjára.

```
<Button Content="Send" Command="{Binding SendCommentCommand}" Margin="0,0,0,10" />
```

Próbáljuk ki, hogy meghívódik-e a command, és helyes értékek vannak-e a tulajdonságokban.

Megj.: Itt lehetőségünk lenne CommandParametert is megadni, olyankor a DelegateCommand<T> osztályt érdemes használni, ami egy void f(T param) metódust tud meghívni.
Megj2.: A DelegateCommand a command végrehajthatóságára vonatkozóan is tartalmaz információt, aminek a újbóli kiértékelését nekünk kell kezdeményezni a RaiseCanExecuteChanged metódussal.

Önálló feladatok (valószínűleg már nem lesz rá idő)

Komment elküldése (3. feladat)

29. Implementáld a `SendComment` függvényt.
    a. Használd a `POST api/Recipes/{id}/Comments` hívást, amit a szolgáltatásban implementálj! Figyelj arra, hogy ez egy POST metódus, és adatot kell felküldened.
    b. Frissítsd a recept részletes adatait miután elküldted a kommentet!

Send gomb engedélyezése (4. feladat)

30. A `Send` gomb csak akkor lehessen megnyomható, ha a `Name` és a `NewComment` tulajdonságok nem üresek!
    a. `DelegateCommand` felparaméterezése.
    b. `DelegateCommand` állapotának frissítése, ha megváltoztak a végrehajthatóság feltételei.
    c. *(szorgalmi)* Ne csak fókuszvesztésre frissüljön a command állapota, hanem már billentyűleütésre is.

31. Küldés után töröljük a beviteli mezők tartalmát!