

Eseményvezérelt alkalmazások: 7. gyakorlat

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

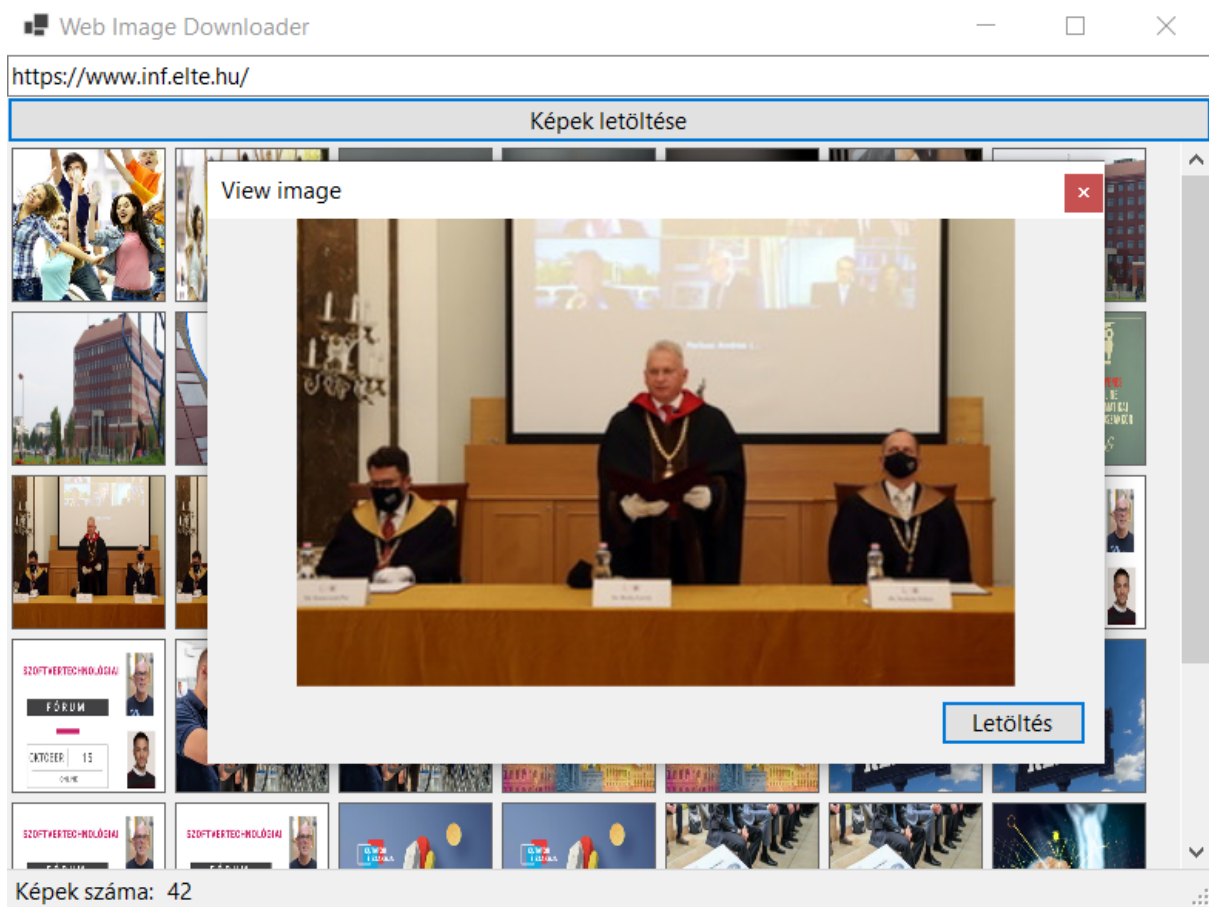


Figure 1: Az alkalmazás megjelenése

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

1 Projekt létrehozása

Készítsünk Visual Studioban egy új *Windows Forms App* 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

A modell réteg két osztályból fog állni:

- **WebImage**: egy webes képet reprezentál, tárolja annak elérési útvonalát az `Uri` tulajdonságában (típusa: `System.Uri`) és magát a képet az `Image` tulajdonságában (típusa: `System.Drawing.Image`).
- **WebPage**: egy weboldaltól 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).

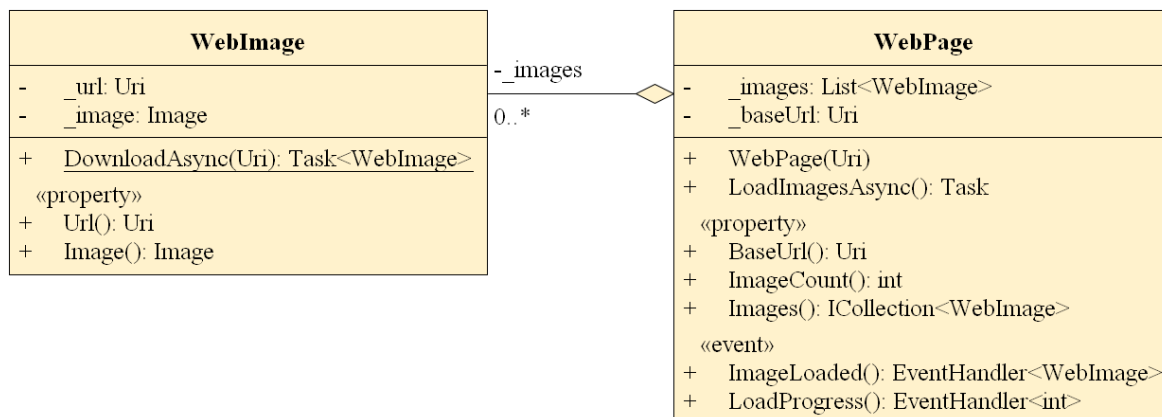


Figure 2: A modell osztálydiagramja

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

2.1 A *WebImage* osztály

A `WebImage` osztályban definiáljunk a `static async Task<WebImage> DownloadAsync(Uri url)` statikus metódust, amely a paraméterben átvett URL-ről a képet aszinkron módon letölti és egy új `WebImage` példánnyal tér vissza.

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();
Stream stream = await client.GetStreamAsync(url);
```

Az adatfolyamból már könnyen előállítható a kívánt `System.Drawing.Image` típusú objektum:

```
Image image = Image.FromStream(stream);
```

Megjegyzés: azért készítünk egy külön statikus `DownloadAsync()` metódust, mivel a `WebImage` osztály konstruktora nem lehet aszinkron, hiszen a tényleges megkonstruált objektumot kell visszaadnia, ami így nem lehet egy `Task`. A `DownloadAsync()` eljárásunk így valójában a *factory* tervezési mintát valósítja meg. A jelentősebb tervezési mintákról a Szoftvertchnológia kurzus keretében lesz majd részletesen szó.

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.

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
```

A HTML tartalomban keressük az 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 töltjük be a letöltött HTML tartalmat:

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

Az osztály elvégzi a HTML tartalom parszolását, így már egyszerűen kiválogathatjuk az 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 elemek src attribútumában található, így ellenőrizzük, hogy ez megadásra került-e (node.Attributes.Contains("src"))? Amennyiben nem, folytassuk a feldolgozást a következő elemmel:

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

    // ...
}
```

A képek elérési útvonala abszolút és relatív formában is szerepelhet az 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);
```

Így már létrehozhatjuk a WebImage példányt és hozzáadhatjuk az osztályban tárolt listához. A WebImage.DownloadAsync() metódus kivételt dobhat (a benne lévő Image.FromStream() hívás), 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
}
```

A WebPage osztály 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 elemet is kell majd feldolgoznunk.

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

3 Nézet

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

3.1 A *MainForm* nézet

A főablakon 4 vezérlőt helyezünk el:

- egy `TextBox`-ot az URL cím bevitelére;
- egy `Button`-t a képek letöltésének elindítására;
- egy `FlowLayoutPanel`-t a képek megjelenítésére (`PictureBox` vezérlőket helyezünk majd rá dinamikusan);
- egy `StatusStrip`-et az állapotsornak.

Az egyes vezérlők `Dock` tulajdonságát módosítva (`Top`, `Fill`, illetve `Bottom` értékekre) dinamikusan kitöltik az ablakot.

A Visual Studio tervező felületén állapotsorra jobb egérgombbal kattintva, majd az *Edit Items* opciót választva szerkeszthetjük az elemeket. (Alternatíva: a *Properties* ablakban az *Items* tulajdonságot szerkesszük.) Vegyünk fel az állapotsorra egy `StatusLabel`-t a letöltött képek számának megjelenítésére és egy `ProgressBar`-t a folyamat előrehaladtának jelzésére.

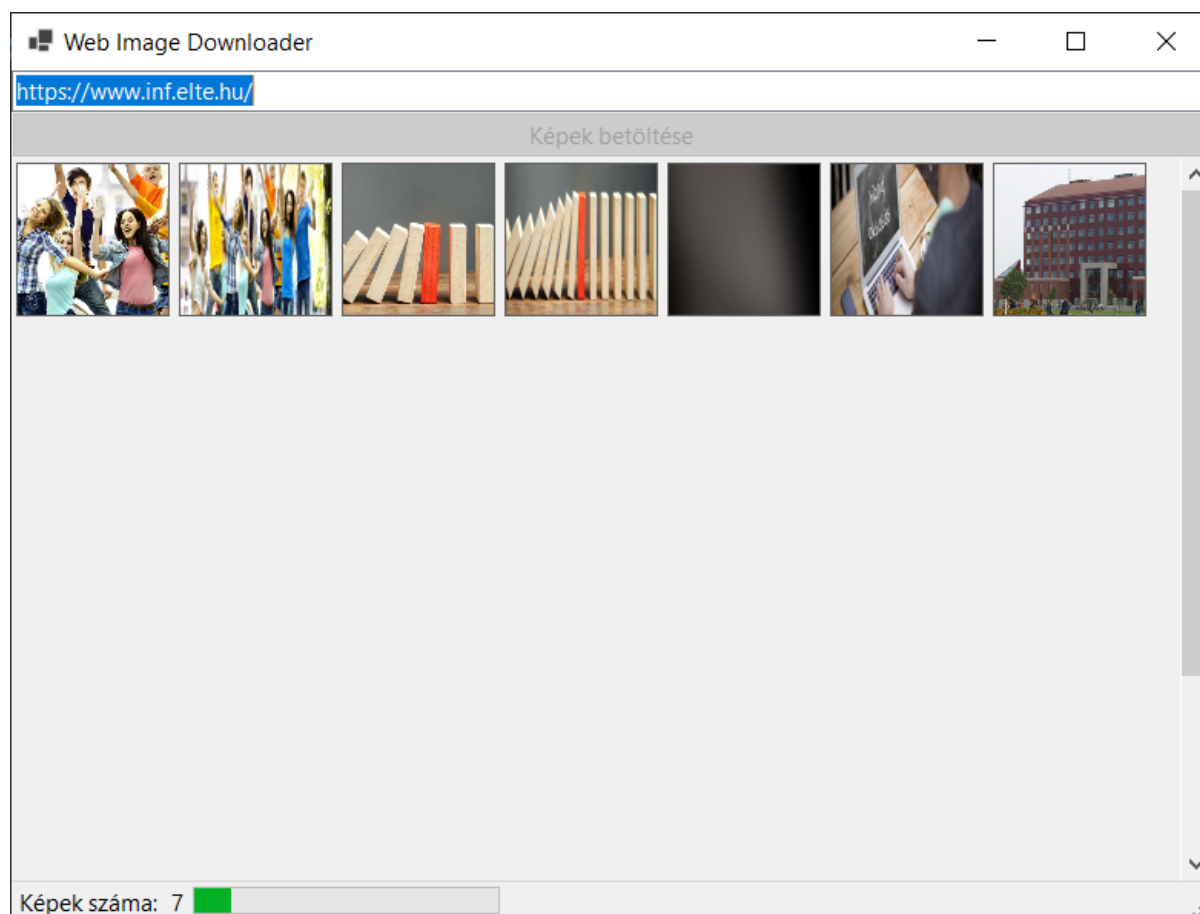


Figure 3: MainForm nézet

A `MainForm` 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 és szintén állítsuk értékét 0-ra. A *Letöltés* gombot tiltsuk le.
2. A **FlowLayoutPanel** elemeit töröljük (korábban betöltött képek).
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 várakozunk az eredményére (`await`).
6. Az eredményét bevárását követően rejtjük 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 nézetben 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. Példányosítsunk egy új **PictureBox** vezérlőt.
2. Állítsuk a méretét (**Size**) `100 x 100` -ra, az átméretezés módját (**SizeMode**) **StretchImage**-re.
3. Állítsuk a megjelenítő képet (**Image**) az esemény argumentumában kapott **WebImage** modell **Image** tulajdonságára. (Mindkettő **System.Drawing.Image** típusú.)
4. Adjuk a **PictureBox**-ot a **FlowLayoutPanel**-hez.
5. Az állapotsorban a letöltött képek számlálóját növeljük 1-gyel.

3.1.2 A *LoadProgress* esemény kezelése

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

3.2 Az *ImageForm* nézet

A kép megjelenítő / letöltő 3 vezérlőt helyezünk el:

- egy **PictureBox**-ot a kép megjelenítésére;
- egy **Panel**-t és rajta egy **Button**-t a kép letöltéséhez.

Az **PictureBox** és **Panel Dock** tulajdonságát módosítva (**Fillés Bottom** értékekre) dinamikusan kitöltik az ablakot.

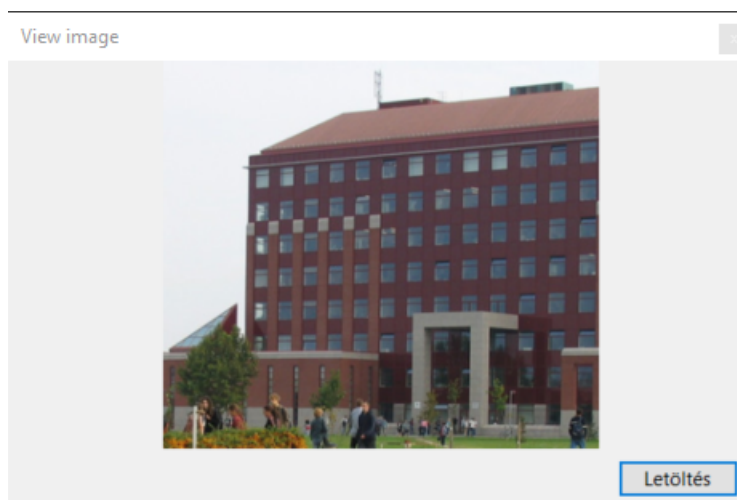


Figure 4: ImageForm nézet

Az **ImageForm** osztály konstruktora várjon egy **System.Drawing.Image** objektumot, ez lesz a megjelenítő és letölthető kép. A konstruktor töltsse is be a **PictureBox** vezérlő **Image** 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 amennyiben a Mentés gombbal zárták be, írjuk ki a megadott útvonalra a képet PNG állományként:

```
if (saveDialog.ShowDialog() == DialogResult.OK)
{
    pictureBox.Image.Save(saveDialog.FileName, ImageFormat.Png);
}
```

Egészítsük ki a `MainForm` 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 `ImageForm` objektumot:

```
private void ShowImage(object sender, EventArgs e)
{
    PictureBox pictureBox = sender as PictureBox;
    if (pictureBox == null)
        return;

    ImageForm form = new ImageForm(pictureBox.Image);
    form.Show();
}
```

4 Megszakítható letöltés

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

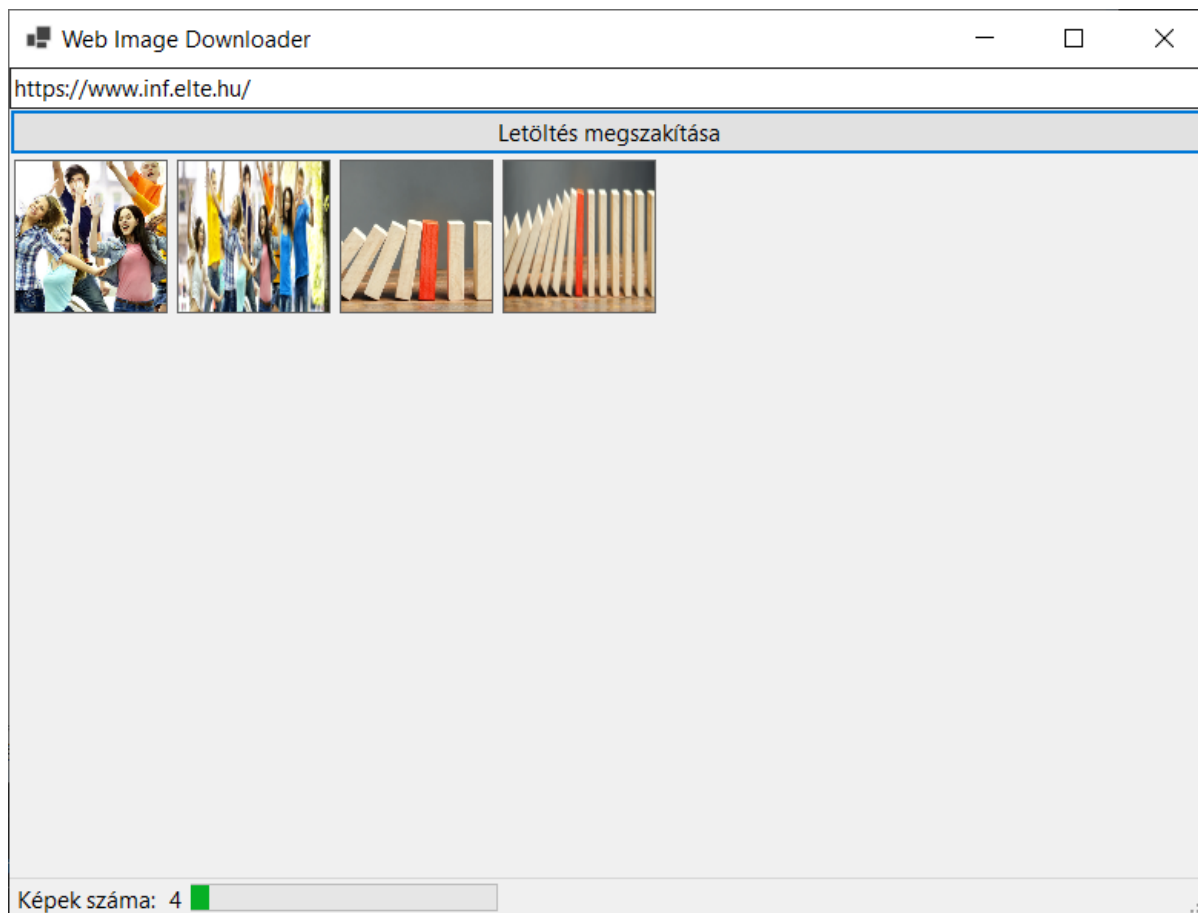


Figure 5: MainForm nézet megszakítható letöltéssel

4.1 A *WebPage* modell módosítása

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

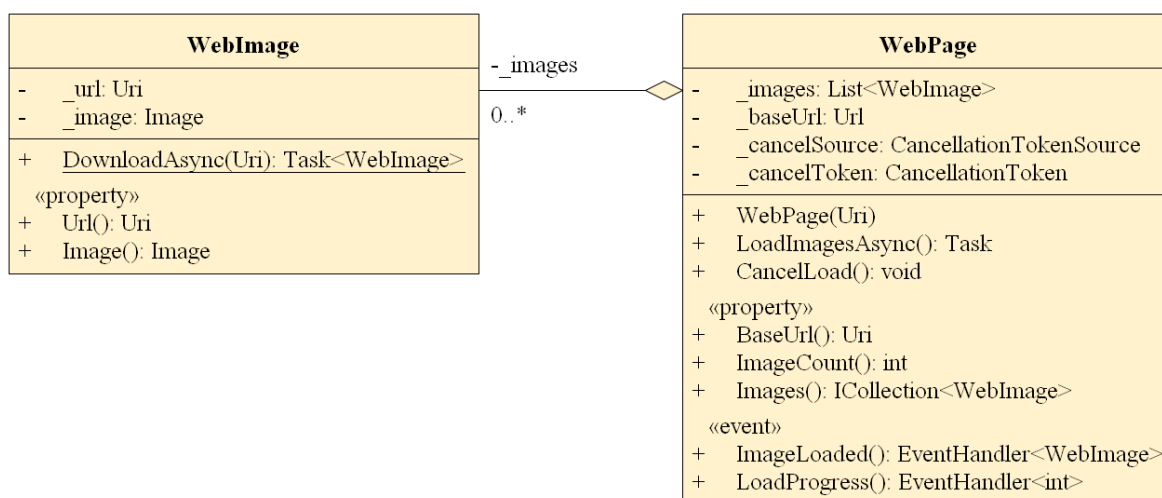


Figure 6: A megszakítható modell osztálydiagramja

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

```
_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 *MainForm* 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 eldönteni, 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 meghívunk.