Eseményvezérelt alkalmazások: 11. gyakorlat

A gyakorlat célja az ismerkedés a **.NET MAUI** *cross-platform* keretrendszerrel. 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.

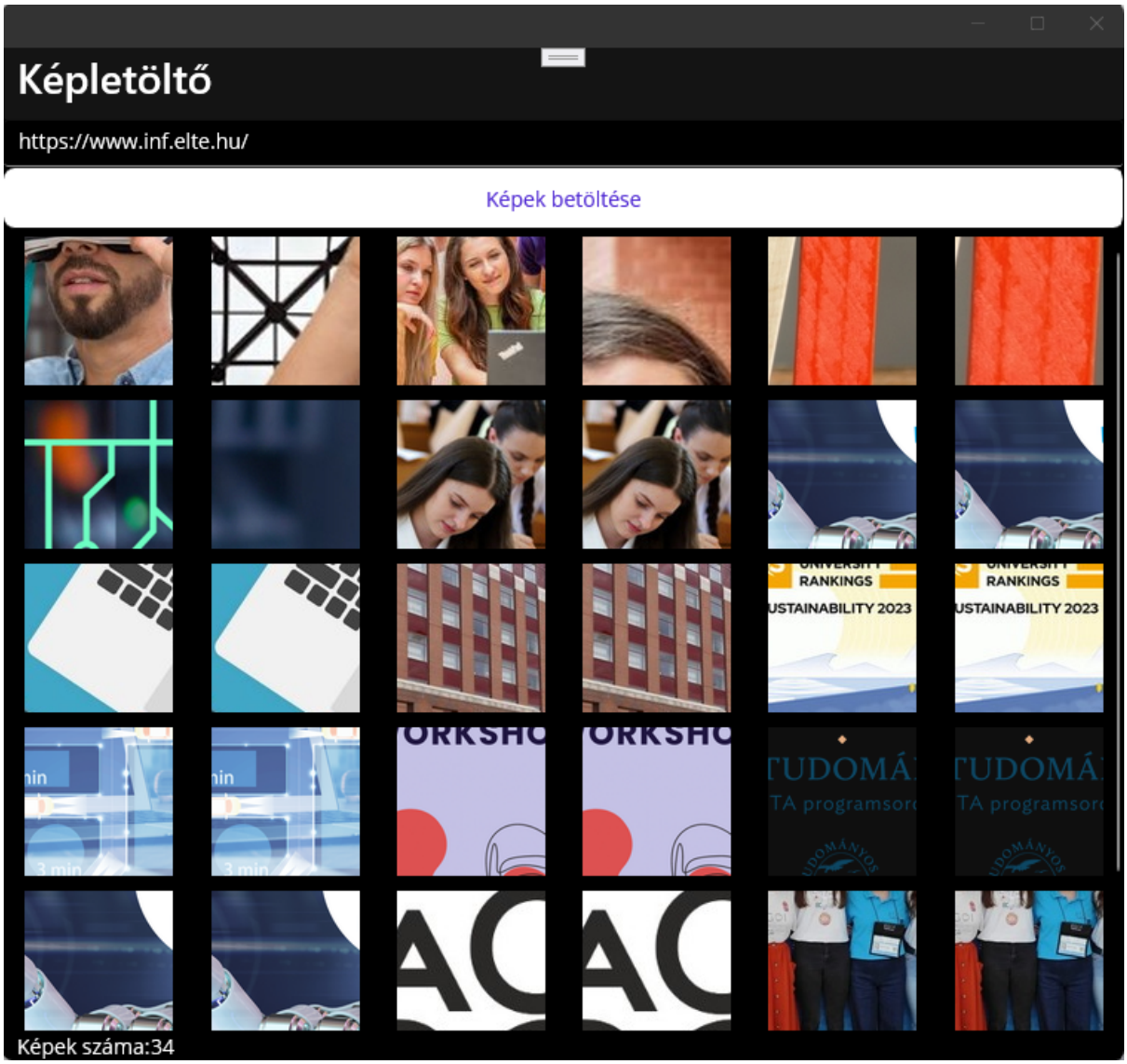

Figure 1: Windows App

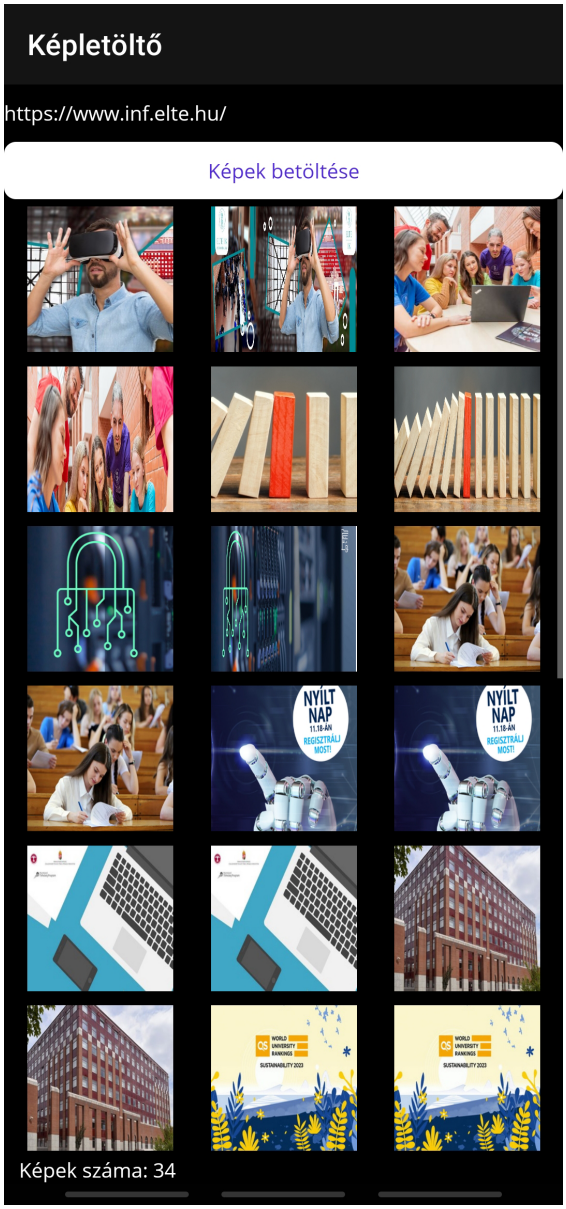

Figure 2: Android App

Az alkalmazást *.NET MAUI* keretrendszerrel, háromrétegű (modell-nézet-nézetmodell) architektúrában, eseményvezérelt paradigma alapján valósítjuk meg. A .NET MAUI használatával egyetlen kódbázist készítve fejleszthetünk olyan alkalmazást, amelyet az alábbi platformokon tudunk futtatni:

- Android
- IOS
- macOs
- Windows

1 Előkészületek .NET MAUI alkalmazás fejlesztéséhez

Első lépésként telepítenünk kell a Visual Studio 2022-höz a MAUI csomagot.

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.

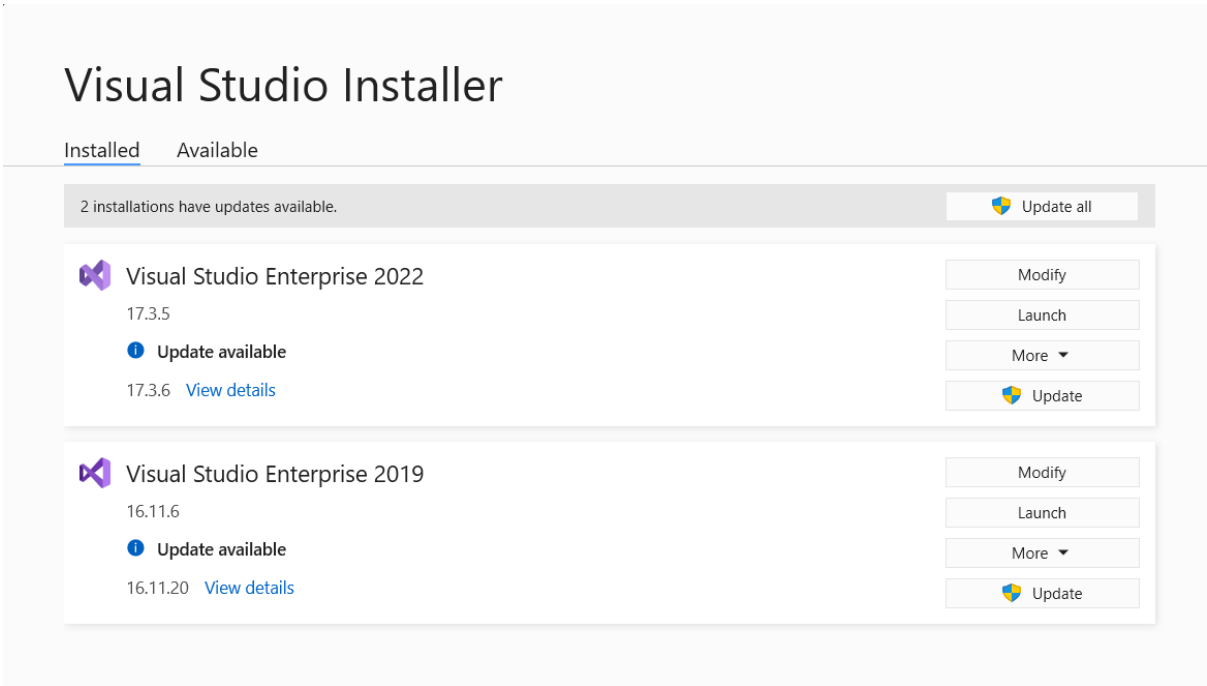

Figure 3: Installer lista

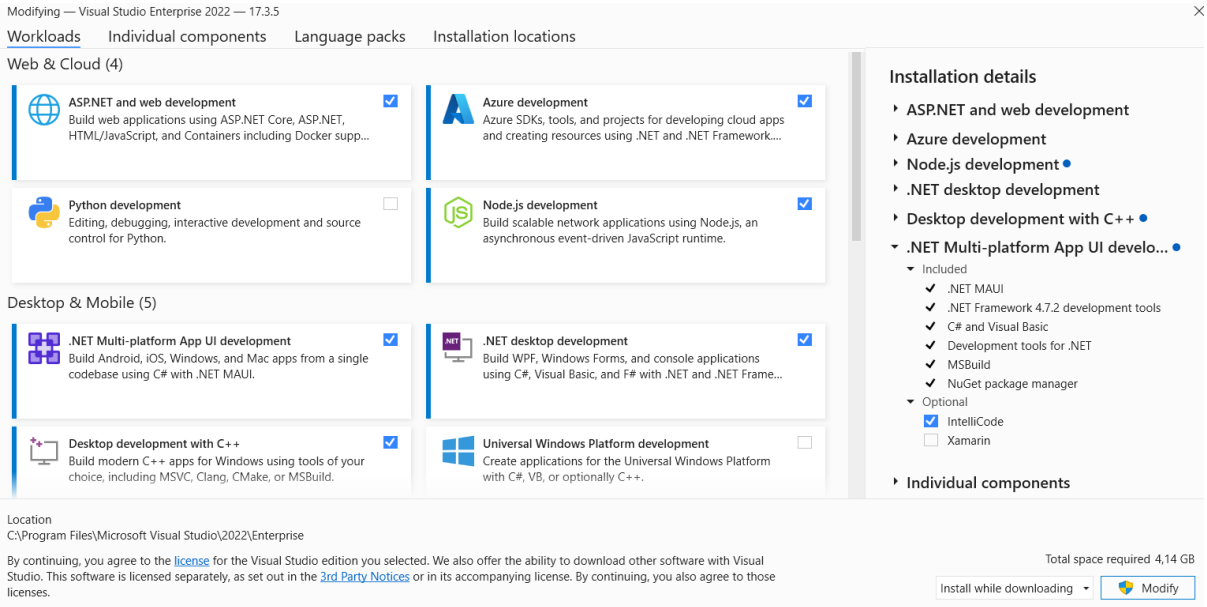

Figure 4: Installer Workloads Maui

Megjegyzés: A .NET MAUI alkalmazás fejlesztéséhez Visual Studio 2022 (vagy magasabb) verziót kell használni. A korábbi Visual Studio verziók **nem** támogatják a MAUI alkalmazások fejlesztését.

1.1 Hyper-V engedélyezése

A Visual Studioval egyszerűen futtathatóak, tesztelhetőek és debuggolhatóak a .NET MAUI 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.

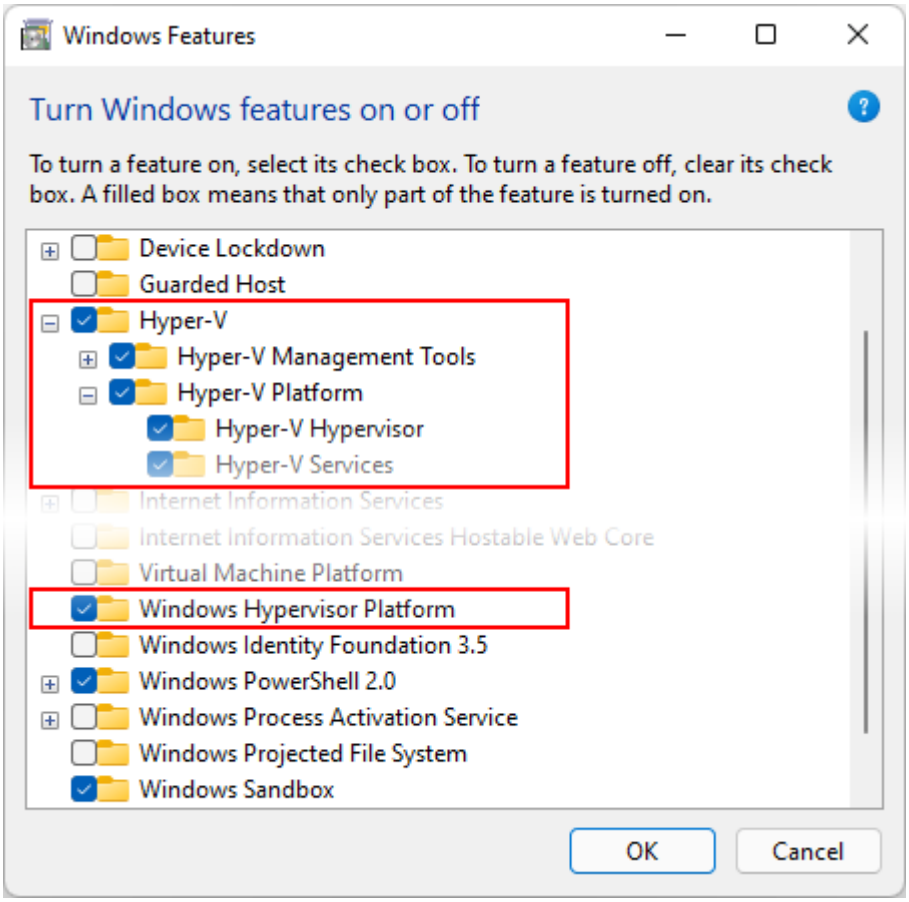

Figure 5: Hyper-V szolgáltatások bekapcsolása

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.

Megjegyzés: alternatívaként Intel processzorral az Intel saját hardveres gyorsítása, a HAXM (*Intel Hardware Accelerated Execution Manager*) is igénybe vehető. Erről részletesebben lásd a dokumentációt.

2 Projekt létrehozása

Készítsünk Visual Studioban egy új *.NET MAUI App* projektet, a *solution* neve legyen *ImageDownloader*, a projekté pedig *ImageDownloader.Maui*.

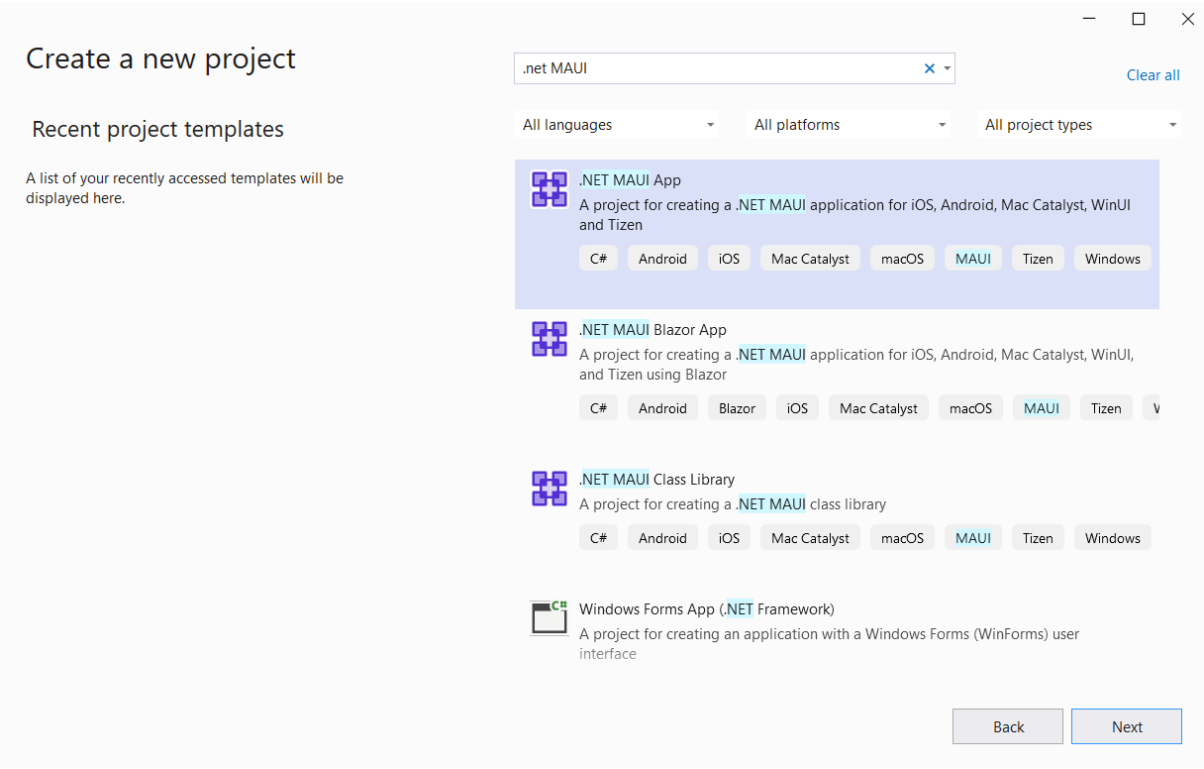

Figure 6: Projekt létrehozása

3 Fordítás több platformra

A menüsorban található alkalmazást indító ikon átváltozott, itt az indítandó projekt neve helyett a futtatáshoz használt platform jelenik meg. A futtatás az egyes platformokon a következő módon lehetséges:

- Windows-on: A lenyíló menüben válasszuk ki a Windows Machine lehetőséget
- Androidon:
  - Fizikai android eszköz: A megjelenő menüben ki kell választani, hogy mely fizikai eszközre szeretnénk, hogy leforduljon az alkalmazás.
  - Emulátor: Windows oprációs rendszeren van lehetőségünk Android készüléket emulálni. Ehhez szükséges egy emulátor telepítése. Itt tudjuk kiválasztani, mely emulált készülékre forduljon le az alkalmazásunk. Az android emulátor telepítése is ebből a menüpontból kezdeményezhető. Link az emulátor telepítéséhez
- iOS, macOs: Ugyan 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 is fizikai eszköz vagy emulátor (szimulátor) használat esetén is. További információkért ld. a dokumentációt.

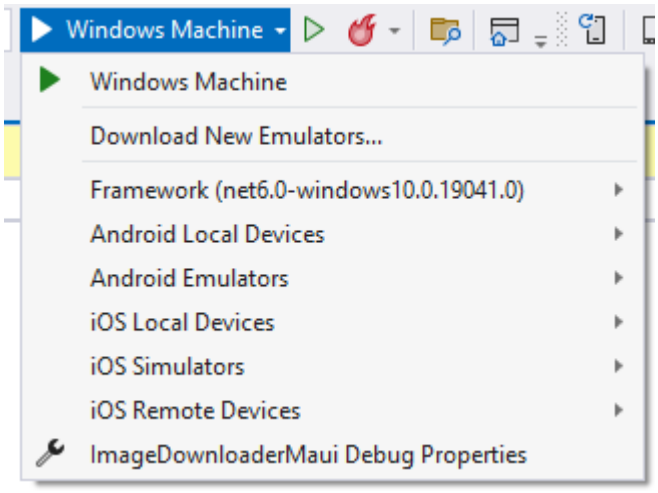

Figure 7: Alkalmazás indítása

Megjegyzés: Az *Android Local Devices* csak akkor jelenik meg, ha ténylegesen van csatlakoztatva Android készülék. Ahhoz, hogy egy Android készülékre tudjunk fordítani, a készüléknek fejlesztői módban kell lennie, illetve a fejlesztői beállításoknál az USB hibakeresésnek engedélyezve kell lennie.
Bővebben a fejlesztői módról illetve USB hibakeresés beállításról.

4 Modell réteg megvalósítása

A modell réteget nem szükséges a nulláról 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 ImageDownloader:

```
<TargetFramework>net6.0-windows</TargetFramework>
```

MAUI ImageDownloader:

```
<TargetFrameworks>net6.0-android;net6.0-ios;net6.0-maccatalyst</TargetFrameworks>
```

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 a .NET 6-ot válasszuk.

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

5 Nézetmodell réteg megvalósítása

Az *ImageDownloader.Maui* projektben hozzunk létre egy *ViewModel* mappát, és benne az alábbi osztályokat / fájlokat: - `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

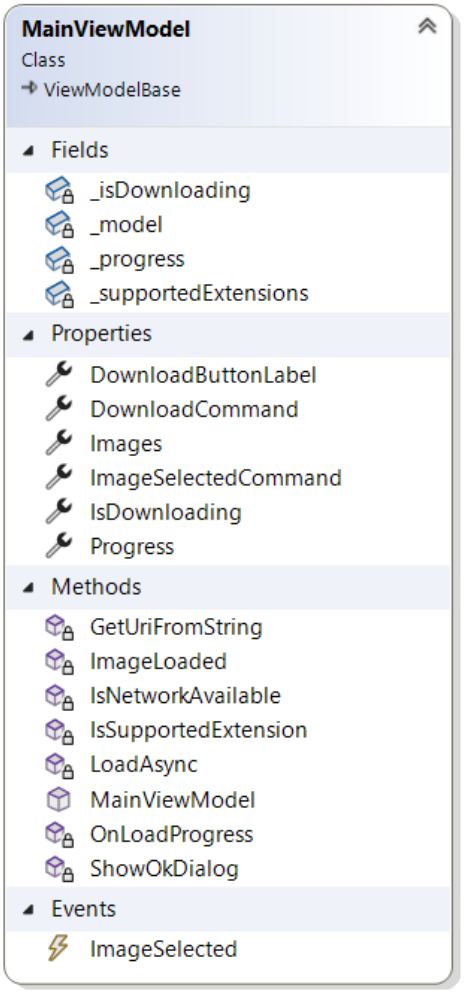

Figure 8: `MainViewModel` osztálydiagramja

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

5.2 `BitmapImage -> ImageSource`

Maui-ban nincs `BitmapImage` típus, ezért helyette az alábbi helyeken használjuk az `ImageSource` típust:

- `ObservableCollection` tulajdonság,
- `ImageSelected` esemény,
- a konstruktorban `ObservableCollection<ImageSource>` példányosításakor.

5.3 `LoadAsync` átalakítása

A `LoadAsync(Uri)` aszinkron metódusban végezzük el az alábbi műveleteket is:

1. Ellenőrizzük, hogy rendelkezünk-e Internet kapcsolattal. Amennyiben nem, úgy dobjunk fel egy üzenetablakot a felhasználónak, hogy először csatlakozzon az Internethez.
    - Az Internet kapcsolat ellenőrzéséhez használjuk a `Connectivity.Current.NetworkAccess` propertyt. Ez egy *NetworkAccess* enumerációs értéket fog visszaadni. Amennyiben ez az érték nem egyezik meg a `NetworkAccess.Internet`-el, akkor nincs Internet kapcsolat.
    - Maui-ban az üzenetablak feldobásához használjuk a `DisplayAlert` metódust. 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.

2. Ellőnőrizzük, hogy az *entry*-ben megadott URL valóban URL formátumú-e illetve, hogy ez abszolút URL-e? Ellenkező esetben dobjunk szintén váltsuk ki az `ErrorOccured` eseményt és szakítsuk meg a képek betöltésének folyamatát.

5.4 `OnLoadProgress` eseménykezelő átalakítása

Az `OnLoadProgress` eseménykezelőben a `Progress` tulajdonságot állítsuk be a paraméterként kapott érték 0.01-szeresésre, ugyanis a WPF-től eltérően MAUI-ban a folyamatjelző értéktartománya 0 és 1 között definiált. Ügyeljünk egy esetleges egész osztás elkerülésére!

5.5 `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 `ImageSource`-ot:
   ```
   var imageSource = ImageSource.FromStream(() => new MemoryStream((webImage.Data)));
   ```
4. Az így elkészített `ImageSource` objektumot adjuk hozzá az `Images` kollekcióhoz.

6 Nézet réteg megvalósítása

Az *ImageDownloader.Maui* projektben hozzunk létre egy *View* mappát.

A fő ablakot (`MainPage.xaml`) a hozzá tartozó osztállyal (`MainPage.xaml.cs`) helyezzük át a létrehozott *View* mappába.

- A fő képernyőhöz tartozó `ContentPage` tartalmát távolítsuk el a XAML kódból.
- A fő képernyőhöz tartozó *C#* osztályból töröljük ki az `OnCounterClicked` eseményhez kapcsolódó eljárást.
- A fő képernyőn állítsuk be a címet (*Title*), például a *"Képletöltő"* kifejezésre.

6.1 MainPage

A vezérlőket egy `Grid`, valamint `StackLayout`-ok segítségével rendezhetjük el a kívánt módon. A `FlexLayout` vezérlőt `ScrollView`-ba helyezve tehetjük a tartalmát görgethetővé. A grid rács 3 sort tartalmazzon.

Az **1.sor** tartalmazzon egy `StackLayout` vezérlőt:

- Ennek az `Orientation`-je legyen `Vertical`
- Továbbá tartalmazza az alábbi elemeket:
  - egy `Entry` vezérlőt az URL megadásra;
  - egy `Button` vezérlőt a letöltés megkezdésére.
- A letöltés gombot kössük a nézetmodell `DownloadCommand` parancsához. A `CommandParameter` értéke a szöveges beviteli mező (`Entry`) értéke legyen.

A **2. sor** az alábbiakat tartalmazza:

- Egy `ScrollView` vezérlőt sok kép esetén az oldal görgethetőségének biztosítása érdekében.
- A `ScrollView`-n belül egy `FlexLayout` elrendező vezérlőt. Itt legyenek beállítva az alábbi tulajdonságok:
  - `BindableLayout.ItemsSource` a nézetmodell `Images` kollekciójára kötve,
  - a `Wrap` értéke *"Wrap"*, a `Direction` értéke *"Row"*, a `JustifyContent` értéke *"SpaceEvenly"* legyen.

- A `FlexLayout`-on belül legyen egy `BindableLayout.ItemTemplate`, azon belül egy `DataTemplate` a megjelenített elemek sablonjának megadására.
- A `DataTemplate`-n belül legyen egy `ImageButton` az alábbi tulajdonságokkal:
  - a `Source` bindolja a képet,
  - a `WidthRequest` és a `HeightRequest` értéke 100 legyen,
  - az `Aspect` értéke *"Fill"*, a `Margin` értéke 5 legyen,
  - a `Command`-ot pedigkésőbb fogjuk megvalósítani.

> *Megjegyzés:* Image helyett azért készítünk ImageButton-t, mert a sima Image-nek nincs Command-ja. Nekünk azonban szükségünk lesz rá, hiszen ezzel tudunk elnavigálni az új oldalra, ahol a képet nagyobb méretben is megtekinthetjük. Ezzel a későbbiekben foglalkozunk majd.

A `FlexLayout` vezérlőt a nézetmodell-ben megfigyelhető `Images` kollekciója alapján, dinamikusan töltsük fel, a `DataTemplate` segítségével megadva, hogy minden elemét egy `ImageButton` vezérlővel kívánunk helyettesíteni.

```
<FlexLayout BindableLayout.ItemsSource="{Binding Images}"
            Wrap="Wrap" Direction="Row" JustifyContent="SpaceEvenly">
    <BindableLayout.ItemTemplate>
        <DataTemplate>
            <ImageButton Source="{Binding}"
                   WidthRequest="100" HeightRequest="100"
                   Aspect="Fill" Margin="5" />
        </DataTemplate>
    </BindableLayout.ItemTemplate>
</FlexLayout>
```

> *Megjegyzés:* A `FlexLayout` a CSS3-ból is ismert *flex-box*-okhoz hasonlóan működik, a nevét is onnan kapta. A FlexLayoutról bővebben a dokumentációban lehet olvasni.

A **3. sor** is egy `StackLayout` vezérlőt tartalmazzon: - Ennek az `Orientation`-je legyen `Vertical`, a `HorizontalOptions` értéke pedig `Start`. - Továbbá tartalmazza az alábbi elemeket: - Egy `Label`-t a *"Képek száma:"* felirattal. - Egy `Label`-t a képek számosságának, amely adatkötéssel az `Images.Count` értékét jeleníti meg. - Egy `ProgressBar`-t az alábbi értékekkel: - a `Progress` tulajdonság legyen bindolva a nézetmodell `Progress` tulajdonságára, - az `InVisible` tulajdonság legyen bindolva a nézetmodell `IsDownloading` tulajdonságára, - a `WidthRequest` értéke legyen 150.

> *Tipp:* Hova tűnt a designer nézet? A Visual Studio 2022-ben nem támogatja a designer nézetet MAUI alkalmazások fejlesztése esetén. Cserébe a Visual Studio biztosít *Hot reload* újratöltés lehetőséget, sőt amennyiben futtatás közben módosítjuk az XAML fájl-t azt az alkalmazás real time frissíti.

7 Vezérlés

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

1. Az `AppShell.xaml.cs` osztály konstruktorában hozzunk létre egy új `MainViewModel` példányt.
2. A `BindingContext` propertynek állítsuk be a létrehozott `MainViewModel`-t.
3. Ne feledkezzünk meg feliratkozni a nézetmodell `ErrorOccured` eseményére és jelenítsük meg a kért hibaüzenetet egy felugró ablakban a `DisplayAlert` használatával.

8 Képmegtekintő megvalósítása *(opcionális)*

8.1 Nézetmodell

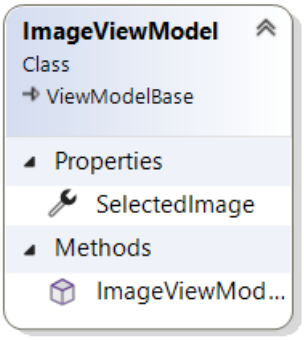

Figure 9: `ImageViewModel` osztálydiagramja

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

Az osztályba vegyünk fel egy *SelectedImage* nevű `ImageSource` típusú tulajdonságot. Hozzuk létre az osztály konstruktorát, amely inicializálja az *SelectedImage* tulajdonságot.

8.2 Nézet

A *View* mappára kattintva az egér jobb gombjával, az *Add New Item* menüben adjunk hozzá egy új *.NET MAUI ContentPage (XAML)* elemet. A neve legyen például *ImagePage*.

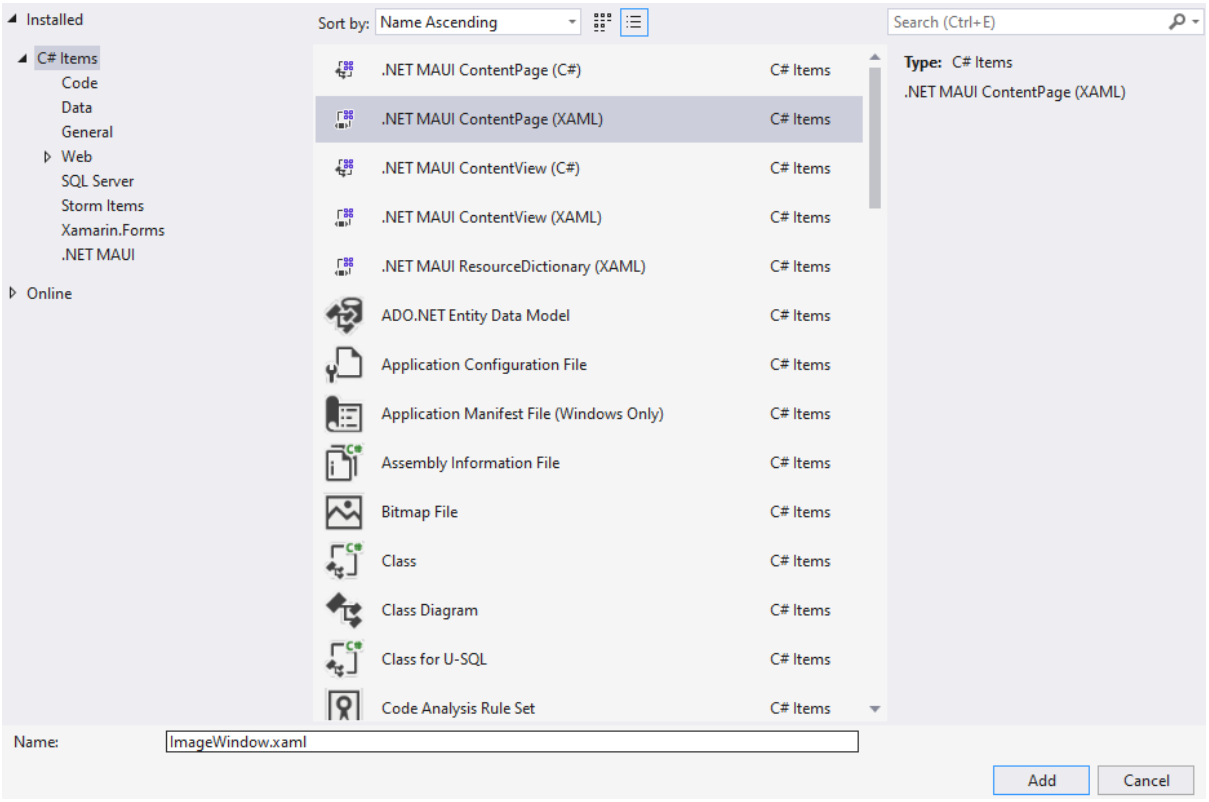

Figure 10: Új nézet hozzáadása a projekthez

A létrehozott képernyőről töröljük az automatikusan generált vezérlőket, és helyette hozzunk létre egy `Image` vezérlőt. Állítsuk be 500-ra a *MaximumHeightRequest* és a *MaximumWidthRequest* paramétereket.

A lapra felhelyezett képnek (`Image`) a `Source` tulajdonságát adatkötéssel a `ImageViewModel`-ben található `SelectedImage` tulajdonságra kössön.

```
Source="{Binding SelectedImage}"
```

Írjuk át a lap címét (`Title` property) például a *"Képmegjelenítő"* kifejezésre.

> *Megjegyzés:* A vissza gombot nem kell megvalósítanunk, mert a MAUI alapértelmezetten megvalósítja nekünk.

8.3 Kép kiválasztása

A `MainViewModel` osztályt egészítsük ki az alábbiakkal:

- Vegyünk fel egy `DelegateCommand` típusú `ImageSelectedCommand` parancsot.
- Vegyünk fel egy új eseményt `ImageSelected` néven.
- Az `ImageSelectedCommand`-ot valósítsuk meg a konstruktorban a `DownloadCommand`-hoz hasonlóan. A megvalósításakor váltsunk ki egy `ImageSelected` eseményt, úgy, hogy a küldő objektum legyen az aktuális objektum, az eseményargumentum pedig legyen az `ImageSource`. Vizsgáljuk meg, hogy a megkapott paraméter valóban `ImageSource` típusú-e.

A `MainPage` nézetet is egészítsük ki. Vegyük észre, hogy ha megpróbáljuk kötni az ImageButton 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, ha a kötésen `RelativeSource`-ot beállítjuk, például arra, hogy `Page` típusú őst keressen. Ilyen esetben viszont a `BindingContext.ImageSelectedCommand`-hoz kell kötni, mert a `Page` típusú vezérlőtől így érhető el a kívánt parancs:

```
<ScrollView Grid.Row="1">
    <FlexLayout BindableLayout.ItemsSource="{Binding Images}"
                Wrap="Wrap" Direction="Row" JustifyContent="SpaceEvenly">
        <BindableLayout.ItemTemplate>
            <DataTemplate>
                <ImageButton Source="{Binding}"
                    WidthRequest="100" HeightRequest="100"
                    Command="{Binding BindingContext.ImageSelectedCommand,
                              Source={RelativeSource AncestorType={x:Type Page}}}"
                    CommandParameter="{Binding}"
                    Aspect="Fill" Margin="5" />
            </DataTemplate>
        </BindableLayout.ItemTemplate>
    </FlexLayout>
</ScrollView>
```

Egészítsük ki az `AppShell.xaml.cs` osztályt az alábbiakkal:

1. A konstruktorban iratkozzunk fel a `MainViewModelhez` tartozó `ImageSelected` eseményre. Ehhez készítsünk egy eseménykezelő eljárást is.
2. Az eseménykezelő eljárásban készítsünk egy új `ImagePage` objektumot. Az elkészített objektum `BindingContext` paraméterének adjunk át egy újonnan példányosított `ImageViewModel`-t.
3. Navigáljunk át az elkészített lapra a `Navigation` osztályban található `PushAsync` eljárással:

   ```
   await Navigation.PushAsync(imageView);
   ```

9 Letöltés megszakíthatósága *(opcionális)*

9.1 `MainViewModel` kiegészítése

Vegyünk fel egy *getter-only* publikus string tulajdonságot *DownloadButtonLabel* néven. Valósítsuk meg, hogy amennyiben letöltés van folyamatban (`_isDwonloading`), akkor a *"Letöltés megszakítása"* kifejezést adja vissza, minden egyéb esetben pedig a *"Képek betöltése"* értéket.

Az `IsDownloading` tulajdonság setterében gondoskodjunk róla, hogy ha az érték változik, frissüljön a letöltés gombhoz tartozó szöveg is.

```
OnPropertyChanged(nameof(DownloadButtonLabel));
```

A `DownloadCommand` megvalósítását egészítsük ki, hogy amennyiben jelenleg fut letöltés, hívjuk meg a modellen a `CancelLoad()` metódust.

9.2 MainPage kiegészítése

A lapon a képek letöltése gombnak a szövegét ne égessük be, hanem kössük rá a korábban a nézetmodellben létrehozott `DownloadButtonLabel`-t.