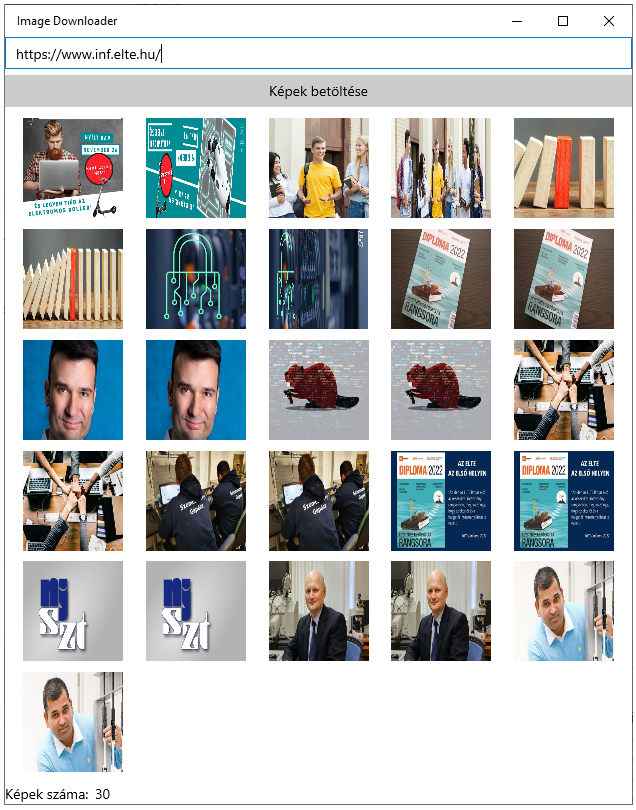
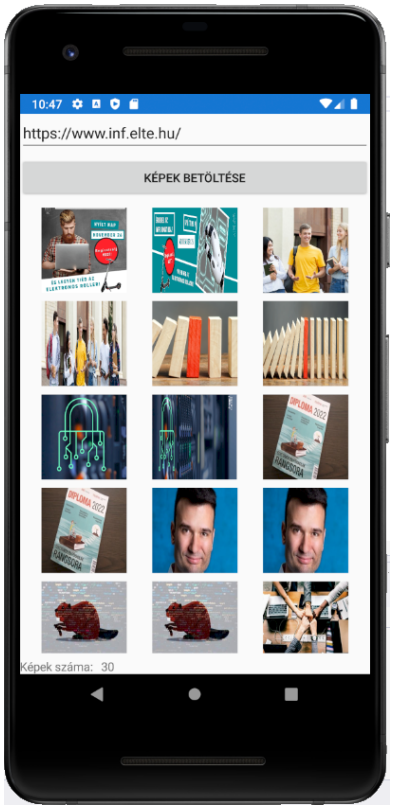
Eseményvezérelt alkalmazások: 12. gyakorlat

A feladat a *7. gyakorlaton* elkészített aszinkron kép letöltő alkalmazás (`ImageDownloader`) többplatformos mobilalkalmazás változatának elkészítése, amely segítségével egyszerűen listázhatjuk egy weboldalon megjelenő képeket.





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

1 Projekt létrehozása

Készítsünk Visual Studioban egy új *Mobile App (Xamarin.Forms)* projektet, a neve lehet például *ImageDownloader*. Az üres (*Blank*) sablont válasszuk, valamint az Android és az UWP achitektúrákat jelöljük be.

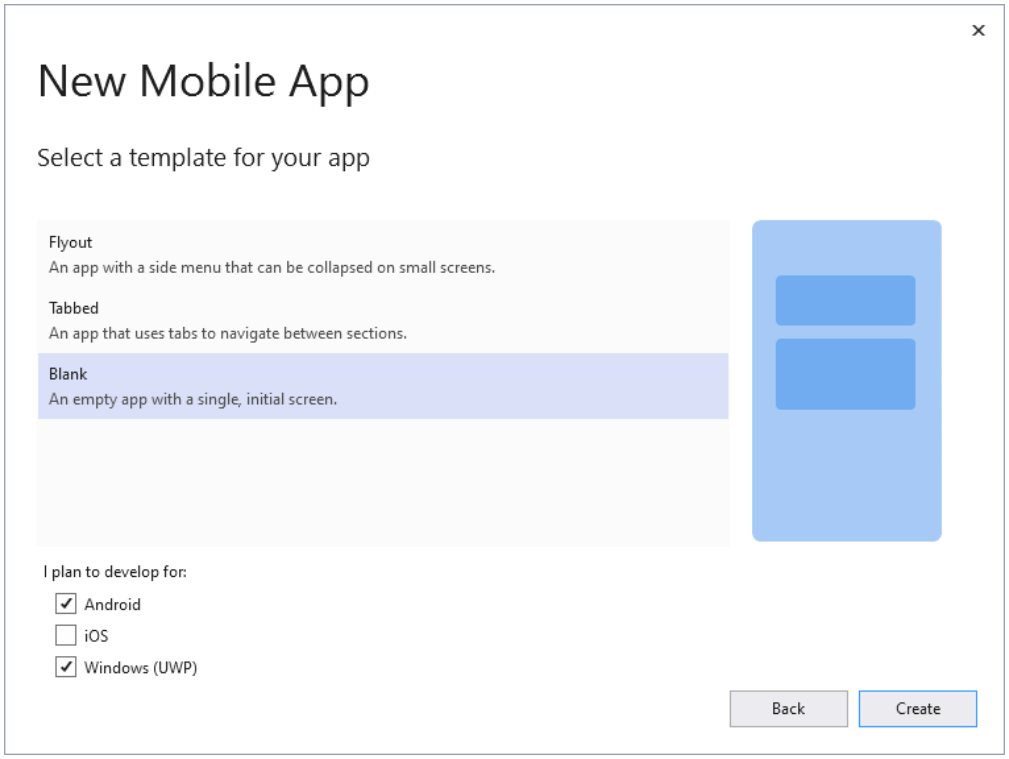


Figure 1: Template választása mobilalkalmazáshoz

Adjunk hozzá egy `Model`, `ViewModel` és egy `View` könyvtárat, ahol a rétegeknek megfelelő osztályokat fogjuk elhelyezni.

Adjuk hozzá a projekthez a *HtmlAgilityPack* NuGet csomagot, amellyel a HTML tartalom parszolását végezzük el, a 7. gyakorlaton már ismertetett módon.

2 Modell refaktorálása

A modell réteg továbbra is két fő osztályból fog állni (`WebImage` és `WebPage`), azonban néhány refaktorálást végezzünk el rajtuk.

2.1 A *WebImage* osztály

A felhasznált `System.Drawing.Image` típus nem platformfüggetlen, alapvetően csak Windows alapú operációs rendszereken támogatott. Választhatnánk helyette valamilyen cross-platform képfeldolgozó könyvtárat (pl. *ImageSharp*), de mivel valójában nincsen képfeldolgozásra szükségünk, ebben az alkalmazásban megfelelő az is, ha a képeket egy bájt tömbbel reprezentáljuk (`byte[]`), és a megjelenítést a Xamarinra bízzuk majd.

Cseréljük le a `WebImage` osztály `Image` tulajdonságát egy `byte[] Data` tulajdonságra (módosítsuk a vonatkozó adattagokat is). A `DownloadAsync()` eljárásban is ennek megfelelően olvassuk be a képfájlt a megadott URL-ről:

```
HttpClient client = new HttpClient();
byte[] image = await client.GetByteArrayAsync(url);
```

2.2 A *WebPage* osztály

A `WebPage` osztályt eseményeit alakítsuk át úgy, hogy konvencionálisan egy `EventArgs` osztályból leszármaztatott eseményargumentum típusú értéket adjon át: `LoadProgressEventArgs` és `ImageLoadedEventArgs`.

A betöltött képek helyességét (azaz, hogy értelmezhetőek-e képként) már nem végzi el a `WebImage` osztály egy `System.Drawing.Image` konstruálásával, így más módon szükséges ezt megoldanunk. Adjunk hozzá egy új `ImageLoading` eseményt is az osztályhoz, amelyet egy kép betöltésekor, de még az `Images` gyűjteményhez adás előtt váltunk ki. A hozzá tartozó `ImageLoadingEventArgs` típus a betöltendő kép mellett tartalmazzon egy `IsValid` logikai értéket is, amely alapértelmezetten legyen igaz, de egy eseménykezelő majd hamisra állíthat.

A `LoadImageAsync()` eljárásban csak akkor vegyük fel a betöltött képet, ha az `IsValid` értéke igaz maradt:

```
var image = await WebImage.DownloadAsync(imageUrl);
var args = new ImageLoadingEventArgs(image);
ImageLoading?.Invoke(this, args);
if (args.IsValid)
{
    _images.Add(image);
    ImageLoaded?.Invoke(this, args);
}
```

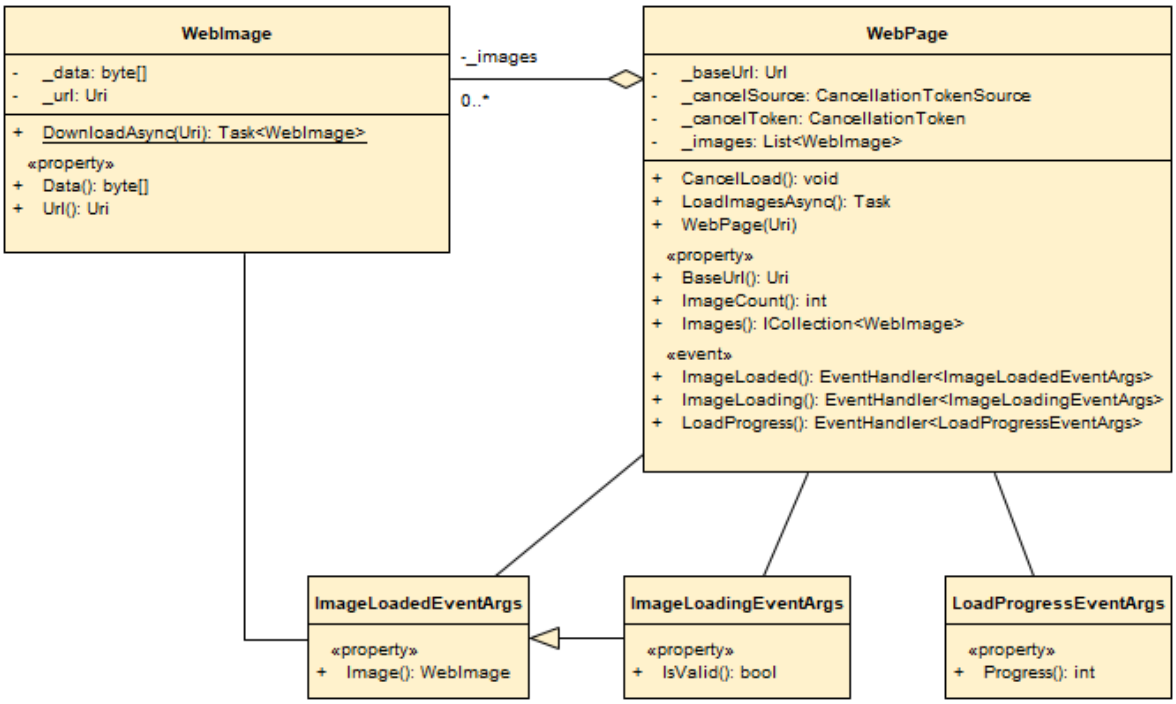


Figure 2: A modell refaktorált osztálydiagramja

Megjegyzés: robusztusabb, alternatív megoldásként a `WebPage` osztályba befecskendezhetnénk egy kép validátor interfészt, amelynek konkrét, platform függő megoldásai implementálhatóak lennének.

3 Nézetmodell

A nézetmodell rétegben használjuk fel a korábbi gyakorlatokon (és előadáson) már bevezetett `ViewModelBase` és `DelegeteCommand` ősosztályokat. Az egszerűség kedvéért ezek forráskódja:

```
public abstract class ViewModelBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
```

```
    public ViewModelBase() { }

    protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

A `DelegeteCommand` osztályt egészítsük ki egy `RaiseCanExecuteChanged()` eljárással, amellyel kívülről is kiváltható a `CanExecuteChanged` eseménye.

```
public class DelegateCommand : ICommand
{
    private readonly Action<Object> _execute;
    private readonly Func<Object, Boolean> _canExecute;

  public event EventHandler CanExecuteChanged;

    public DelegateCommand(Action<Object> execute, Func<Object,Boolean> canExecute = null)
    {
        if (execute == null)
        {
            throw new ArgumentNullException(nameof(execute));
        }

        _execute = execute;
        _canExecute = canExecute;
    }

    public bool CanExecute(object parameter)
    {
        return _canExecute == null ? true : _canExecute(parameter);
    }

    public void Execute(object parameter)
    {
        if (!CanExecute(parameter))
        {
            throw new InvalidOperationException("Command execution is disabled");
        }
        _execute(parameter);
    }

  public void RaiseCanExecuteChanged()
  {
    CanExecuteChanged?.Invoke(this, EventArgs.Empty);
  }
}
```

3.1 A *MainViewModel* osztály

Hozzuk létre a `MainViewModel` osztályt és válasszuk le a korábbi, kétrétegű MV architektúrában készült Windows Forms alkalmazás `MainForm` nézetének nézetmodelljét.

Vegyük fel a következő **tulajdonságokat** (és a szükség szerinti segéd adattagokat):

- `IsDownloading`: logikai érték, amely tárolja, hogy éppen folyamatban van-e letöltés. Értékváltozásáról értesítse a nézetet.
- `Progress`: lebegőpontos érték (0 és 1 között), amelyet a folyamatjelző státuszához használunk majd. Értékváltozásáról értesítse a nézetet.
- `Images`: a megjelenítendő képet tartalmazó `ObservableCollection`. A gyűjteményben `Xamarin.Forms.ImageSource` objektumokat tároljunk.

A következő publikus **eljárásokra** lesz szükségünk a képek letöltéséhez és letöltés megszakíthatóvá tételéhez:

- `LoadAsync(Uri url)`: aszinkron eljárás, amely az `Images` kollekciót kiüríti, majd példányosít a paraméterként kapott URL-lel egy `WebPage` modellt, feliratkozik az eseményeire és aszinkron várakozással elindítja a képek letöltését. A folyamat előtt és után az `IsDownloading` értékét állítsuk be megfelelően. A modell eseményeinek kezelésével kicsit később foglalkozunk.
- `CancelLoad()`: megszakítja a modellen a képletöltést, ha az még fut.

Végezetül adjunk egy `DelegateCommand` típusú `DownloadCommand`-ot a nézetmodellhez, amellyel képek letöltése parancskötésen keresztül elindítható lesz. A parancs akcióként a `LoadAsync()` eljárást hívja meg, a parancs paramétereként kapott URL-lel. A parancs végrehajthatósága az `IsDownloading` tulajdonsághoz legyen kötve.

Módosítsuk az `IsDownloading` tulajdonság *setter* ágát úgy, hogy értékváltozás esetén a `DownloadCommand` parancs `CanExecuteChanged` eseményét is kiváltsa.

3.1.1 Az *ImageLoading* esemény kezelése

Ellenőrizzük, hogy a betöltött kép `jpg`, `png` vagy `gif` kiterjesztéssel rendelkezik-e, továbbá a kapott bájt tömbből `ImageSource` típusú objektum konstruálható-e. Ennek megfelelően állítsuk be az `IsValid` tulajdonság értékét az eseményargumentumban.

```
private void OnImageLoading(object sender, ImageLoadingEventArgs e)
{
    try
    {
        if (!e.Image.Url.LocalPath.EndsWith(".jpg") &&
            !e.Image.Url.LocalPath.EndsWith(".jpeg") &&
            !e.Image.Url.LocalPath.EndsWith(".png") &&
            !e.Image.Url.LocalPath.EndsWith(".gif"))
        {
            e.IsValid = false;
            return;
        }
        ImageSource.FromStream(() => new MemoryStream(e.Image.Data));
    }
    catch
    {
        e.IsValid = false;
    }
}
```

3.1.2 Az *ImageLoaded* esemény kezelése

Adjuk az `Images` kollekcióhoz a `ImageSource` objektumba betöltött bájt tömböt.

3.1.3 A *LoadProgress* esemény kezelése

Állítsuk be a `Progress` tulajdonság értékét az eseményben kapott értéknek megfelelően.

4 Nézet

A nézet réteg egy lapból (`MainPage`) fog állni, amelyen a következő vezérlőket helyezzünk el:

- egy `Entry`-t az URL cím bevitelére;
- egy `Button`-t a képek letöltésének elindítására;
- egy `FlexLayout`-ot a képek sorfolytonos, automatikusan tördelt megjelenítésére (`Image` vezérlőket helyezünk majd rá dinamikusan);
- egy `Label`-t és egy `ProgressBar`-t a letöltött képek számának és letöltési folyamat állapotának kijelzésére.

A vezérlőket egy `Grid`, valamint `StackLayout`-ok segítségével rendezhetjük el a kívánt módon. A `FlexLayout` vezérlőt `ScrollView`-ba helyezbe tehetjük a tartalmát görgethetővé.

4.1 Letöltés gomb parancskötése

A letöltés gombot kössük a nézetmodell `DownloadCommand` parancsához. A `CommandParameter` értéke a szöveges beviteli mező (`Entry`) értéke legyen.

4.2 Képek dinamikus megjelenítése

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

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

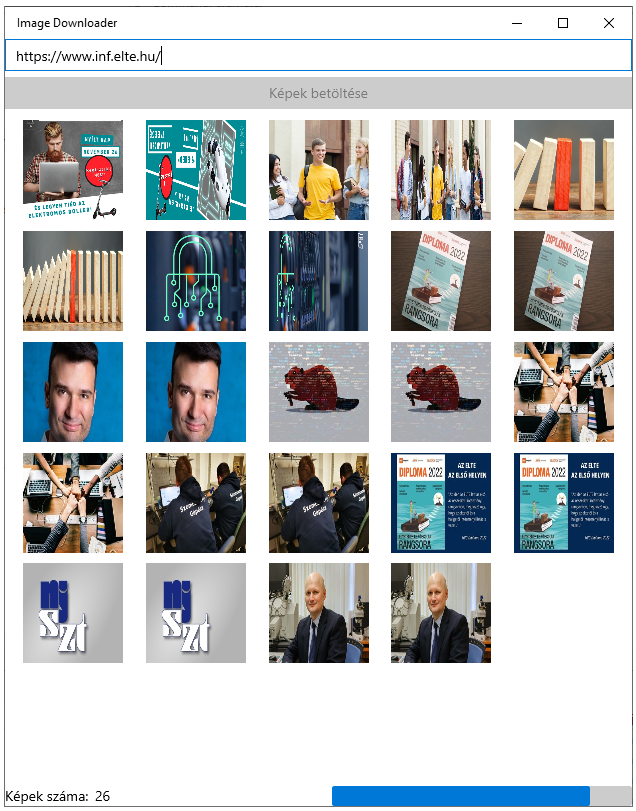


Figure 3: Az UWP alkalmazás letöltés közben

Megjegyzés: A `FlexLayout` a CSS3-ból is ismert *flex-box*-okhoz hasonlóan működik, a nevét is onnan kapta.

5 Alkalmazás környezet réteg

Az `App` osztályunk konstruktorában példányosítsunk egy `MainViewModel` nézetmodellt és egy `MainPage` nézetet. A nézetbe fecskendezzük be a nézetmodellt (`BindingContext`).

5.1 Mobil alkalmazások életciklusa

A mobil alkalmazások életciklusa eltér az asztali alkalmazásokétól, szükséges gondoskodnunk az alkalmazás felfüggesztett állapotba kerülésekor, valamint az aktív állapotba visszatérésekor végrehajtandó tevékenységekről.

- Amikor a képletöltő alkalmazás felfüggesztett állapotba kerül (`OnSleep`), szakítsuk meg a képletöltést, ha éppen folyamatban van, továbbá ez esetben mentsük el az utoljára használt URL-t. A nézetmodellt töröljük a hatékony erőforrás gazdálkodás érdekében.
- Amikor a képletöltő alkalmazás folytatásakor (`OnResume`) példányosítsunk egy új nézetmodellt, amelyet átadhatunk a nézetnek. Amennyiben volt elmentett URL, indítuk el újra a képek letöltését.

Tipp: az utoljára használt URL-t elmenthetjük az `Application.Properties` dictionary-be, ez az alkalmazás gyorsítótáraként (*cache*) tartósan mentésre kerül.