Lab5 – Web scraper för nedladdning av bilder

Web scraping är en metod för att leta igenom webben, eller avgränsade delar av den, efter data och samla in information. Detta görs oftast genom att skanna (HTML) koden som websidor är byggda av, och i många fall i kombination med en så kallad ”crawler” som följer länkar man hittar för att ta sig vidare till fler sidor.

Uppgiften

Vi ska bygga en enkel web scraper (utan crawler) där man kan mata in en webadress och automatiskt ladda ner bilder som finns på sidan. Appen ska byggas med winforms (.NET Framework) och använda asynkrona anrop.

Programmet består av en TextBox där man matar in en webaddress (URL) och en knapp ”Extract”. När man trycker på knappen (eller enter) så laddar programmet ner HTML-koden från den URL man matat in, söker igenom denna efter länkar till bilder och visar alla länkar den hittar i en multiline TextBox. Det ska även finnas en Label som visar hur många bild-länkar som hittades på sidan.

Sedan ska det finnas en knapp ”Save Images” som öppnar en FolderBrowserDialog där man kan välja en mapp dit man vill spara alla bilderna. När man valt en mapp (och klickat ”OK”) så ska programmet asynkront hämta alla bilder via länkarna och spara ner till filer i den ordning nedladdningarna blir klara.

Detta görs genom att skapa en ny Task för varje nedladdning och sedan använda Task.WhenAny() för att hantera varje Task när den blir klar. När en Task är klar tar ni det binärdata som den laddat ner från länken och skriver rakt av till en binär fil. Ni kan döpa filerna t.ex Image1, Image2 etc. men tänk på att filändelsen måste matcha datat för att filen ska gå att öppna. Så om länken pekar på t.ex en .png fil så måste filen heta t.ex Image5.png

HTML

Websidor byggs vanligen i HTML (HyperText Markup Language) som är ett (markup) språk för att beskriva just websidor. HTML är alltså inget programmeringsspråk på det viset vi är vana vid från C#, utan är bara en (ASCII)

text med så kallade taggar (t.ex. <html>) som beskriver för webläsaren hur sidan ska ritas upp (renderas). Olika taggar beskriver olika saker, och det enda vi behöver veta för den här uppgiften är att det finns en tagg <img> i HTML som innehåller länkar till bilder.

Vi behöver alltså söka efter ”<img” i HTML-koden för att hitta (början på) en sådan tagg. En tagg i HTML börjar alltid med < och slutar med >, så för att hitta slutet på taggen så söker vi framåt tills vi hittar >.

Exempel på en typisk <img> tagg:
<img class=”myclass” src=”/example-image.png”>

HTML taggar kan innehålla noll, en, eller flera attribut. Taggen ovan innehåller attributen ’class’ och ’src’. För denna uppgift behöver vi bara bry oss om värdet på ’src’-attributet. Det värdet är nämnligen den bild-länk vi letar efter. (I bland kan attributet som håller länken kallas ’data-src’ istället).

Regular expressions

Med Regex.Matches() kan ni få ut alla delar av HTML-koden som matchar den expression ni skriver. Det som behövs är i princip en look-behind som kollar så taggen börjar med *<img*, samt att *src=”* kommer precis innan det ni vill matcha. Ni behöver också en look-ahead för att se att den följs av *”* och *>*

Tänk på att ’src’-attributet inte nödvändigtvis är det sista attributet i en <img> tagg. Sen kan olika websidor vara uppbyggda på väldigt många olika sätt. Vissa lagrar bildlänkar i andra typer av taggar, eller på andra sätt. Det är därför väldigt svårt att skriva en generell scraper som hittar alla bilder, och fungerar på alla typer av sidor. Om man vill kan man experimentera med regex och försöka få den att fungera i så många fall som möjligt. För den här uppgiften räcker det dock att det fungerar på en specifik sida, och ni behöver inte heller hitta alla bilder på sidan. Se G- och VG-kriterier.

Tips & Hjälp

Använd GetByteArrayAsync() på HttpClient-objektet för att asynkront ladda ner binärdata till en byte[].

Använd Task.WhenAny() för att skapa en Task som avslutas när någon task i listan är klar.

Använd WriteAsync() på FileStream objektet för att asynkront skriva data till fil.

Om någon av dina metoder använder await, se till att använda await i anropande metoder hela vägen tillbaks till (inklusive) metoden som triggades av ett event.

Event handlers metoder är vanligtvis av typ void. Detta är egentligen det enda fall där det är okej att använda async void. (annars använder ni async Task eller async Task<T>).

När en Task avslutats kan ni kolla om den kastat någon exception med task.Exception (t.ex. om en bild inte kunde laddas ner för att en länk var felaktig).

Redovisning

Lämna in uppgiften på ithsdistans med en kommentar med github-länken.

Jobba själva eller i grupper om 2-3 personer. Om ni jobbar i grupp måste alla skicka in länken till github, samt vem/vilka ni jobbat med.

Jag kommer testa er app mot http://gp.se så se till så att det fungerar främst där.

Betygskriterier

För godkänt:

- Programmet ska fungera enligt ovan beskrivning.
- Nedladdningar och skrivningar till filer ska ske asynkront på ett sådant sätt att ni inte låser UI-tråden.
- Det räcker att appen kan ladda ner 1 typ av fil som ni väljer (t.ex png).
- Appen ska hitta och och kunna spara ner minst 3 bilder på gp.se
- **Alla 3 projekt ska använda .NET framework**, (inte .NET Core)
- **Lösningen ska vara incheckad korrekt på GitHub**
  Rootmappen ska alltså innehålla VS solution-filen (.sln) samt en mapp för varje (3) projekt. Var och en av dessa mappar ska innehålla en VS projektfil (.csproj) och alla projektets .cs filer (och .resx filer för winforms).

För väl godkänt krävs även:

- Koden ska vara väl strukturerad och lätt att förstå.
- Allting ska fungera vid inlämning. Gör det inte det så kommer ni få tillbaks uppgiften med möjlighet att fixa det som krävs för godkänt. Men ni har alltså bara en chans på er att få VG. Så var noga med testning.
- Inlämning sker före deadline.
- Appen ska kunna hitta och korrekt ladda ner filer i formaten bmp, png, gif, och jpg/jpeg.
- Appen ska hitta och spara ner minst 50 bilder på gp.se