

Logging in to Websites-Theory

Bu kısımda scrapy ile sitelere nasıl giriş yaparız onu göstereceğiz, aslında bunu splash ile yapmayı zaten öğrendik diye düşünüyorum, ancak bu kısmı da izleyelim.

Burada login için kullanacağımız site aşağıdaki basit örnek olacak, daha sonra gerçek hayat örneğinde deneme yapacağız.

<http://quotes.toscrape.com/login>

Öncelikle formun javascript ile çalışıp çalışmadığını anlamak istiyoruz bunun için:

Inspector'ı aç > CTRL + Shift + P ile command palette'i aç > disable javascript > refresh the website

UI üzerinde hiçbirşey değişmedi, o yüzden javascript'in gerekli olmadığını söyleyebiliriz.

Quotes to Scrape

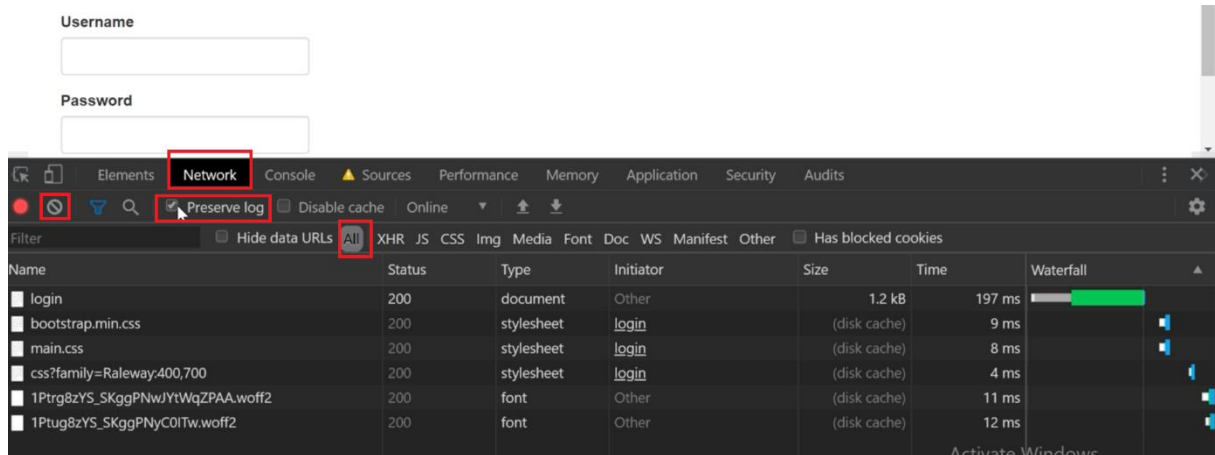
Username

Password

Login

Önce login sırasında nasıl bir request gönderildiğini anlayalım:

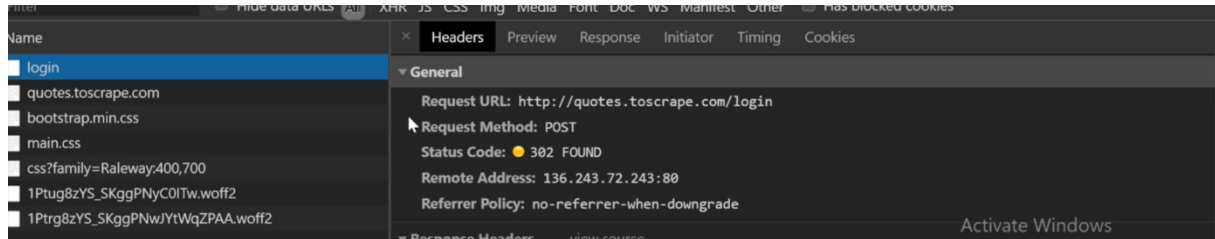
Şimdi inspector'dan network tab'ine geliyorum, filter olarak All'ı seçiyorum ve login olunca gönderilecek request'i kaydetmek için preserve log seçeneğini de işaretliyorum.



Bu website eğitim amaçlı yaratıldığı için username ve password'a istediğimiz bir değer atayabiliriz.

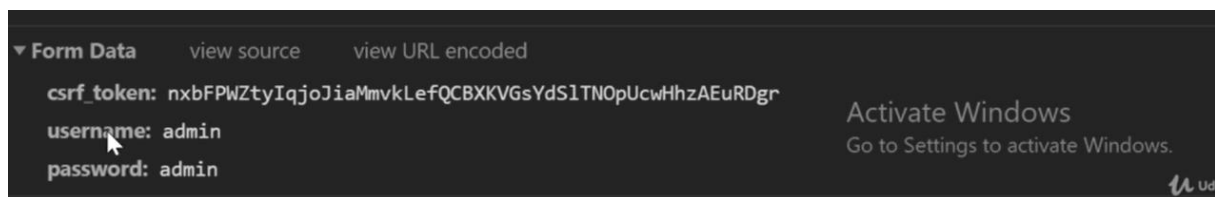
Bu örnek için username'i de password'u da "admin" olarak forma giriyoruz, login demeden önce request'leri temizlemek için üstteki resimde en soldaki kırmızı karedeki clear butonuna tıklıyoruz. Daha sonra login butonuna tıklayarak login işlemini yapıyoruz.

Bu sırada gönderilen login request'i kaydedildi:



Burada görüyoruz ki ...toscrape.com/login url'sine bir POST request gönderilmiş, status code 302 found diyor yani yönlendirilme yapılmış diyor, login işleminden sonra user redirected to the main page.

Daha da aşağı indiğimizde request'in Form Data'sını görüyoruz:

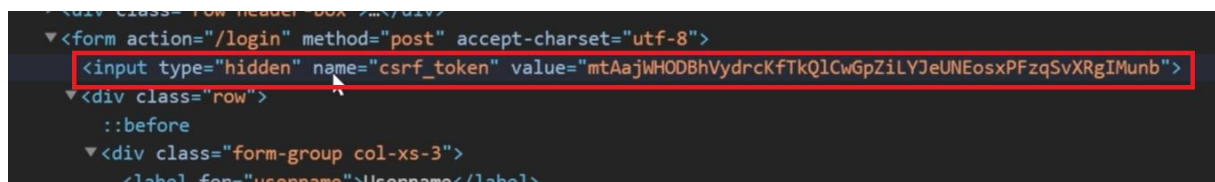


Buradaki bilgiler form'la birlikte nelerin gönderildiğini gösterir.

- csrf_token, username ve password form'la post edilmiş.

İşte scrapy ile login olmaya çalışırken de böyle bir Form_Data'yı içeren bir POST request göndereceğiz. Form_Data içerisindeki csrf_token dinamik olarak üretiliyor ve form içerisinde tutuluyor.

quotes.toscrape.com/login adresine gidip form'u inspect edersek, o anda yaratılmış olan token'ı görebiliyoruz, her refresh ile bu değer değişecektir.



Biz de post request gönderirken bu token'ı login sayfasından scrape edip göndereceğiz.

Build the Spider

Ne yapmamız gerektiğini kabaca anladık, şimdi yeni bir proje oluşturalım ve scrapy ile login işlemini gerçekleştirelim.

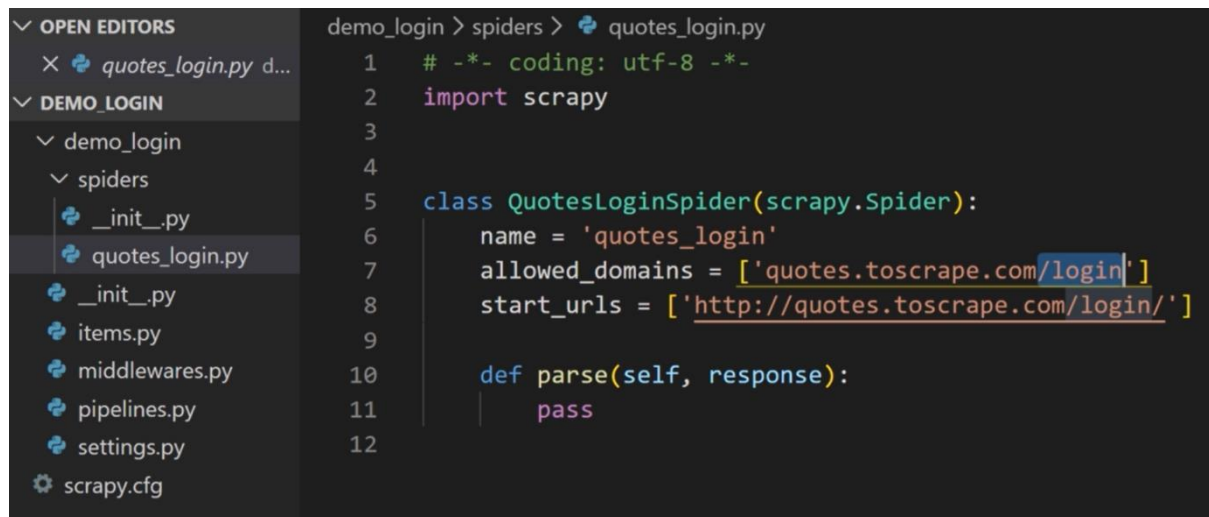
```
scrapy startproject demo_login
```

```
cd demo_login
```

```
scrapy genspider quotes_login quotes.toscrape.com/login
```

proje ve spider yaratıldı, şimdi terminalden vscode'ü doğrudan açacağım:

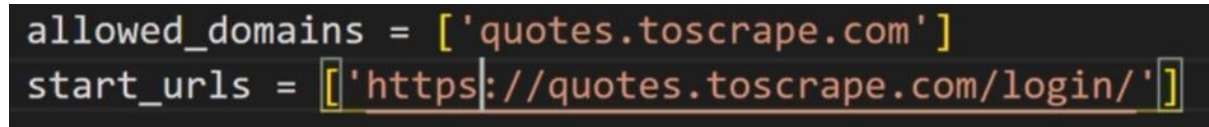
code . dediğimde direkt olarak proje açılacaktır, spider'ımız aşağıdaki gibi olacak:



The screenshot shows the VS Code interface. On the left, the 'EXPLORER' sidebar displays the project structure: 'demo_login' contains 'spiders', which includes 'quotes_login.py'. The main editor area shows the code for 'quotes_login.py'.

```
demo_login > spiders > quotes_login.py
1  # -*- coding: utf-8 -*-
2  import scrapy
3
4
5  class QuotesLoginSpider(scrapy.Spider):
6      name = 'quotes_login'
7      allowed_domains = ['quotes.toscrape.com/login']
8      start_urls = ['http://quotes.toscrape.com/login/']
9
10     def parse(self, response):
11         pass
12
```

İlk olarak allowed_domains'i ve start urls'i düzenliyorum:



This close-up shows the modifications to the 'allowed_domains' and 'start_urls' attributes of the 'QuotesLoginSpider' class.

```
allowed_domains = ['quotes.toscrape.com']
start_urls = ['https://quotes.toscrape.com/login/']
```

Şimdi login sayfasından csrf_token'i elde edelim:

Parse methodunun içinde, yani main url'in response'unun yakalandığı yerde:

```
import scrapy

class QuotesLoginSpider(scrapy.Spider):
    name = 'quotes_login'
    allowed_domains = ['quotes.toscrape.com']
    start_urls = ['https://quotes.toscrape.com/login/']

    def parse(self, response):
        csrf_token = response.xpath('//input[@name="csrf_token"]/@value').get()
```

Token'i bu şekilde elde edebiliyoruz, ne demiştik login işlemi için bu token'i kullanıcı adını ve şifreyi içeren form_data ile bir post request göndermeliyiz.

BURADA start_urls'in sonundaki / bize daha sonradan hata verebiliyor bunu silmek lazım!

Bu post request'i gönderebilmek için bir form request class'ını import etmeliyiz, ve daha sonra bir form request yield edebiliriz:

```
import scrapy
from scrapy import FormRequest

class QuotesLoginSpider(scrapy.Spider):
    name = 'quotes_login'
    allowed_domains = ['quotes.toscrape.com']
    start_urls = ['https://quotes.toscrape.com/login ']

    def parse(self, response):
        csrf_token = response.xpath('//input[@name="csrf_token"]/@value').get()
        yield FormRequest.from_response(
            response,
            formxpath='//form',
            formdata={
                'csrf_token': csrf_token,
                'username': 'admin',
                'password': 'admin'
            },
            callback=self.after_login
        )

    def after_login(self, response):
        if response.xpath("//a[@href='/logout']/text()").get():
            print('logged in')
```

Yukarıdaki kısımları açıklayalım:

- csrf_token main response'dan elde edildikten sonra, bir form request gönderildi.
- Bu FormRequest methodunun ilk argümanı response object, sanıyorum bu response object main response'umuz yani login sayfasına gönderilen request'in response'u.
- Daha sonra submit edilecek form'u identify etmemiz gerek, bunu fromxpath parametresine gerekli xpath expression'ı yazarak elde edebiliriz, yani login sayfasındaki form'u seçmiş olduk.
- Daha sonra gönderilecek post request için formdata'yı tanımlıyoruz.

- Son olarak da post request gönderildikten sonra elde edilen response'un nerede catch edileceğini callback ile belirtiyorum.

After_login methodunun içinde login sonrası yönlendirilen sayfanın response'u yakalanacak, eğer bu response sayfasında logout elemanı exists ise print('logged in') dendi.

Artık bu kodu kaydedip, integrated terminal'de çalıştırabiliriz:

```
scrapy crawl quotes_login
```

Sonuçta terminalde logged in ifadesini göreceğiz:

```
2020-04-22 21:45:17 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://quotes.toscrape.com/login>
: http://quotes.toscrape.com/login)
logged in
2020-04-22 21:45:17 [scrapy.core.engine] INFO: Closing spider (finished)
2020-04-22 21:45:17 [scrapy.statscollectors] INFO: Dumping Scrapy stats:
```

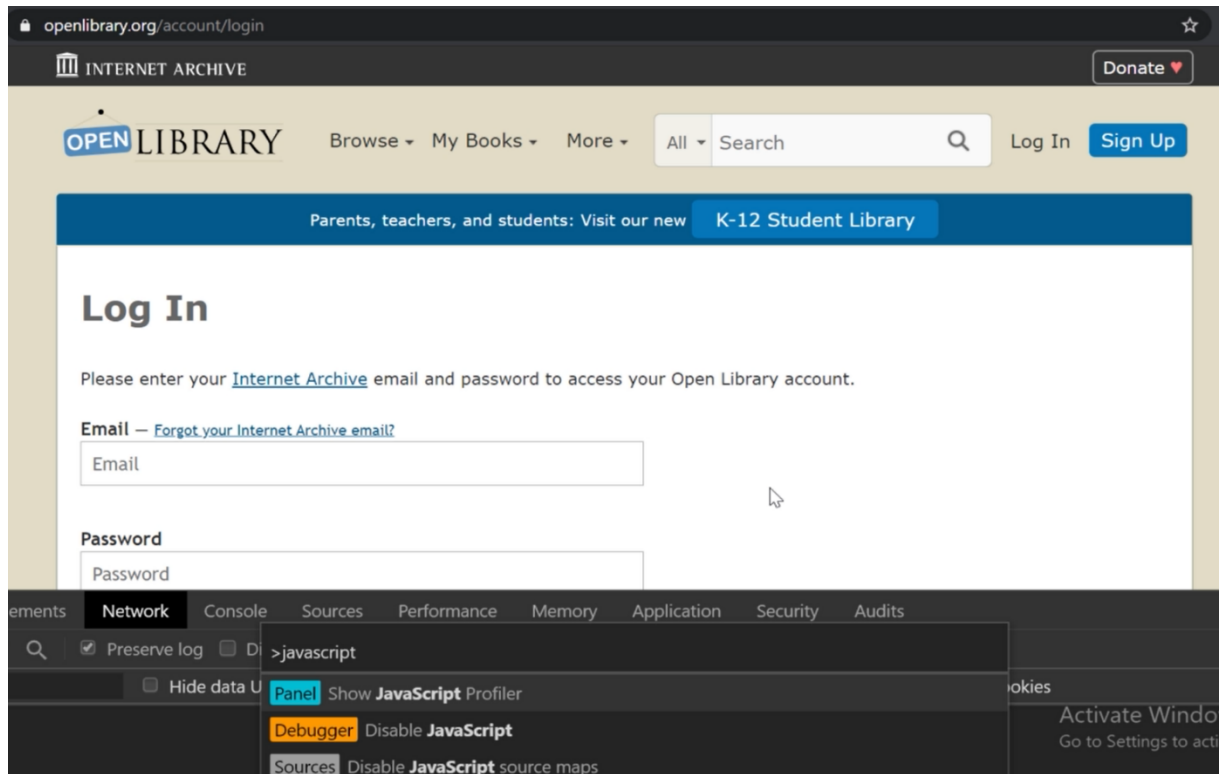
Yani login işlemi başarıyla gerçekleştirildi.

Logging in to Websites-Real World Example

Intro

Geçtiğimiz kısımda login işlemini örnek bir site üzerinde gerçekleştirdik, şimdi gerçek bir sitede aynı işlemi gerçekleştirelim.

Örneğimiz openlibrary.com, öncelikle bu adrese gidip bir hesap yaratıyoruz daha sonra [openlibrary.com/account/login](https://openlibrary.org/account/login) sayfasına gidiyoruz, ve login için javascript'in gerekli olup olmadığını aynı bir önceki sectiondaki gibi test ediyoruz.



Görüyorum ki javascript'i disable etsemde UI üzerinde bir değişim olmadı, yani login için javascript gerekmiyor.

Bir önceki section'daki gibi deneme için login yapıyorum ve login request'ini kaydettim, şimdi de içini inceliyorum. Form Data'sı aşağıdaki gibi:

Form Data view source view URL encoded

username: meinherz75@gmail.com

password: test123

redirect: /

debug_token:

login: Log In

Yukarıdaki son üç variable yani redirect, debug_token ve login html markup içerisinde gömülü şekilde bulunabilir ancak bu değerler static olduğu için biz post request'imizi gönderirken bu değerleri scrape etmeyeceğiz, onun yerine doğrudan variable olarak elle tanımlayacağız.

Building the Spider

Önceki projeyi bozmadan, aynı projenin içerisinde yeni bir spider yaratıyorum ismine de openlibrary_login diyorum:

```
PS C:\Users\Ahmed\projects\demo_login> scrapy genspider openlibrary_login openlibrary.org/account/login
Created spider 'openlibrary_login' using template 'basic' in module:
demo_login.spiders.openlibrary_login
PS C:\Users\Ahmed\projects\demo_login>
```

Şimdi proje içerisinden yeni yaratılan spider'ı açıyorum:

```
quotes_login.py d... 1  # -*- coding: utf-8 -*-
X openlibrary_login... 2  import scrapy
DEMO_LOGIN 3
demo_login 4
  spiders 5  class OpenlibraryLoginSpider(scrapy.Spider):
    _init_.py 6      name = 'openlibrary_login'
    openlibrary_login.py 7      allowed_domains = ['openlibrary.org/account/login']
    quotes_login.py 8      start_urls = ['http://openlibrary.org/account/login/']
    _init_.py 9
    items.py 10     def parse(self, response):
    middlewares.py 11         pass
12
```


Aynı önceki örnekteki gibi allowed domains ve start_urls değişkenlerini set ediyorum:

```
allowed_domains = ['openlibrary.org']
start_urls = ['https://openlibrary.org/account/login']
```

Daha sonra post request'i göndermek için aynı önceki örnekteki gibi gerekli işlemleri yapıyorum.

```
# -*- coding: utf-8 -*-
import scrapy
from scrapy import FormRequest

class OpenlibraryLoginSpider(scrapy.Spider):
    name = 'openlibrary_login'
    allowed_domains = ['openlibrary.org']
    start_urls = ['https://openlibrary.org/account/login']

    def parse(self, response):
        yield FormRequest.from_response(
            response,
            formid='register',
            formdata={
                'username': 'meinherz75@gmail.com',
                'password': 'test123',
                'redirect': '/',
                'debug_token': '',
                'login': 'Log In'
            },
            callback=self.after_login
        )

    def after_login(self, response):
        print('logged in...')
```

- Burada formdata içeriğinin, inspector üzerinde yapılan test login sonrası elde edilen request'in içeriğine bakılarak düzenlendiğini unutma!
- Bunun dışında kalan kısımlar aynı, ilgili response'daki form elemanı bu kez xpath ile değil id bazlı seçilmiş bu yüzden formid kullanılmış.
- Form_data set edildikten sonra, ilgili login response'un yakalanacağı method callback olarak belirtilmiş.

After login methodu login response'u catch edip, ekrana logged in... ibaresini print ediyor.

`scrapy crawl openlibrary_login` ile bu spider'ı çalıştırabilirim ve logged in ibaresini göreceğim.

Logging in to JavaScript Websites

Hi guys, just one quick remark here, if the form does **require JavaScript** then **you can't use the FormRequest class**, so as an alternative solution you have to use another class called **SplashFormRequest** which does have the same methods as in the **FormRequest** class and takes the same arguments.

Example:

```
1  # Splash should be running on the background
2
3  import scrapy
4  from scrapy_splash import SplashRequest, SplashFormRequest
5
6
7  class QuotesLoginSpider(scrapy.Spider):
8      name = 'quotes_login'
9      allowed_domains = ['quotes.toscrape.com']
10
11     script = '''
12         function main(splash, args)
13             assert(splash:go(args.url))
14             assert(splash:wait(0.5))
15             return splash:html()
16         end
17     '''
18
19     def start_requests(self):
20         yield SplashRequest(
21             url='https://quotes.toscrape.com/login',
22             endpoint='execute',
23             args = {
24                 'lua_source': self.script
25             },
26             callback=self.parse
27         )
28
29     def parse(self, response):
30         csrf_token = response.xpath('//input[@name="csrf_token"]/@value').get()
31         yield SplashFormRequest.from_response(
32             response,
33             formxpath='//form',
34             formdata={
35                 'csrf_token': csrf_token,
36                 'username': 'admin',
37                 'password': 'admin'
38             },
39             callback=self.after_login
40         )
41
42     def after_login(self, response):
43         if response.xpath("//a[@href='/logout']/text()").get():
44             print('logged in')
```

Elbette yukarıdaki örneğin çalışması için splash'ı projemize dahil etmeliyiz, bunun için splash örneklerine bakarsan, öncelikle settings.py üzerinde bazı değişiklikler yapmamız gerektiğini görürsün onları yapıp daha sonra buna dönebiliriz.

Yukarıdaki örnekte olan şey şu, start request içinde tanımlanan şeyin amacı script içerisinde belirtilen lua kodunun yani splash kodunun ilgili url'e main request'i göndermesi, daha sonra bu request ile login sayfasına gidilmiş olacak, login sayfasının response'u parse içerişine alınacak.

İşte bu noktadan sonra yapılan işlem bir post request göndermek, ancak burada post request'i de splash ile gönderiyoruz çünkü javascript gerekiyor. Yine de yöntem çok çok benzer.