Eseményvezérelt alkalmazások: 8. gyakorlat

A munkafüzet bevezet a *Windows Presentation Foundation* (WPF) alapú asztali grafikus alkalmazás fejleszésbe. Hálózati kommunikációt használó alkalmazást készítve ügyeljünk a felhasználói felület reszponzivitására, a költséges háttér műveletek aszinkron megvalósítására.

A feladat egy **aszinkron kép letöltő alkalmazás elkészítése**, amely segítségével egyszerűen listázhatjuk egy weboldalon megjelenő képeket, majd azokat egy külön ablakban megnyitva letölthetjük.

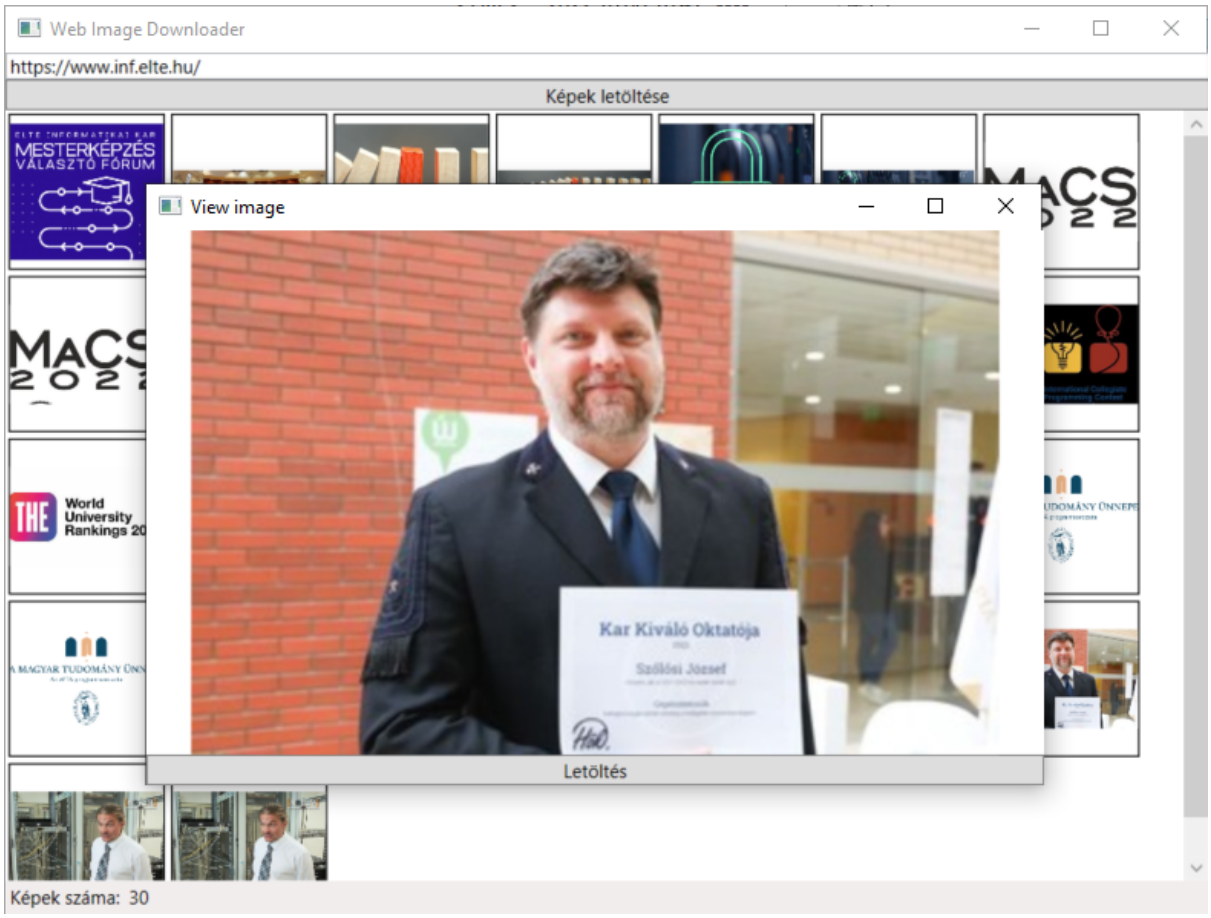

Figure 1: Az alkalmazás megjelenése

Az alkalmazást *Windows Presentation Foundation* keretrendszerrel, kétrétegű (modell-nézet) architektúrában, eseményvezérelt paradigma alapján valósítjuk meg.

1 Projekt létrehozása ^{KM}

Készítsünk Visual Studioban egy új *WPF Application* projektet, a neve lehet például *ImageDownloader*. Adjunk hozzá egy *Model* és egy *View* könyvtárat, ahol a rétegeknek megfelelő osztályokat fogjuk elhelyezni.

Adjuk hozzá a projekthez a *HtmlAgilityPack* NuGet csomagot, amellyel a HTML tartalom parszolását végezhetjük majd el magas absztrakciós szinten.

2 Modell ^{KM}

A modell réteg két osztályból fog állni:

- `WebImage`: egy webes képet reprezentál, tárolja annak elérési útvonalát az `Url` tulajdonságában (típusa: `System.Uri`) és magát a képet a `Data` tulajdonságában bájt tömbként (típusa: `byte[]`).
- `WebPage`: egy weboldalról betöltött képek gyűjteményét reprezentálja. Tárolja a weboldal címét a `BaseUrl` tulajdonságában (típusa: `System.Uri`) és a betöltött képeket az `Images` tulajdonságában (típusa: `ICollection<WebImage>`, a tényleges reprezentációnak `List<WebImage>` választható).

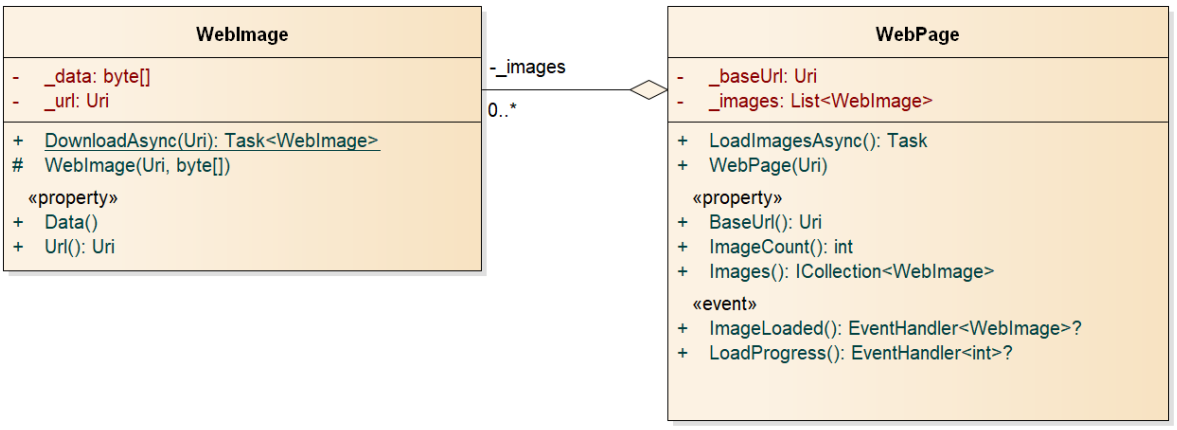

Figure 2: A modell osztálydiagramja

Készítsük el az alkalmazás modell rétegét a leírtak és az osztálydiagram alapján.

2.1 A *WebImage* osztály

A `WebImage` osztályban a két tulajdonság (`Url` és `Data`) mellett hozzunk létre egy triviális konstruktort, amely inicializálja a két tulajdonságot.

Mivel a `WebImage` példányokat a várható erőforrásigény (kép letöltésre az Internetről) miatt szeretnénk aszinkron módon előállítani, definiáljunk az osztályban `static async Task<WebImage> DownloadAsnyc(Uri url)` statikus metódust, amely a paraméterben átvett URL-ről a képet aszinkron módon letölti és egy új `WebImage` példánnyal tér vissza.

Ehhez előbb ellenőrizzük, hogy a paraméter (`url`) ki van-e töltve (nem `null` érték), valamint abszolút elérési útvonalat tartalmaz-e (`IsAbsoluteUri`)! Dobjunk kivételt, amennyiben nem.

A kép letöltéséhez egy HTTP kérés végrehajtására van szükség, amelyhez a `System.Net.Http.HttpClient` osztályt használhatjuk:

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

Megjegyzés: azért készítünk egy külön statikus `DownloadAsnyc()` metódust, mivel a `WebImage` osztály konstruktora nem lehet aszinkron, hiszen a tényleges megkonstruált objektumot kell visszaadnia, ami így nem lehet egy `Task`. A `DownloadAsnyc()` eljárásunk így valójában a *factory* tervezési mintát valósítja meg. A jelentősebb tervezési mintákról a Szoftvertechnológia kurzus keretében lesz majd részletesen szó. Az osztály konstruktorát akár `private` vagy `protected` láthatóságúvá is tehetjük, így (kívülről) csak a `DownloadAsnyc()` eljárással készíthető új `WebImage` objektum.

2.2 A *WebPage* osztály

A `WebPage` osztály konstruktora csak a `BaseUrl` tulajdonság értéket állítsa be; itt is ellenőrizzük, hogy a paraméter elérési útvonal ki van-e töltve és abszolút-e! A képek tényleges betöltését a `async Task LoadImagesAsync()` aszinkron metódus végezze.

2.2.1 HTML tartalom beltöltése

Kérjük le a megadott weboldalt (HTML sztringként), amelyből majd a képeket szeretnénk kikeresni:

```
HttpClient client = new HttpClient();
var response = await client.GetAsync(BaseUrl); // HttpResponseMessage
var content = await response.Content.ReadAsStringAsync(); // string
```

Használhatjuk a `client.GetStringAsync(BaseUrl)` eljárást is a tartalom egy utasítással történő letöltéséhez.

A HTML tartalomban keressük az `<img>` elemeket. Az egyszerűség kedvéért a dinamikusan (pl. JavaScript-tel) betöltött képekkel nem fogunk foglalkozni. A `HtmlAgilityPack` NuGet csomagot használva hozzunk létre egy új `HtmlDocument` típusú objektumot és töltsük be a letöltött HTML tartalmat:

```
HtmlAgilityPack.HtmlDocument doc = new HtmlAgilityPack.HtmlDocument();
doc.LoadHtml(content);
```

2.2.2 Képek keresése és feldolgozása

Az osztály elvégzi a HTML tartalom parszolását, így már egyszerűen kiválogathatjuk az `<img>` elemeket:

```
var nodes = doc.DocumentNode.SelectNodes("//img");
```

Megjegyzés: a keresett elemeket (`//img`) az XPath lekérdező nyelv segítségével adjuk meg.

Amennyiben a HTML sztring parszolása sikertelen volt (pl. nem HTML tartalom volt a megadott URL-en), a `nodes` változó értéke `null` lesz. Ennek ellenőrzése után végigiterálhatunk a gyűjtemény összes elemén. A képek forrása az `<img>` elemek `src` attribútumában található, így ellenőrizzük, hogy ez megvan-e adva (`node.Attributes.Contains("src")`)! Amennyiben nem, folytassuk a feldolgozást a következő elemmel:

```
foreach (var node in nodes)
{
    if (!node.Attributes.Contains("src"))
        continue;

    // ...
}
```

2.2.3 Abszolút URL-ek előállítása

A képek elérési útvonala abszolút és relatív formában is szerepelhet az `<img>` elemek `src` attribútumában.

- Abszolút URL: `https://inf.elte.hu/mappa/kep.jpg`
- Relatív URL: `mappa/kep.jpg`

Állítsunk elő egy garantáltan abszolút elérési útvonalat a `System.Uri` osztály 2 paraméteres konstruktorával, amely egy abszolút elérési útvonalat és egy ahhoz képest relatív elérési útvonalat vár:

```
Uri imageUrl = new Uri(node.Attributes["src"].Value, UriKind.RelativeOrAbsolute);
if (!imageUrl.IsAbsoluteUri)
    imageUrl = new Uri(BaseUrl, imageUrl);
```

2.2.4 Képek letöltése (`WebImage` példányosítás)

Így már létrehozhatjuk a `WebImage` példányt és hozzáadhatjuk az osztályban tárolt listához. A `WebImage.DownloadAsync()` metódus kivételt dobhat (a benne lévő `HttpClient` objektum), amennyiben a megadott URL-en mégsem kép volt, vagy olyan formátum, amely nem támogatott. Ezeket az elemeket egyszerűen ugorjuk át, a kivétel elkapásával.

```
try
{
    var image = await WebImage.DownloadAsync(imageUrl);
    _images.Add(image);
}
catch
{
    // ignored
}
```

2.2.5 Értesítés eseményekkel

A `WebPage` osztályt végezetül egészítsük ki 2 eseménnyel:

- `ImageLoaded`: egy új kép sikeres letöltésekor váltódik ki és átadja letöltött `WebImage` példányt.
- `LoadProgress`: százalékosan jelzi (0 és 100 közötti egész értékkel), hogy hol tart a képek letöltési folyamata. A `nodes.Count` révén előre ismerjük hány `<img>` elemet is kell majd feldolgoznunk.

A `nodes` változót bejáró ciklusban váltsuk ki a megfelelő helyeken az `ImageLoaded` és a `LoadProgress` eseményeket.

> *Megjegyzés:* A *Microsoft* tervezési konvencióját akkor követjük megfelelően, ha az események argumentumait az `EventArgs` osztály egy leszármaztatott példányában adjuk át.

3 Nézet *EM*

A nézet réteg 2 osztályból áll: a főablakból (`MainWindow`) és a kép megjelenítő / letöltő ablakból (`ImageWindow`).

3.1 A *MainWindow* nézet

A főablakon az alapértelmezetten generált `Grid` panelt cseréljük le egy `DockPanel`-re, amelyben 4 vezérlőt helyezzünk el:

- egy `TextBox`-ot az URL cím bevitelére;
- egy `Button`-t a képek letöltésének elindítására;
- egy `StatusBar`-t az állapotsornak;
- egy `WrapPanel`-t a képek megjelenítésére (`Image` vezérlőket helyezünk majd rá dinamikusan).

Az egyes vezérlők `DockPanel.Dock` tulajdonságát módosítva dinamikusan kitöltik az ablakot. Ehhez a szövegdobozt és a nyomógombot felfele (`Top`) dockoljuk, míg az állapotsort lefele (`Bottom`). A `WrapPanel`-t nem kell `Dock` tulajdonságát nem kell megadnunk, így lesz középre helyezve.

Az állapotsor elkészítéshez a Visual Studio tervező felületén egyszerű *drag-and-drop* módszerrel húzzuk a vezérlőket a `StatusBar` vezérlőre. (Alternatíva: a *Properties* ablakban az *Items* tulajdonságot szerkesszük.) Vegyünk fel az állapotsorra egy `TextBlock`-t a letöltött képek számának megjelenítésére és egy `ProgressBar`-t a folyamat előrehaladtának jelzésére.

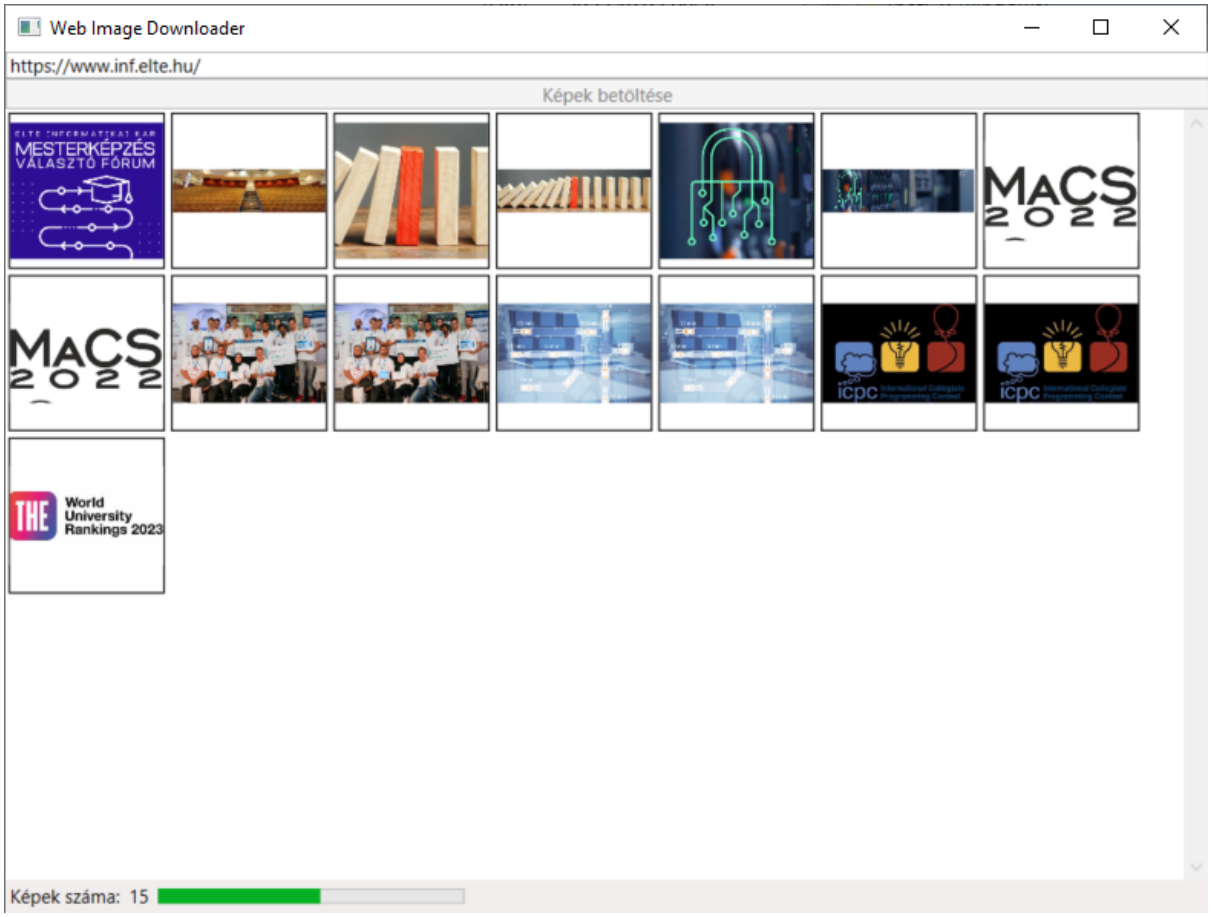

Figure 3: MainWindow nézet

A `MainWindow` nézet a reprezentációjában aggregáljon egy `WebPage` modell példányt (`_model` adattag). A *Képek letöltése* gomb `Click` eseményéhez kapcsoljunk egy eseménykezelőt, és hajtsuk végre a következő teendőket:

1. Az állapotsorban a képek számát állítsuk 0-ra. A folyamatjelző (*progress bar*) megjelenítését engedélyezzük (`Visibility` tulajdonság) és szintén állítsuk értékét 0-ra. A *Letöltés* gombot tiltsuk le (`IsEnabled` tulajdonság).
2. A `WrapPanel` elemeit (korábban betöltött képek) töröljük, a `Children` kollekció kiürítésével.
3. Példányosítsunk egy új `WebPage` modellt a felületen megadott URL-lel a `_model` adattagba.
4. Iratkozzunk fel a modell `ImageLoaded` és `LoadProgress` eseményére 1-1 eseménykezelő eljárással.
5. Hívjuk meg a modell `LoadImagesAsync()` eljárását és aszinkron módon várakozzunk az eredményére (`await`).
6. Az eredményét bevárását követően rejtsük el a folyamatjelzőt. A *Letöltés* gombot engedélyezzük.

3.1.1 Az *ImageLoaded* esemény kezelése

Az esemény minden kiváltásakor a modell sikeresen letöltött egy új képet, ennek megfelelően a nézeten is megjeleníthetjük. Ilyen módon a nézet is folyamatosan frissül, ami jobb felhasználói élményt nyújt, ha a weboldal sok képet tartalmazott vagy lassú a hálózati kapcsolat. A következő feladatokat kell elvégeznünk:

1. Készítsünk egy új memóriabeli képet egy `BitmapImage` objektum létrehozásával. A modellben bájt tömbként tárolt képet a `StreamSource` tulajdonság beállításával adhatjuk meg, az alábbi módon:

```
var bitmap = new BitmapImage();
bitmap.BeginInit();
bitmap.StreamSource = new MemoryStream(e.Data); // e: paraméter WebImage
bitmap.EndInit();
```

Megjegyzés: a `BitmapImage` típus egyszer inicializálható, az inicializációs lépéseket a `BeginInit` és `EndInit` eljárások közé kell foglalni. A későbbi *property* módosításoknak már nincsen hatása.

2. Példányosítsunk egy új `Image` vezérlőt.
   - Állítsuk a méretét (`Width` és `Height`) *100px*-re.
   - Állítsuk az adatforrását (`Source`) az előbb létrehozott bitmap képre.
3. Adjuk az `Image` vezérlőt a `WrapPanel`-hez (a `Children` kollekciójához).
4. Az állapotsorban a letöltött képek számlálóját növeljük 1-gyel.

Tipp: a képeknek keretet és közéjük térközt helyezhetünk, ha az `Image` vezérlőt egy `Border` vezérlőbe ágyazzuk, és azt adjuk a panelhez:

```
var border = new Border
{
   BorderThickness = new Thickness(1),
   BorderBrush = Brushes.Black,
   Margin = new Thickness(2)
};
border.Child = image;

picturePanel.Children.Add(border);
```

3.1.2 A *LoadProgress* esemény kezelése

Módosítsuk az folyamatjelző (*progress bar*) értékét (`Value`) az eseményben kapott értéknek megfelelően.

3.2 Az *ImageWindow* nézet ^{OP}

Figure 4: ImageWindow nézet

A kép megjelenítő / letöltő-n helyezzünk el 2 vezérlőt:

- egy `Image` vezérlőt a kép megjelenítésére;
- egy `Button`-t a kép letöltéséhez.

Ezt az ablakot is egy `DockPanel` felülettel készítsük el, amelyben a nyomógombot lefelé dockoljuk (`Bottom`), a képet pedig középre. (Ehhez egyszerűen nem kell a `DockPanel.Dock` tulajdonságát kitölteni.)

Az `ImageWindow` osztály konstruktora várjon egy `BitmapImage` objektumot, ez lesz a megjelenítendő és letölthető kép. A konstruktor töltse is be a képet `Image` vezérlő `Source` tulajdonságába.

A *Letöltés* gomb `Click` eseményéhez kapcsoljunk egy eseménykezelőt, és hajtsuk végre a következő teendőket:

1. Példányosítsunk egy új `SaveFileDialog` objektumot.
2. Inicializáljunk egy szűrőt, hogy csak a PNG képeket jelenítsük meg a dialógusablakban. További beállításokat is tetszés szerint megadhatunk:

```
SaveFileDialog saveDialog = new SaveFileDialog();
saveDialog.Filter = "PNG files (*.png)|*.png";
saveDialog.InitialDirectory = Environment.GetFolderPath(
   Environment.SpecialFolder.MyPictures);
saveDialog.RestoreDirectory = true;
```

3. Jelenítsük meg a dialógusablakot, és ellenőrizzük, hogy Mentés gombbal zárták-e be:

```
if (saveDialog.ShowDialog() == true)
{
    // ... save file to disc ...
}
```

4. Amennyiben igen, írjuk ki a megadott útvonalra a képet PNG állományként. Ehhez a `PngBitmapEncoder` típust fogjuk használni, amelyhez előbb hozzáadunk egy új *frame*-t, majd egy `FileStream`-be írjuk a képet. A `FileStream`-et mi nyitjuk meg a felhasználó által megadott útvonalra.

```
using (var fileStream = new FileStream(saveDialog.FileName, FileMode.Create))
{
  BitmapEncoder encoder = new PngBitmapEncoder();
  encoder.Frames.Add(BitmapFrame.Create(picture.Source as BitmapImage));
  encoder.Save(fileStream);
}
```

Egészítsük ki a `MainWindow` osztályban az aggregált `WebPage` modell `ImageLoaded` eseményének kezelését:

1. Az új `PictureBox` létrehozásakor iratkozzunk fel annak `Click` eseményére (pl. `ShowImage()` eseménykezelővel).
2. Az eseménykezelőben példányosítsunk és jelenítsünk meg egy új `ImageWindow` objektumot:

```
private void ShowImage(object sender, EventArgs e)
{
      if(sender is Image image && image.Source is BitmapImage bitmap)
      {
          ImageWindow window = new ImageWindow(bitmap);
          window.Owner = this;
          window.Show();
      }
}
```

> *Megjegyzés:* az `Owner` tulajdonság beállításával a főablak bezárásával az összes képmegjelenítő ablak is automatikusan bezárul.

4 Megszakítható letöltés ^{OP}

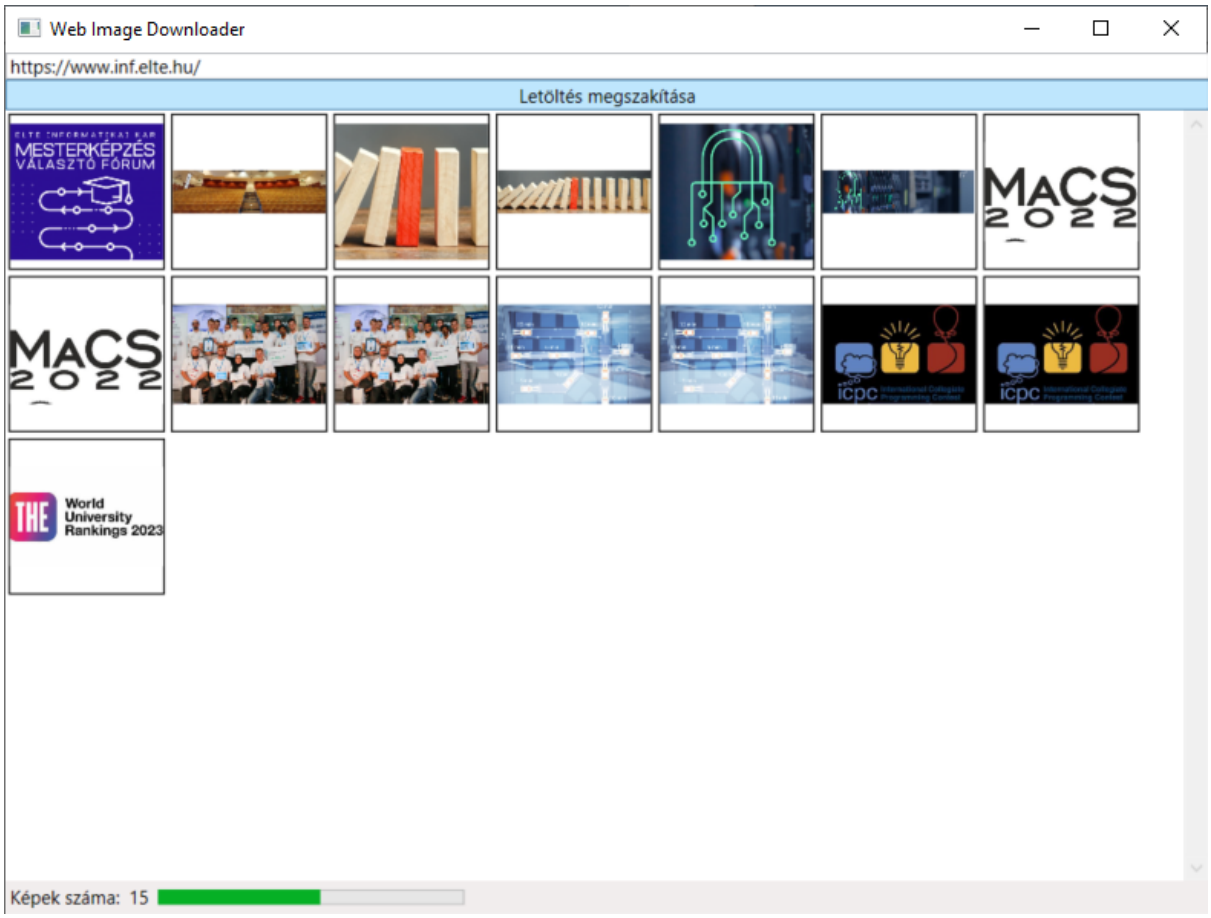

Figure 5: MainWindow nézet megszakítható letöltéssel

A képek letöltési folyamatát a főablakon tegyük megszakíthatóvá: a letöltést a felhasználó bármikor állíthassa le. Az addig már letöltött képek maradjanak meg.

4.1 A *WebPage* modell módosítása

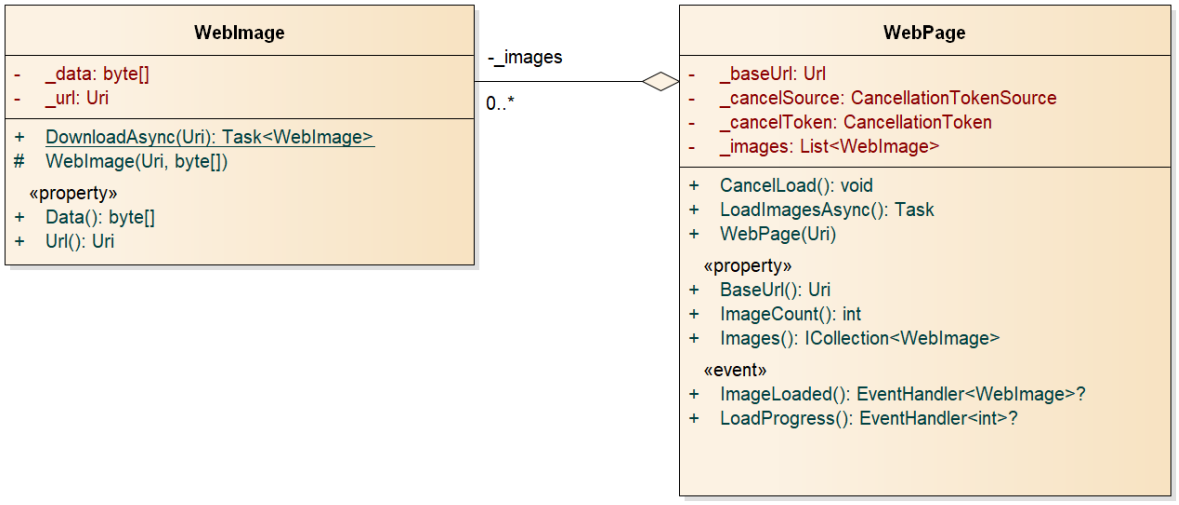

Figure 6: A megszakítható modell osztálydiagramja

A `WebPage` modellt adattagjait egészítsük ki egy `CancellationTokenSource` és egy `CancellationToken` objektummal. Adjunk az osztályhoz egy `CancelLoad()` metódust.

Az osztály konstruktorában készítsünk egy új `CancellationTokenSource` objektumot és kérjük le hozzá tartozó tokent:

```
_cancelSource = new CancellationTokenSource();
_cancelToken = _cancelSource.Token;
```

A `LoadImageAsync()` metódusban minden `<img>` elem feldolgozása előtt ellenőrizzük a tokenen, hogy nem kérték-e a folyamat megszakítását!

```
foreach (var node in nodes)
{
   if (_cancelToken.IsCancellationRequested)
       break;
   // ...
}
```

A weboldal HTML tartalmának letöltését is megszakíthatóvá tehetjük:

```
HttpClient client = new HttpClient();
var response = await client.GetAsync(BaseUrl, _cancelToken);
var content = await response.Content.ReadAsStringAsync();
```

Az újonnan az osztályhoz adott `CancelLoad()` metódusban a token forrásán keresztül jelezzük a megszakítás igényét:

```
public void CancelLoad()
{
    _cancelSource.Cancel();
}
```

4.2 A *MainWindow* nézet módosítása

A *Képek letöltése* gombra `Click` eseményéhez rendelt eseménykezelő metódusban a gombot ne tiltsuk le, hanem a feliratát módosítsuk *Letöltés megszakítására*, majd a letöltés végeztével vissza.

A gombra kattintáskor tudnunk kell, hogy a felhasználó letöltést indítani vagy megszakítani kíván-e. Ezt nem érdemes a gomb felirata alapján eldöntenünk, hanem két lehetőségünk van:

1. A nézet adattagjai közé vegyünk fel egy új logikai változót (`_isDownloading`), amely tárolja, hogy éppen folyamatban van-e letöltés, és ez alapján döntsünk az eseménykezelőben a teendőkről.
2. Két eseménykezelő metódusunk legyen (egy a letöltés indítására és egy a megszakítására) és váltakoztassuk melyikkel iratkoztunk fel a gomb `Click` eseményére.

Megszakítás esetén egyszerűen az aggregált `WebPage` modell `CancelLoad()` metódusát kell meghívnunk.