Eseményvezérelt alkalmazások: 11. gyakorlat

A feladat az előző, *10. gyakorlaton* elkezdett, több platformon is futtatható **aszinkron kép letöltő alkalmazás bővítése**, amely segítségével már ne csak listázhassuk az egy weboldalon megjelenő képeket, de azokat egy külön ablakban megnyitva nagyobb méretben tekinthessük meg és tölthessük le!

Az alkalmazást *Avalonia UI* keretrendszerrel, háromrétegű (modell-nézet-nézetmodell) architektúrában, eseményvezérelt paradigma alapján valósítjuk meg.

1 Nézetmodell *EM*

Figure 1: `ImageViewModel` osztálydiagramja

A *ViewModels* mappában hozzunk létre egy `ImageViewModel` osztályt, amely leszármazik a `ViewModelBase` osztályból.

Az osztályba a 8. gyakorlaton elkészített módon vegyünk fel:

- egy `Image` nevű, `Bitmap` típusú tulajdonságot;
- egy `SaveImage` nevű, `EventHandler<Bitmap>` típusú eseménnyel; és
- egy `SaveImageCommand` parancssal.

Egészítsük ki továbbá:

- egy `Close` nevű eseménnyel; és
- egy `CloseCommand` paranccsal.

Hozzuk létre az osztály konstruktorát, amely inicializálja az `Image` tulajdonságot (paraméterként várja a konstruktor), és két parancsot (olyan módon, hogy mindkettő a kapcsolódó eseményt váltsa ki).

2 Nézet *EM*

A *Views* mappára kattintva az egér jobb gombjával, az *Add New Item* menüben adjunk hozzá egy új *Avalonia UserControl* elemet. A neve legyen *ImageView*.

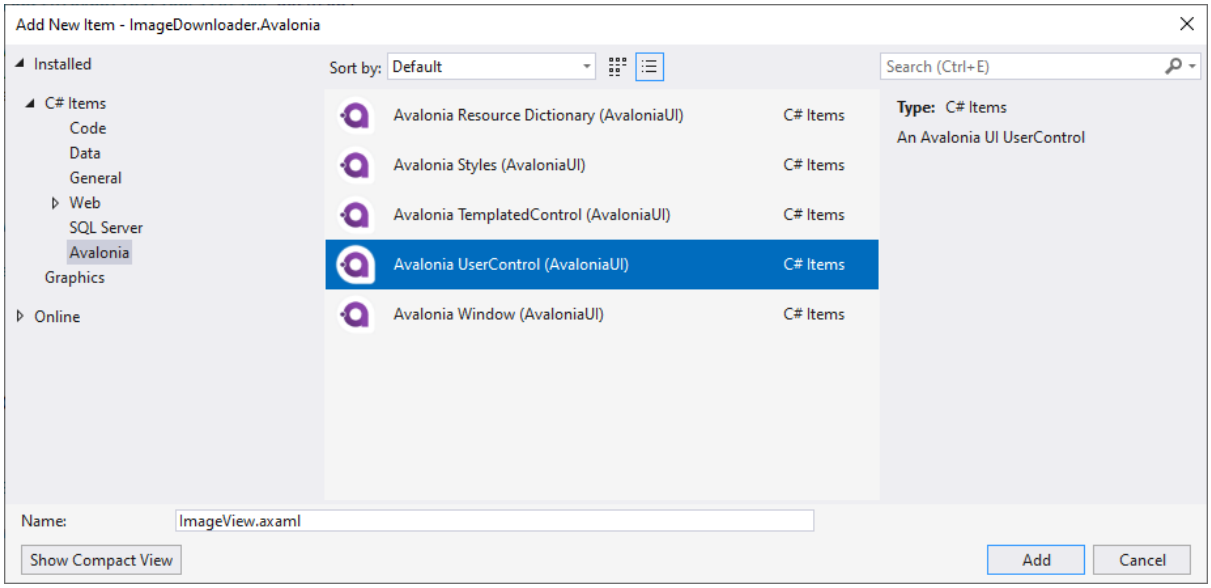

Figure 2: Új nézet hozzáadása a projekthez

Az `ImageView` képernyőt a következő módon alakítsuk ki:

- Méretét állítsuk *600 x 450* pixel-re.
- Elrendező vezérlőként adjunk hozzá egy `DockPanel`-t, abba pedig egy `Button` és egy `Image` vezérlőt. A gombot dockoljuk a nézet aljára.
- A felhelyezett képnek (`Image`) a `Source` tulajdonsága adatkötéssel a `ImageViewModel`-ben található `Image` tulajdonságra kössön.

  ```
  Source="{Binding Image}"
  ```

- A gombra azért van szükségünk, mert mobil platformon is vissza kell tudnunk lépni a fő képernyőre, és nem lesz az asztali ablakos alkalmazásokhoz hasonló módon bezárható. A gomb felirata ezért legyen *"Bezárás"*, és parancskötéssel a nézetmodell `CloseCommand` parancsához kössük.

A *Views* mappához adjunk hozzá egy új *Avalonia Window* elemet is *ImageWindow* névvel. Az ablakba ágyazzuk be az előbb létrehozott kompozit vezérlőt. Írjuk át az ablak címét (`Title` property) például a *"Képmegjelenítő"* kifejezésre.

2.1 Bezárás gomb kondícionális megjelenítése

A *Bezárás* gombra igazából csak mobil platform, azaz nem ablakos megjelenítés esetén van szükségünk. Az `OnPlatform` XAML markup kiterjesztéssel (*markup extension*) kondícionálisan, platformspecifikusan alakíthatjuk ki a felületet:

```
<Button ...
        IsVisible="{OnPlatform Android=True, Default=False}"
        />
```

2.2 `MainView` nézet refaktorálása

A `MainView` nézetben a képek megjelenítését ágyazzuk gombokba (`Button`), hogy a kattintásuk parancskötéssel kezelhető legyen.

A *8. gyakorlathoz* hasonlóan itt is vegyük észre, hogy ha megpróbáljuk kötni a `Button` vezérlő `Command`-jához az `ImageSelectedCommand`-ot, azt tapasztalhatjuk, hogy a kötés kialakítása sikertelen. Ennek oka, hogy a parancsot alapértelmezetten az éppen aktuális elemen keresi. Egy lehetséges megoldás, hogy a `UserControl` típusú ős (szülő) vezérlőt kikeressük, majd annak a `DataContext.ImageSelectedCommand` parancsára végezzük el a kötést:

```
Command="{Binding $parent[UserControl].DataContext.ImageSelectCommand}"
```

Összevetésként ez így nézett ki WPF-ben, ott még `Window` típusú őst keresve:

```
Command="{Binding DataContext.ImageSelectedCommand,
                  RelativeSource={RelativeSource AncestorType=Window}}"
```

3 Vezérlés *EM*

Egészítsük ki az `App.axaml.cs` fájlban az `App` osztályt az alábbiakkal:

1. A `OnFrameworkInitializationCompleted` metódusban a nézetmodell példányosítása után iratkozzunk fel a `MainViewModelhez` tartozó `ImageSelected` eseményre. Ehhez készítsünk egy eseménykezelő eljárást is.
2. Az eseménykezelő eljárásban példányosítsunk egy `ImageViewModel` nézetmodellt, majd:
   - Asztali alkalmazás (`IClassicDesktopStyleApplicationLifetime` életciklus) esetén készítsünk egy új `ImageWindow` ablakot. Az elkészített ablak `DataContext` paraméterének adjunk át egy újonnan példányosított `ImageViewModel`-t, majd jelenítsük meg az ablakot.
   - Mobil alkalmazás (`ISingleViewApplicationLifetime` életciklus) esetén készítsünk egy új `ImageView` képernyőt. (Megjeleníteni az `ApplicationLifetime.MainView` tulajdonságának értékül adásával lehet. Előtte azonban iratkozzunk fel a `Close` eseményére, amelynek eseménykezelőjében állítsuk vissza az eredeti, fő képernyőt olyan módon, hogy a `ApplicationLifetime.MainView` módosítása előtt annak korábbi értéket elmentjük egy segédváltozóba.
3. Adjunk egy `TopLevel` *property*-t az osztályhoz, ami a *TopLevel* lekérdezését platform-független módon lehetővé teszi. A *TopLevel* felel a felület megjelenítéséért és a szolgáltatások (mint például a dialógus ablakok) elérhetőségért az Avalonia UI keretrendszerben.

```
private TopLevel? TopLevel
{
    get
    {
        return ApplicationLifetime switch
        {
            IClassicDesktopStyleApplicationLifetime desktop =>
                TopLevel.GetTopLevel(desktop.MainWindow),
            ISingleViewApplicationLifetime singleViewPlatform =>
                TopLevel.GetTopLevel(singleViewPlatform.MainView),
            _ => null
        };
    }
}
```

   Megjegyzés: asztali alkalmazás esetén maga az ablak (jelen esetben a `desktop.MainWindow`) a *TopLevel* vezérlő, az egyszerűsítést az egységesség érdekében nem végeztük el.

4. Az `ImageSelected` eseménykezelő metódusban a nézetmodell példányosítása után iratkozzunk fel az `ImageModelhez` tartozó `SaveImage` eseményre. Ehhez is készítsünk egy eseménykezelő eljárást, ami a `TopLevel` használatával nyisson dialógus ablakot a kép mentéséhez:

```
IStorageFile? file = await TopLevel.StorageProvider.SaveFilePickerAsync(
    new FilePickerSaveOptions()
    {
        Title = "Save image",
        SuggestedFileName = "download.png",
        FileTypeChoices = new [] { FilePickerFileTypes.ImagePng }
    }
);
```

Az eredményként kapott `file` változó amennyiben nem `null`, akkor mentsük PNG-ként a képet a `file.OpenWriteAsync()` által megnyitható adatfolyamba. A kép mentése a 8. gyakorlaton a Windows specifikus `BitmapEncoder` segítségével történt, ezért most helyette használjuk a platformfüggetlen *ImageSharp* NuGet csomagot:

```
using var stream = new MemoryStream();
e.Save(stream); // írjuk adatfolyamba a bitmap-et
stream.Seek(0, SeekOrigin.Begin); // ugorjunk vissza az adatfolyam elejére

using var image = SixLabors.ImageSharp.Image.Load<Rgba32>(stream);
using var fileStream = await file.OpenWriteAsync();
await image.SaveAsync(fileStream, new PngEncoder());
// mentsük PNG-ként a képet az ImageSharp osztálykönyvtár használatával
```

4 MVVM Toolkit használata ^{EM}

Az előadáson bemutatott *MVVM Community Toolkit* használatával az Avalonia UI keretrendszerben készített alkalmazásaink nézetmodelljét gyorsabban és könnyebben elkészíthetjük, a kapott forráskód is rövidebb és áttekinthetőbb lesz. (WPF és Windows Form alkalmazások esetén is használható.)

Az MVVM Toolkit NuGet csomag formájában érhető el, azonban már a projekt létrehozásakor hozzáadásra került, mert ezt a tervezés mintát választottuk ki. Végezzük el a következő refaktorálásokat a program nézetmodell komponensén:

1. A `ViewModelBase` osztályunkat a `ObservableObject` típusból származtassuk és jelenlegi tartalmát törölhetjük, mert az meg is örökli ettől az ősosztálytól.
2. A `DelegateCommand` osztály töröljük, helyette a `RelayCommand` és a `RelayCommand<T>` osztályokat használjuk. A generikus változat tud parancs paramétert fogadni. A paraméter típusellenőrzésére itt már nincs szükség, mert nem csak `Object` vehető át.
3. A `MainViewModell` nézetmodellben az `IsDownloading` és a `Progress` *property*-ket töröljük, helyette lássuk el az `ObservableProperty` attribútummal az adattagokat (`_isDownloading`, `_progress`). A projekt újrafordítása után a *property*-k kódgenerálása megtörténik.
4. Az `IsDownloading` *property*-t lássuk el a `NotifyPropertyChangedFor(nameof(DownloadButtonLabel))` attribútummal, így konfigurálva, hogy amennyiben az `IsDownloading` értéke megváltozik, akkor a `DownloadButtonLabel`-re is ki kell váltani a `PropertyChanged` eseményt.
5. A `DownloadCommand` és az `ImageSelectedCommand` parancsokat töröljük. Helyette készítsünk két privát metódust `Download()` és `ImageSelect()` néven, a parancsokhoz korábban tartozó tevékenységgel. Lássuk el mindkét metódust a `RelayCommand` attribútummal, így a projekt újrafordításával megtörténik a `DownloadCommand` és az `ImageSelectCommand` kódgenerálása. (*Megjegyzés: a metódus neve szándékosan `ImageSelect`, ugyanis `ImageSelected` szibólum néven már van egy esemény ebben az osztályban.*)
6. A `DownloadCommand`-hoz tartozó `RelayCommand` attribútumot lássuk el az `AllowConcurrentExecutions=true` argumentummal. Erre azért van szükség, mert a `RelayCommand` alapértelmezetten nem engedélyezné egy parancs újbóli végrehajtását, amíg az előző futtatása még zajlik. Erre pedig itt szükségünk van, mert a parancs második meghívásával érhető el a letöltés megszakítása.