Eseményvezérelt alkalmazások: 11. gyakorlat

A munkafüzet az *Avalonia UI keretrendszer* használatába vezet be minket. Az Avalonia UI egy a .NET környezetre épülő, cross-platform keretrendszer eseményvezérelt grafikus alkalmazások fejlesztésére. Támogat több asztali (Windows, Linux, macOS) és mobil (Android, iOS) platformot is, valamint WebAssemblyként böngészőben is futtatható. Lehetővé teszi a fejlesztést MV architektúrában is, de kifejezetten az MVVM architektúra használata javasolt.

A feladat egy több platformon is futtatható **aszinkron kép letöltő alkalmazás elkészítése**, amely segítségével egyszerűen listázhatjuk az egy weboldalon megjelenő képeket, majd azokat egy külön ablakban megnyitva nagyobb méretben tekinthetjük meg.

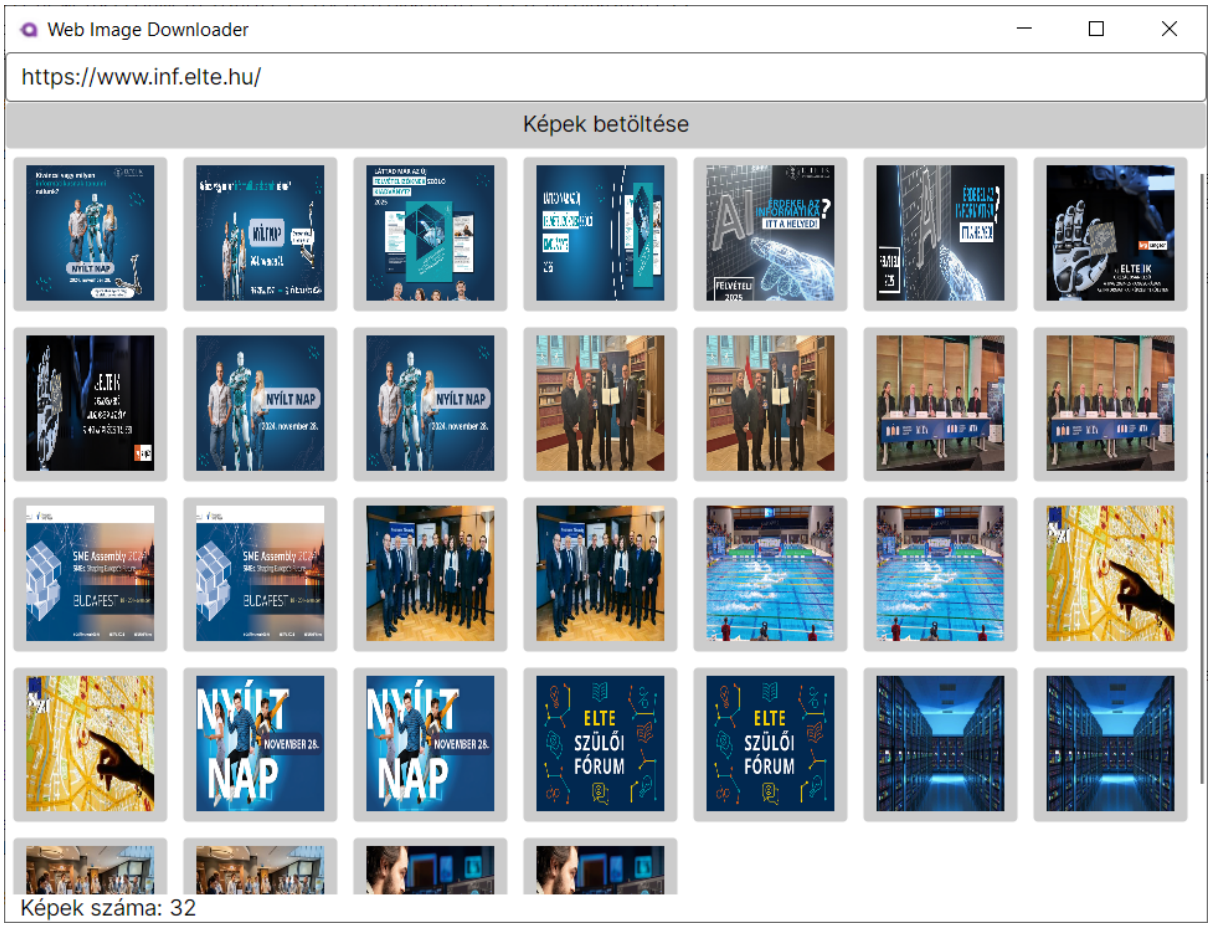

Figure 1: Windows App

Figure 2: Android App

Az alkalmazást *Avalonia UI* keretrendszerrel, háromrétegű (modell-nézet-nézetmodell) architektúrában, eseményvezérelt paradigma alapján valósítjuk meg. Az Avalonia UI használatával egyetlen kódbázist készítve fejleszthetünk olyan alkalmazást, amelyet több platformon is tudunk futtatni.

1 Előkészületek Avalonia UI alkalmazás fejlesztéséhez ^{KM}

Első lépésként telepítenünk kell az *Avalonia for Visual Studio 2022* kiegészítőt a Visual Studio-hoz.

1. Indítsuk el a Visual Studio-t és válasszuk az *Extensions* menüből a *Manage extensions* menüpontot.
2. Keressünk rá az *Avalonia for Visual Studio* kiegészítőre, majd telepítsük. Ezzel elérhetővé válik az Avalonia UI felületű alkalmazások grafikus tervezése és debuggolása Visual Studio-ban.
3. Figyeljük meg, hogy ez az *Avalonia Template Studio* kiegészítőt is telepíti, ilyen módon új projekt létrehozásakor már Avalonia UI-os sablonok közül is választhatunk.

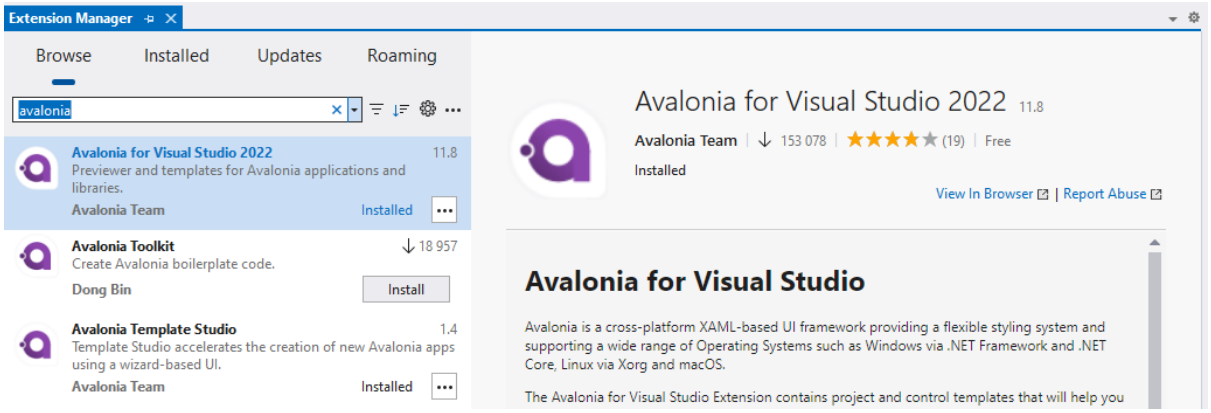

Figure 3: Kiegészítők kezelése Visual Studio-ban

JetBrains Rider fejlesztőkörnyezet használata esetén az *AvaloniaRider* plugin telepítésével kaphatunk hasonló támogatást az IDE-ben. A projekt sablonokat az alábbi konzol utasítással telepíthetjük:
`dotnet new install Avalonia.Templates`

1.1 Android platform fejlesztési támogatása

Második lépésként, az Android platforma fejlesztéshez telepítenünk kell a Java SDK-t, az Android SDK-t és a .NET SDK Android Workload-ot. A legegyszerűbben ezt úgy tehetjük meg, ha a Visual Studio 2022-höz a teljes *Multi-platform App UI (MAUI) development* csomagot telepítjük. (A géptermi gépeken ez már előzetesen telepítésre kerül.)

1. A telepített programok közül válasszuk ki a *"Visual Studio Installer"* alkalmazást.
2. A megjelenő listából válasszuk ki a Visual Studio 2022-t, majd kattintsunk a *Modify* gombra.
3. A Workloads fülön pipáljuk be a *.NET Multi-platform App UI development* lehetőséget, majd kattintsunk a *Modify* gombra.

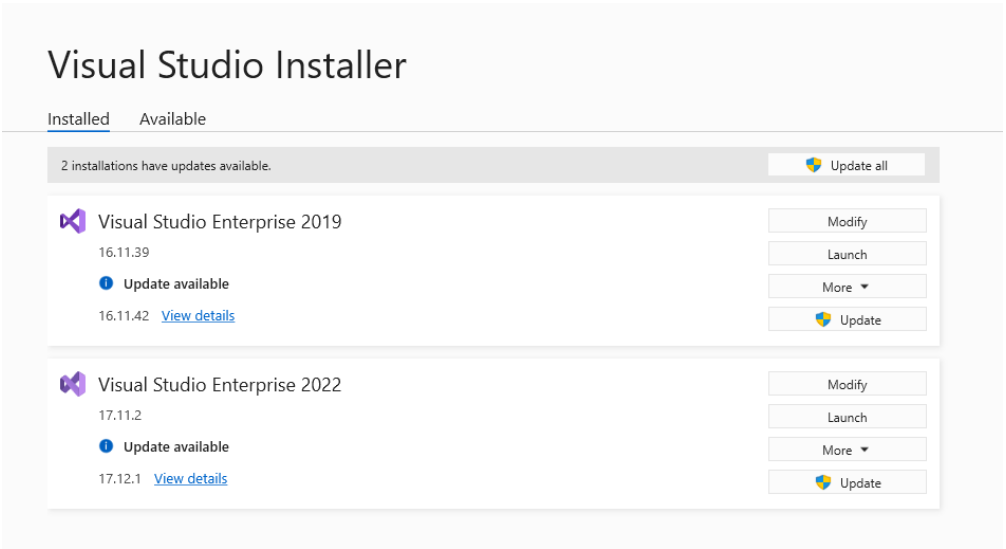

Figure 4: Installer lista

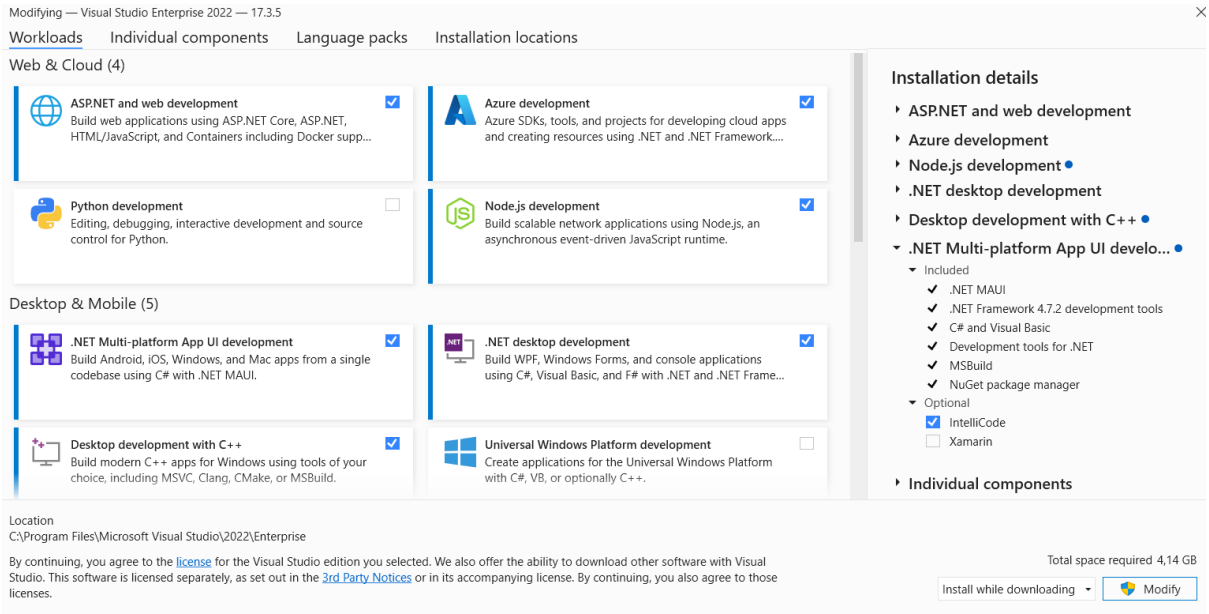

Figure 5: Installer Workloads: MAUI

> *JetBrains Rider* fejlesztőkörnyezet használata esetén külön szükséges Java SDK-t és Android SDK-t telepítenünk egy preferált forrásból. A .NET SDK Android Workloadja a következő utasítással telepíthető:
> `dotnet workload install android`

1.2 Hyper-V engedélyezése

A Visual Studioval egyszerűen futtathatóak, tesztelhetőek és debuggolhatóak az Avalonia UI alkalmazások virtuális Androidhoz eszközön is, emulátor segítségével. Ha azonban a hardveres gyorsítás (*hardware acceleration*) nem érhető el az adott számítógépen (vagy nincs engedélyezve), akkor az emulátor jellemzően meglehetősen lassú lesz. A hardveres gyorsítás engedélyezésével az emulálás sebessége szignifikánsan javítható.

A hardveres gyorsítást legegyszerűbben a Hyper-V engedélyezésével kapcsolhatjuk be, ehhez rendszergazdaként a *Windows-szolgáltatások be-és kikapcsolása* vezérlőpanelen (angol nyelvű operációs rendszer esetén: *Turn Windows features on or off*) kapcsoljuk be a jelölt Hyper-V szolgáltatásokat, majd indítsuk újra a számítógépet.

A géptermi gépeken a Hyper-V már engedélyezve van!

> *Tipp:* A Hyper-V a Windows 10/11 *Home* verziójában nem érhető el, azonban az Azure Dev Tools for Teaching akadémiai együttműködési program keretében a Visual Studio 2022 Enterprise verziója mellett többek között a Windows 10/11 Education verziója is jogtisztán beszerezhető az ELTE IK hallgatói számára (INF-es azonosítóval bejelentkezve). A Windows 10/11 Education verziója a Pro változattal közel azonos funkciókészlettel rendelkezik.
> Amennyiben valaki már telepített Windows 10/11 Home verzióval rendelkezik, frissíthet is az Education verzióra. Ehhez a *Gépház -> Névjegy* ablakban (angol változatban: *Settings -> About*) van lehetőség, az *Azure Dev Tools for Teaching*-ből beszerzett termékkulcsot megadva.

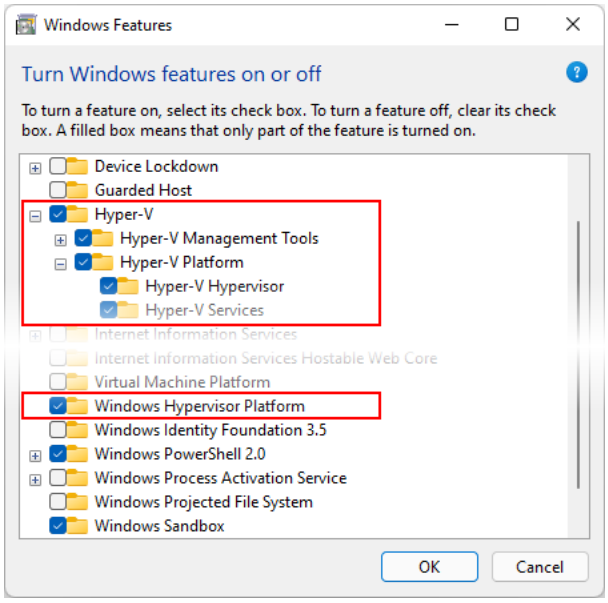

Figure 6: Hyper-V szolgáltatások bekapcsolása

1.3 iOS platform fejlesztési támogatása ^{OP}

Technikai akadálya nem lenne, de Windows operációs rendszerről licenszelési problémák miatt nem tudunk sem iOS-re se macOS-re fordítani. Ehhez szükségünk lesz egy XCode-ot futtató Mac-re, fizikai eszköz vagy emulátor (szimulátor) használat esetén is. Amennyiben valaki rendelkezik ilyen eszközökkel, innentől alapvetően két lehetősége van:

- Avalonia alkalmazásunkat Mac-en fejlesztjük (van Visual Studio Mac-re is, Azure Dev Tools for Teachingből letölthető) és ott az alkalmazást iOS Simulátorban futtatjuk vagy telepíthetjük egy kapcsolt fizikai eszközre is. XCode telepítésére szükség lesz. Ld. a dokumentációt.
- Egy Windowsos fejlesztői számítógépen dolgozunk továbbra is, ami egy macOS-es számítógépen keresztül telepíti az alkalmazást a telefonra. Ez már kicsit összetettebb lépéseket igényel, konfigurálása egyszeri 20-30 percet igényelhet. Az ehhez vonatkozó MAUI-s dokumentáció Avalonia esetében is használható. Megjegyzések:
  - XCode telepítése itt is szükséges lesz a macOS-es gépre. Fontos, hogy legalább egyszer el is kell indítani, mert az első indítással lesz teljes a telepítés.
  - Ha az XCode már telepítve van, de mégis olyan hibaüzenet jelentkezik, mintha nem lenne, érdemes megpróbálni azt, hogy az XCode-on belül a *Locations* menü *Command line tools* beállításánál újra kiválasztani az egyébként is kiválasztott elemet a listából.

2 Projekt létrehozása ^{KM}

Készítsünk Visual Studioban egy új *Avalonia C# Project*-et, a *solution* és a *projekt* neve legyen *ImageDownloader.Avalonia*.

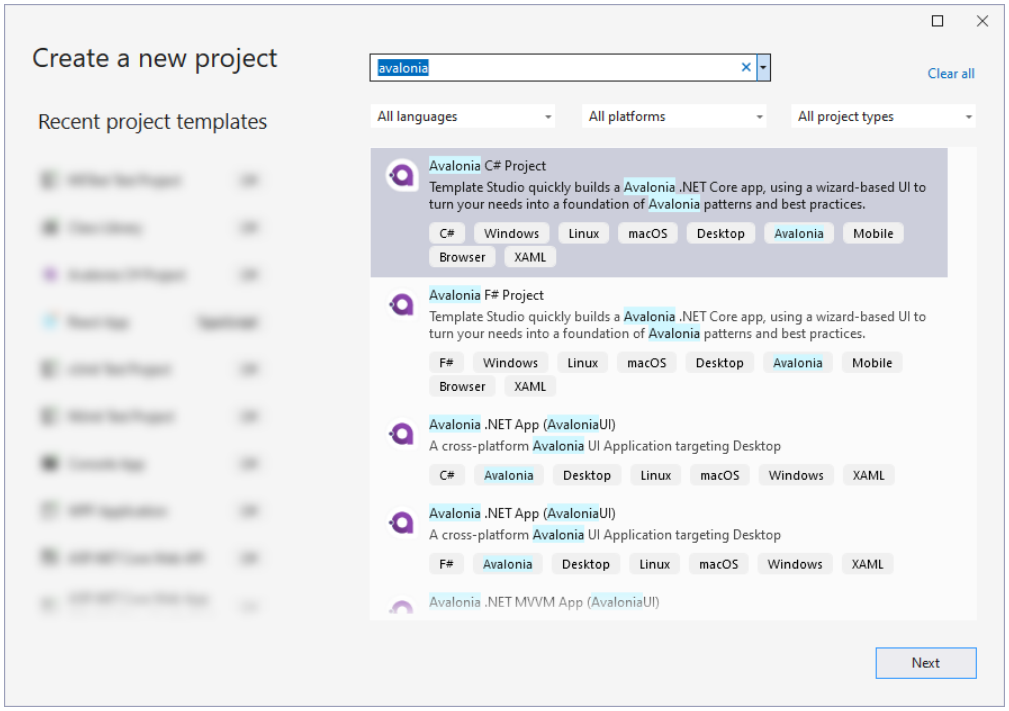

Figure 7: Projekt létrehozása

A projekt létrehozását egy több lépéses varázsló segíti. Válasszuk ki a *Desktop* és *Android* platformokat támogatásra, alkalmazott tervezési mintaként (*design pattern*) pedig a *Community Toolkit*-et. A varázsló 3. lépésben a *Feature*-k kiválasztása most nem fontos.

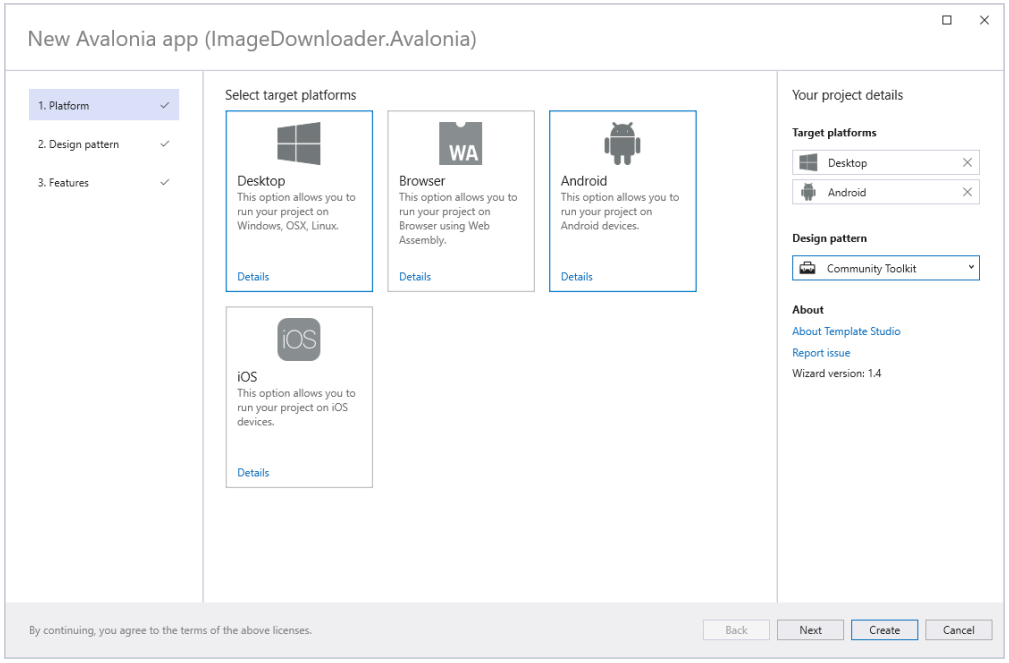

Figure 8: Támogatott platformok és tervezési minta kiválasztása

3 Fordítás több platformra ^{KM}

Az Avalonia alkalmazások multi-projekt felépítést követnek, azaz a `ImageDownloader.Avalonia` projekt egy *Class Library* projekt, és a `.Desktop`, valamint `.Android` projektekből készülnek futtatható binárisok.

A futtatás az egyes platformokon a következő módon lehetséges:

- Windows-on: válasszuk ki a `ImageDownloader.Avalonia.Desktop` projektet, és a megszokott módon fordítsuk, illetve futassuk. A programunk így fordított változata a többi támogatott asztali operációs rendszeren is futtatható.
- Androidon: válasszuk ki a `ImageDownloader.Avalonia.Android` projektet, majd mellette egy fizikai eszközt vagy emulátort a futtatáshoz.
  - Ahhoz, hogy egy fizikai Android készülékre tudjunk fordítani, a készüléknek fejlesztői módban kell lennie, illetve a fejlesztői beállításoknál az USB hibakeresésnek engedélyezve kell lennie. Ld. a kapcsolódó MAUI-s dokumentációt ennek beállításához.
  - Emulátort a *Tools -> Android -> Android Device Manager* menüpontban hozhatunk létre, ha alapértelmezetten nem jött volna létre egy a MAUI workload telepítésekor.

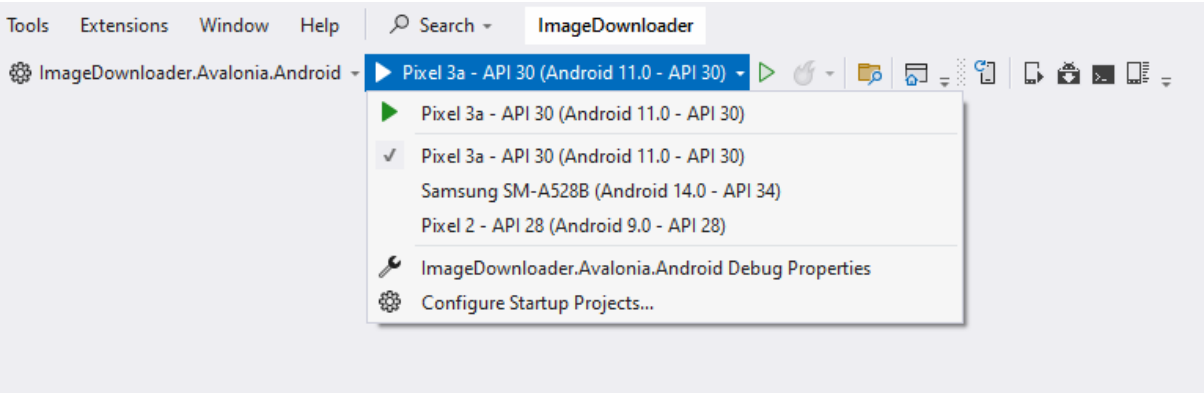

Figure 9: Alkalmazás indítása Androidon

4 Modell réteg megvalósítása ^{KM}

A modell réteget nem szükséges megvalósítanunk, hiszen az alkalmazás a modell réteg tekintetében megegyezik a *9. gyakorlaton* megvalósított képletöltő alkalmazásunk modell rétegével.

A 9. gyakorlaton létrehozott WPF alkalmazás egyetlen projektként készült el, azaz nincs szétbontva külön model és view projektre. A teljes projekt pedig a WPF mivolta miatt csak a Windows platformot támogatja (a jelenlegi alkalmazásunk pedig cross-platform).

Megjegyzés: ez jól látható a *csproj* fájlokat megnyitva a `TargetFramework` mezőben is.
WPF-es ImageDownloader:

```
<TargetFramework>net8.0-windows</TargetFramework>
```

Avalonia UI-os ImageDownloader (platformfüggetlen projekt):

```
<TargetFramework>net8.0</TargetFramework>
```

Emiatt a korábbi *ImageDownloader* projekt model részét emeljük át a jelenlegi solution-be egy új, platformfüggetlen *Class Library* projektbe. Ezen *class library* projekt neve lehet például: *ImageDownloader.Model*. A keretrendszernél itt választhatjuk a .NET 8-at, mint *LTS* verziót.

Adjuk hozzá az *ImageDownloader.Avalonia* projektünkhöz a létrehozott *ImageDownloader.Model* projektet függőségként (*"Add Project Reference"*).

5 Nézetmodell réteg megvalósítása ^{KM}

Az *ImageDownloader.Avalonia* projektben már adott egy *ViewModels* mappa. A benne lévő fájlokat töröljük, és helyette az alábbi osztályokat / fájlokat hozzuk létre:

- `DelegateCommand`: megvalósítása egyezzen meg a 9. gyakorlaton használt megvalósítással.

- `ViewModelBase`: megvalósítása egyezzen meg a 9. gyakorlaton használt megvalósítással.
- `MainViewModel`: megvalósítását lásd lejjebb.

5.1 MainViewModel osztály

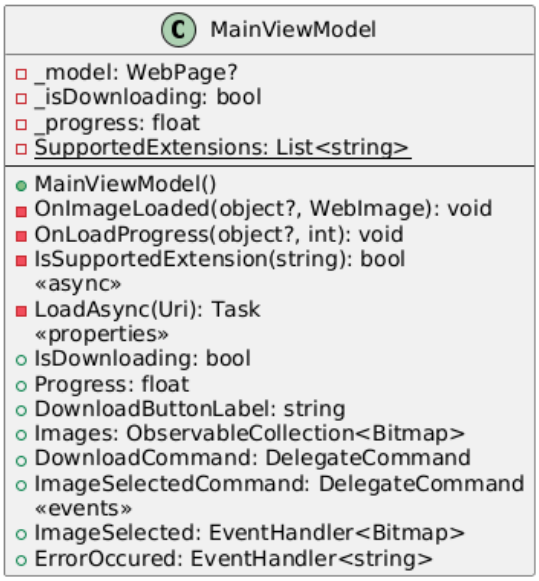

Figure 10: `MainViewModel` osztálydiagramja

A `MainViewModel` nagyrészt megegyezik a 9. gyakorlaton elkészített `MainViewModel` osztállyal, viszont szükséges néhány átalakítás. Ezért induljunk ki ebből.

5.2 Képek kezelése

Avalonia UI-ban a WPF-es `System.Windows.Media.Imaging.BitmapImage` típus helyett az `Avalonia.Media.Imaging.Bitmap` típust használthasztjuk az alábbi helyeken:

- `Images` tulajdonság,
- `ImageSelected` esemény,
- a konstruktorban az `Images` példányosításakor,
- az `ImageSelectedCommand`-ban az argumentum típsuának ellenőrzésekor.

5.3 `LoadAsync` átalakítása

Kezeljük a `_model.LoadImagesAsync()` hívás által dobott esetleges kivételeket, hiszen mobil eszköz esetén a Internet kapcsolat stabilitása is kevésbé megbízható. Avalonia UI keretrendszerben üzenetablak feldobásához használjuk a `MessageBox.Avalonia` NuGet csomagot. Az üzenetablakot azonban ne a nézetmodell jelenítse meg, hanem definiáljunk egy eseményt (pl. `ErrorOccured`), amelyre a vezérlési rétegben feliratkozva tudjuk majd megjeleníteni a felhasználónak a hibaüzenetet.

```
await MessageBoxManager.GetMessageBoxStandard(
    "Cím",
    "Üzenet",
    ButtonEnum.Ok, // gombok
    Icon.Error)    // ikon
.ShowAsync();
```

A `MessageBox.Avalonia` NuGet csomag telepítése előtt érdemes lehet a projekt sablon által hozzáadott Avalonia NuGet csomagok frissítése, az esetleges kompatibilitási problémák elkerülése érdekében.

5.4 `OnImageLoaded` eseménykezelő átalakítása

Az `ImageLoaded` esemény minden kiváltásakor a modell sikeresen letöltött egy új képet, ennek megfelelően ezt a nézeten is megjeleníthetjük.

1. Ellenőrizzük, hogy a letöltött kép kiterjesztése az alábbiak közül valamelyik-e: *jpg, jpeg, png, gif.* Ezt az eseményargumentumban érkezett `webImage.Url.LocalPath` vizsgálatával tudjuk megtenni.
2. Készítsünk egy új memóriabeli képet: `new MemoryStream(webImage.Data)`
3. A `MemoryStream`-ből készítsünk egy `Bitmap`-et, amit már az `Images` kollekcióhoz adhatunk.

6 Nézet réteg megvalósítása ^{EM}

Az *ImageDownloader.Avalonia* projektben már adott egy *Views* mappa.

A fő nézetet (`MainView.axaml`) a *9. gyakorlaton* megvalósított WPF ablak mintájára készítsük el. Mindössze a következő eltérésekre kell készülnünk:

- Az URL címet bekérő szövegdoboz tartalmára a gomb parancsának paraméterénél eltérő, egyszerűbb szintaxissal tudunk hivatkozni:
  `{Binding #UrlTextBox.Text}`
- Nem lesz szükségünk a `BooleanToVisibilityConverter`-re, ugyanis az Avalonia UI-ban a `ProgressBar`-nak egy logikai értéket váró `IsVisible` tulajdonsága van.
- Nincsen `StatusBar` panel az Avalonia UI keretrendszerben, helyette egy horizontálisan orientált `StackPanel`-t használhatunk.
- A képek megjelenítését most is gombokba ágyazzuk (`Button`), hogy a kattintásuk parancskötéssel kezelhető legyen. A 9. 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 `DockPanel`-t elnevezzük (legyen pl. `Panel` a neve), így a kötést annak a vezérlőnek a `DataContext`-jére végezhetjük el:
  `{Binding #Panel.((vm:MainViewModel)DataContext).ImageSelectedCommand}`

7 Vezérlés ^{EM}

Kapcsoljuk hozzá a létrehozott nézetmodellünket a nézethez. Ezt az alábbi lépésekben tegyük meg:

1. Az `App.axaml.cs` osztály `OnFrameworkInitializationCompleted` metódusában hozzunk létre egy új `MainViewModel` példányt.
2. A `DataContext` *propertynek* állítsuk be a létrehozott `MainViewModel`-t, mind az asztali, mind a mobilos alkalmazás életciklus esetén.
3. Ne feledkezzünk meg feliratkozni a nézetmodell `ErrorOccured` eseményére és jelenítsük meg a kért hibaüzenetet egy felugró ablakban a `MessageBoxManager` használatával.

8 Képmegtekintő megvalósítása ^{EM}

8.1 Nézetmodell

Figure 11: `ImageViewModel` osztálydiagramja

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

Az osztályba a 9. 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).

8.2 Nézet

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

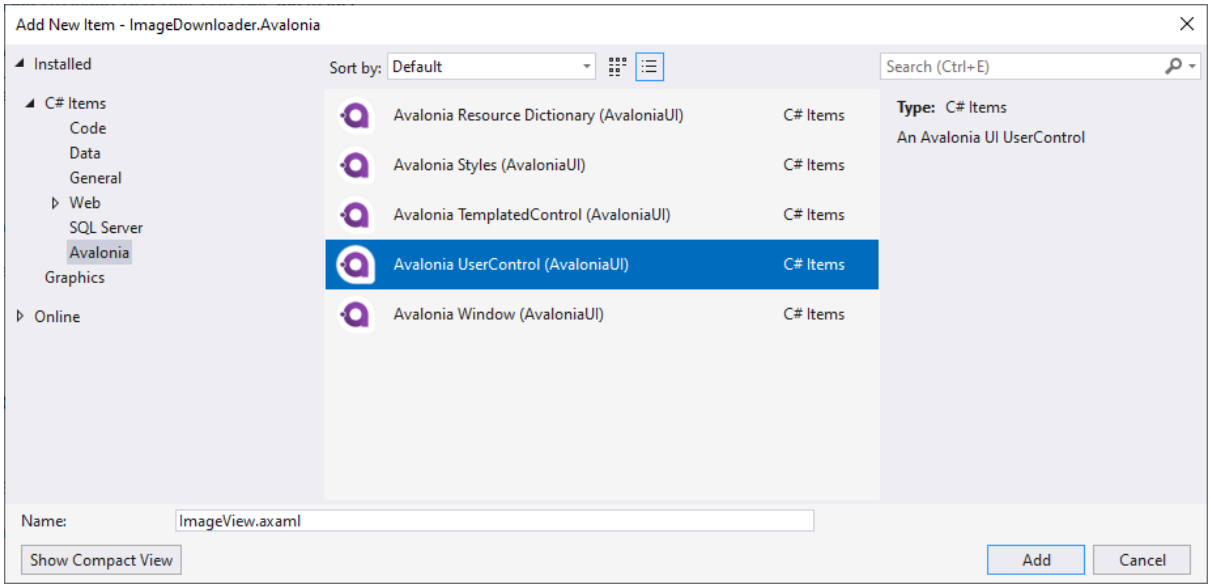

Figure 12: Új nézet hozzáadása a projekthez

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.

8.2.1 Bezárás gomb kondícionális megjelenítése ^{OP}

A *Bezárás* gombra igazából csak mobil platform, azaz nem ablakos megjelenítés esetén van szükségünk. Az `OnPlatform` XAML vezérlővel kondícionálisan, platformspecifikusan alakíthatjuk ki a felületet. Fontos, hogy egy alapértelmezett működést is meg kell adnunk, ami nem lehet üres, ezért ott egy kevés erőforrást használó, általános `Control` vezérlőt helyezünk el, amit elrejtünk:

```
<OnPlatform>
  <OnPlatform.Default>
    <Control IsVisible="False" />
  </OnPlatform.Default>
  <OnPlatform.Android>
    <Button Content="Bezárás" ... />
  </OnPlatform.Android>
</OnPlatform>
```

8.3 Vezérlés

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 9. 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
```

9 MVVM Toolkit használata ^{OP}

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.