SCRAPING APIs PART 1 – INTRO

BTCTurk pro sayfasını scrape etmeye çalıştım ama sayfa veriyi API'dan aldığı için ne scrapy ile ne splash ile sayfayı scrape edemedim. İşte bu yüzden bu kısımı çalışacağım.

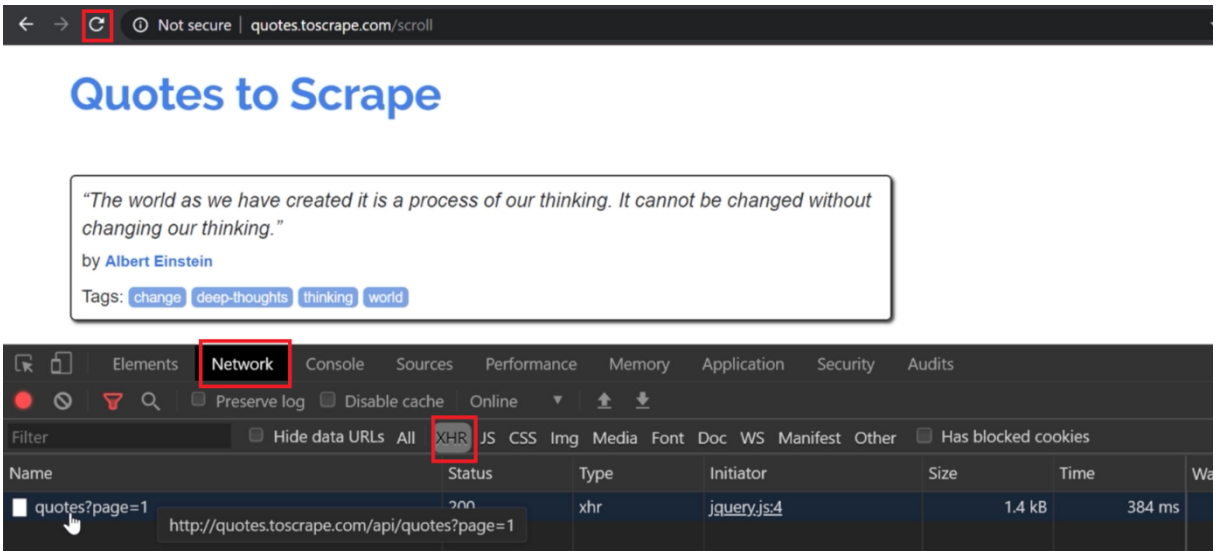
API scraping yaparken her zaman değil ama çoğunlukla html markup'ı scrape etmeyiz. Yani Xpath veya CSS Selectors ile uğraşmayız. Bunun yerine büyük bir Json object'i parse etmemiz gerekir.

Öncelikle örnek olarak aşağıdaki website'ı scrape edeceğiz, daha sonra real world örneğe geçeceğiz.

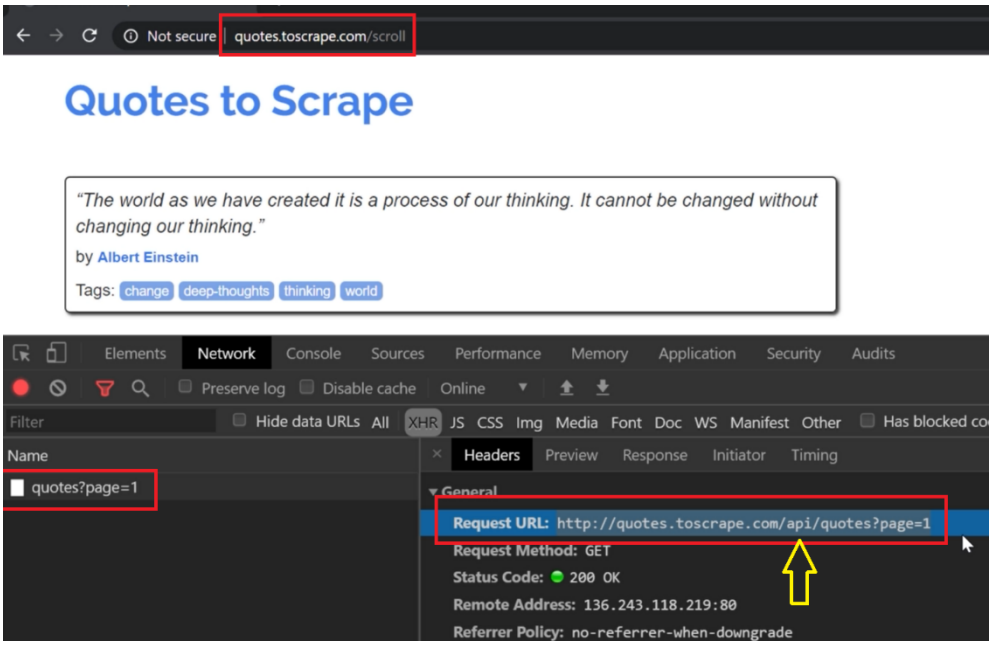
http://quotes.toscrape.com/scroll

Bu sayfada pagination yok ancak, scroll down yaptıkça yeni quote'lar insert ediliyor. Böyle sayfalara dynamic pages denir, yani javascript dependent pages. Ancak sırf sayfa javascript kullanıyor diye bu demek değil ki splash kullanmalıyız.

Sayfadayken developer tools'u açalım F12 > Network tabine gelelim > API olup olmadığını anlamak için XHR(Xml Http Request)filtresini uygulayalım > Sayfayı yenileyelim CTRL + R

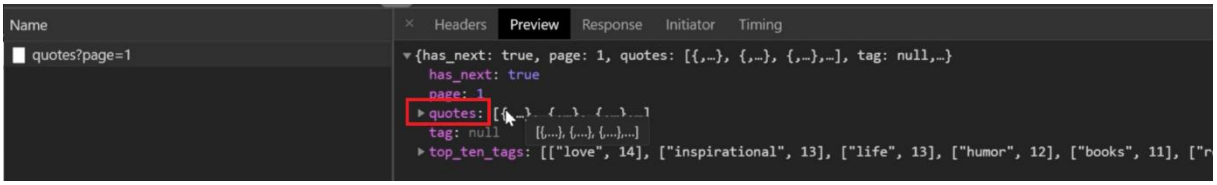


Sayfa yenilenince yukarıdaki gibi quotes?page=1 isminde bir request'in kaydedildiğini görüyoruz.



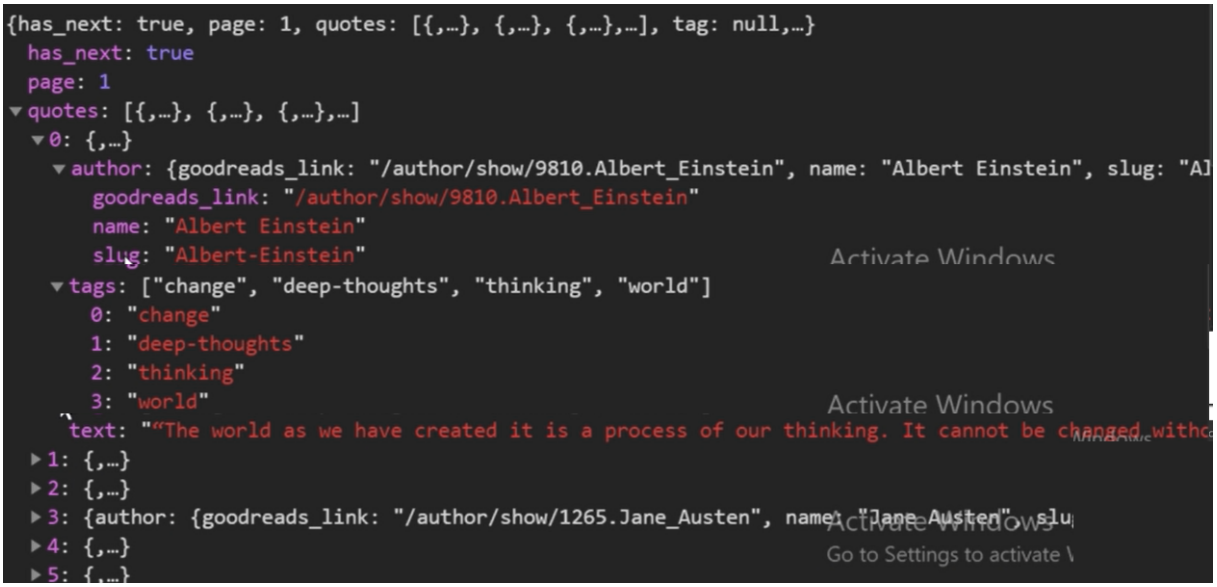
Burada ilgili request'in headers kısmına geldiğimizde Request Url'in sayfa url'inden farklı olduğunu görürüz, eğer sayfa API kullanıyorsa bu ikisi her zaman farklı olur!

Şimdi ilgili requestteyken Preview sekmesine geliyoruz:



Burada görünen şey bir JSON OBJECT'tir!

- has_next, page quotes, tag ,top_ten_tags gibi bazı variables görünüyor. Bunları daha detaylı inceliyor olacağız.
- Burada önemli olan quotes kısmı: quotes'un içine bakalım:



- Görüyoruz ki quotes'un içinde 0:, 1:, 2: şeklinde görünen farklı farklı objeler yer alıyor, aslında her object bir quote'a karşılık geliyor.
- Objenin içine girdiğimizde author, tags ,text gibi özellikleri görüyoruz.

Böylece bir giriş yapmış olduk, sonraki kısımda scraping'e başlayalım.

SCRAPING APIs PART 2 – START SCRAPING

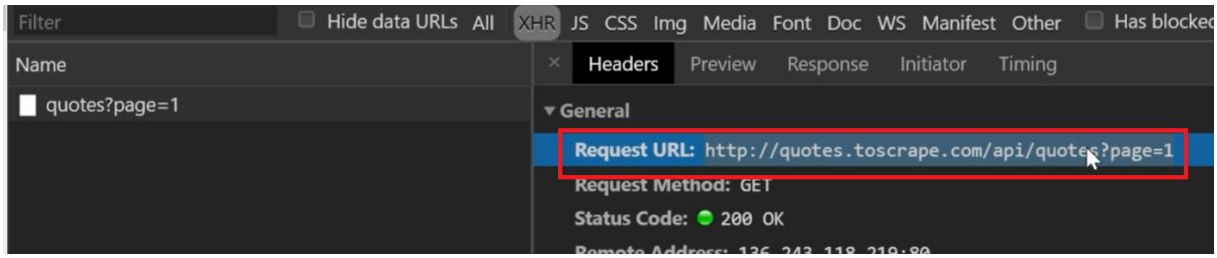
Öncelikle yeni bir project oluşturalım:

- scrapy startproject demo_api
- cd demo_api
- scrapy genspider quotes quotes.toscrape.com
- code .

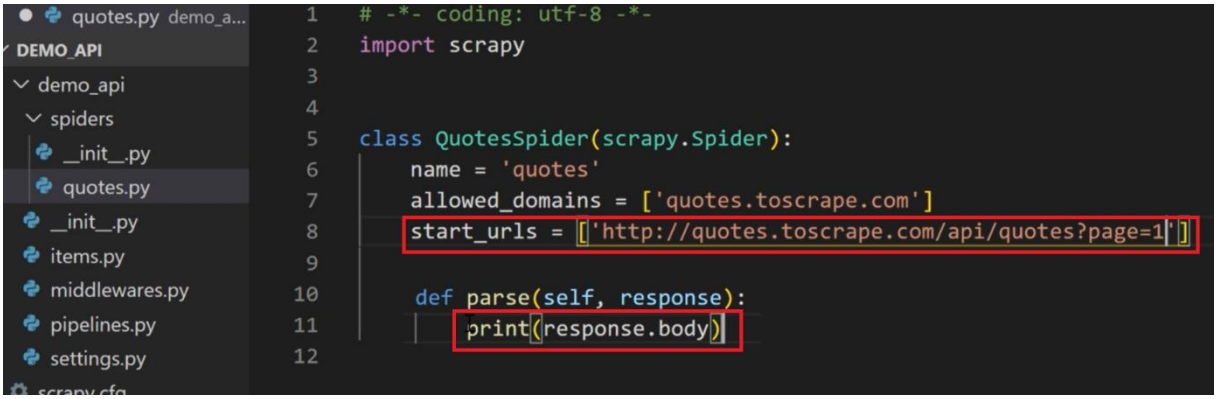
→API scrape ederken hep base template'i kullanın, eğer crawl spider'ı kullanırsak rule objects'i define edemeyiz, çoğu zaman sayfada follow edilecek linkler olmayacaktır.

Code . komutu ile terminalde vscode'u açarız.

Spider'ımıza girmeden önce browser üzerinden Request URL'i kopyalarız, bu sayfanın URL'i değil, API'in url'i oluyor anladığım kadarıyla.

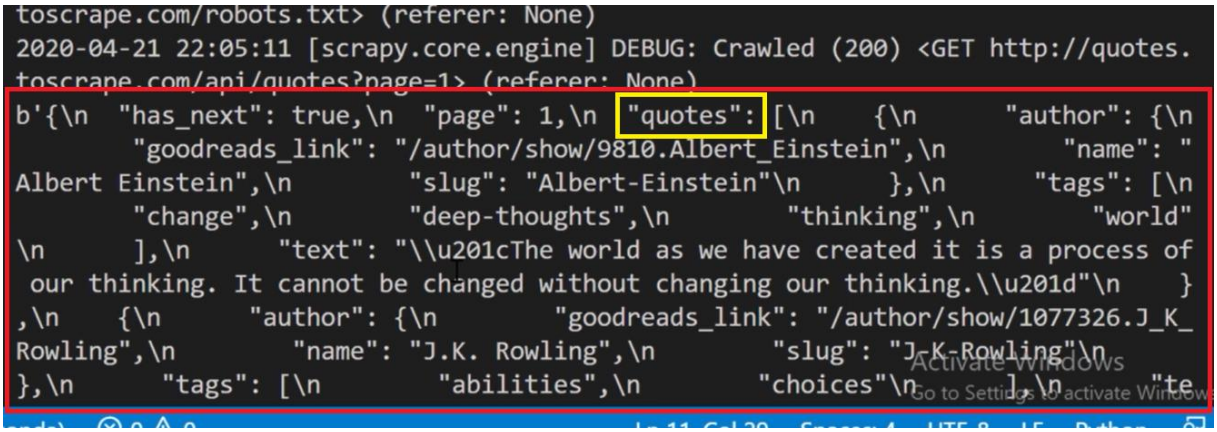


Şimdi spider'a dönüyorum ve start_urls'in içine kopyalanan url'i yapıştırıyorum yani initial scrapy request'in API url'ine gönderilmesini sağlayacağım:



Ayrıca API url'ine gönderilen initial request'in response body'sini görüntülemek için parse içinde print(response.body) komutunu çağırdık:

Terminalde scrapy crawl quotes ile spider'ı çalıştıralım ve response body'e bakalım:



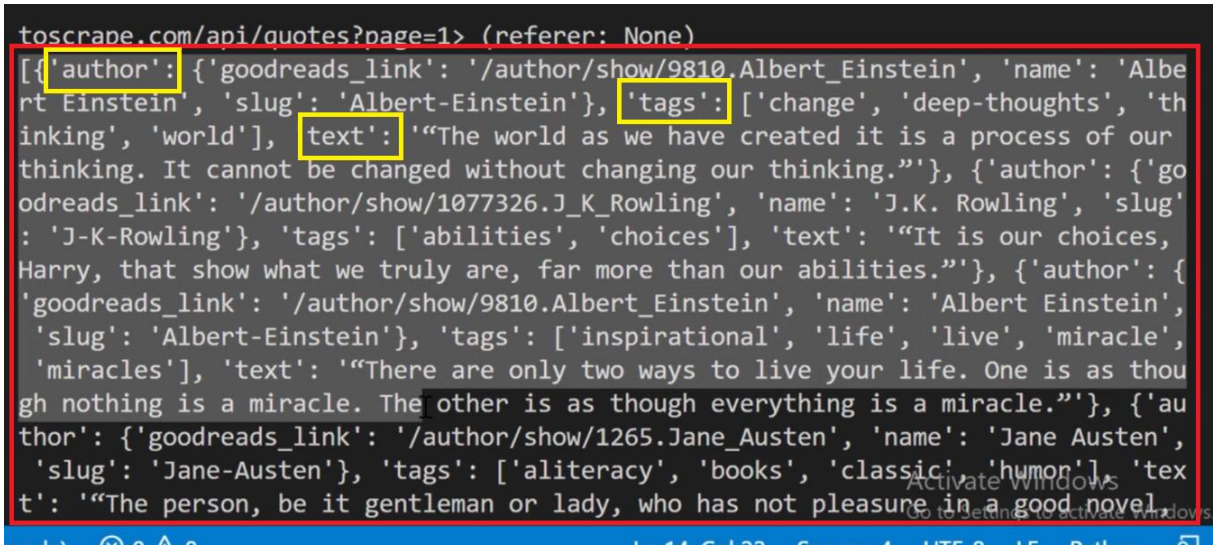
- Sonuçta bir Json object outpu edildiğini görüyoruz.
- Bizim amacımız bu Json object içindeki quotes kısmını elde etmek, çünkü quote'larla ilgili asıl bilgi bunun içinde yatıyor.

Sıradaki aşamada yapılacak olan şey şu: spider içerisinde bu Json object'i bir python dictionary'e cast edeceğiz ki içerisindeki verilere rahatlıkla ulaşabilelim:

```
1   # -*- coding: utf-8 -*-
2   import scrapy
3   import json
4
5
6   class QuotesSpider(scrapy.Spider):
7       name = 'quotes'
8       allowed_domains = ['quotes.toscrape.com']
9       start_urls = ['http://quotes.toscrape.com/api/quotes?page=1']
10
11      def parse(self, response):
12          resp = json.loads(response.body)
13          quotes = resp.get('quotes')
14          print(quotes)
15
```

- Bunu yapabilmek için json module'ümü import ediyoruz.
- Daha sonra bu json.loads() methodu ile, bir json object olan response.body'i bir dictionary'e çeviriyoruz.
- Son olarak elde edilen dictionary içerisindeki quotes key'ini çekip print ettiriyoruz:

İlgili spider'ı scrapy crawl quotes ile çalıştırırsak:



- Yukarıdaki gibi bir liste içerisinde tüm quote'ları elde ederiz, liste içinde bir çok dictionary var her dictionary içerisinde authot, tags ve text key'leri var.

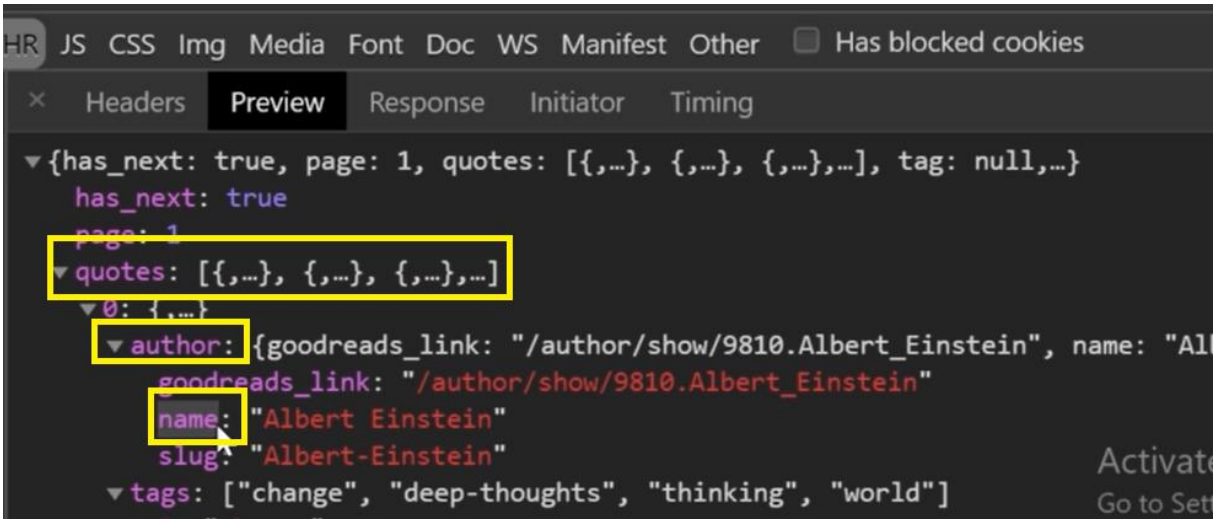
O halde bu kısımda yapılacak son iş her quote'un auhor'ını tags'ini ve text'ini ayrı ayrı yield etmek olacak.

Anladığımız üzere elde edilen quotes içindeki bir sürü quote içeren bir liste, her bir quote bir dictionary olarak tutuluyor.

O halde quotes'u for loop ile gezebilir ve her quote dictionary'si içinden istediğimiz values'u yield edebiliriz:

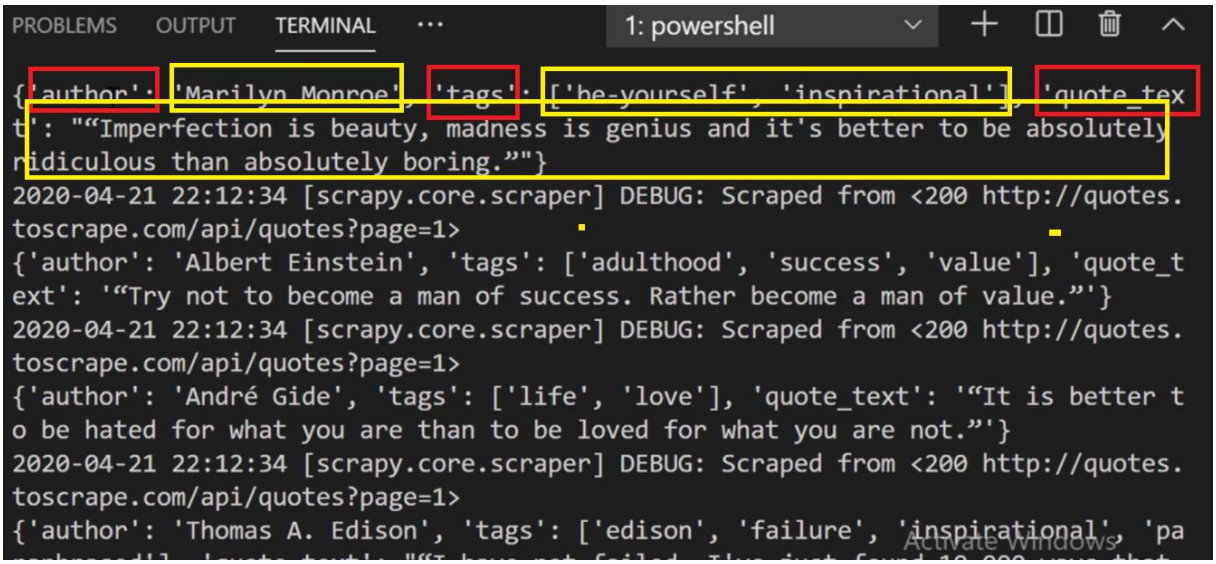
```python
# -*- coding: utf-8 -*-
import scrapy
import json


class QuotesSpider(scrapy.Spider):
    name = 'quotes'
    allowed_domains = ['quotes.toscrape.com']
    start_urls = ['http://quotes.toscrape.com/api/quotes?page=1']

    def parse(self, response):
        resp = json.loads(response.body)
        quotes = resp.get('quotes')
        for quote in quotes:
            yield {
                'author': quote.get('author').get('name'),
                'tags': quote.get('tags'),
                'quote_text': quote.get('text')
            }
```


Burada author u elde ederken farkındaysan iki kere get kullandım bunun sebebini browser'a gidip preview'e bakarak anlarız.



Elde etmek istediğimiz author aslonda quotes içindeki author içindeki name property'si bu yüzden 2 kere get kullandık.

Sonuçta görüldüğü gibi output'lar yield edilebilidi:



YALNIZ UNUTMA Kİ BURADA YIELD EDİLEN QUOTE'LARIN HEPSİ YALNIZCA İLK SAYFADAN ELDE EDİLEN QUOTE'LARDIR!

DYNAMIC SAYFADA AŞAĞIYA DOĞRU İNDİKÇE YENİ PAGE'LERE GEÇİLİYOR, TIKLAMA İLE DEĞİL, SCROLL DOWN YAPILDIKÇA YENİ PAGE İÇİN API'A REQUEST GÖNDERİLİYOR.

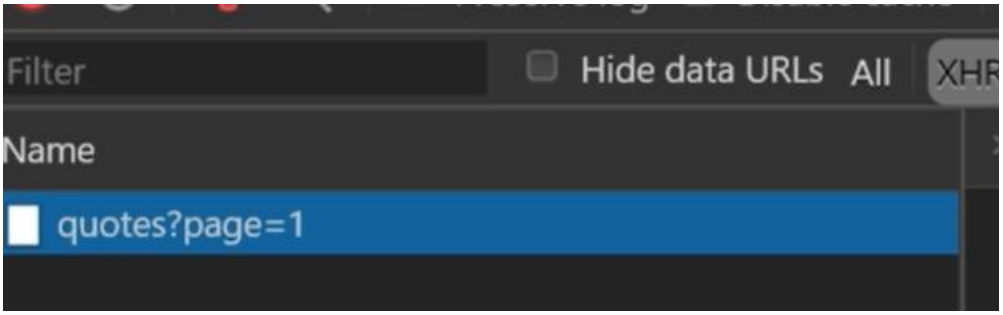
HER GÖNDERİLEN REQUEST'İN URL'Sİ FARKLI. BİZ YUKARIDA START URL'İ SADECE PAGE 1 İÇİN SET ETTİK BU YÜZDEN SADECE PAGE 1'İ SCRAPE EDEBİLDİK.

BİR SONRAKİ KISIMDA DİĞER SAYFALARDAKİ QUOTE'LARI NASIL ELDE ECEĞİMİZİ DE GÖRECEĞİZ.

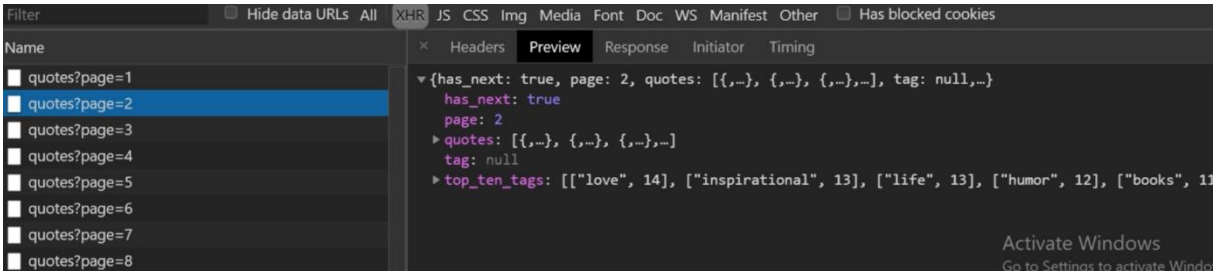
SCRAPING APIs PART 3 – HOW TO HANDLE PAGINATION

Önceki başlıkta ilk page'i scrape ettik. Request'i hatırlarsak aşağıdaki gibi page=1 isminde bir request'i scrape etmiştik.

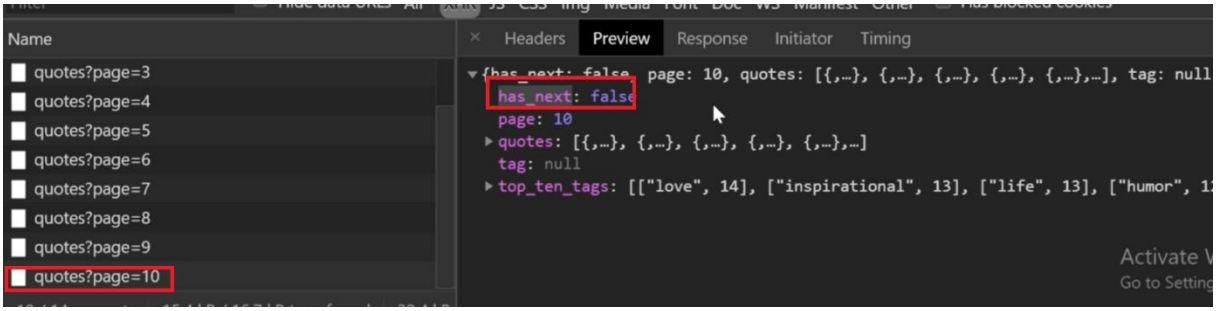
Sadece bu request'in request header'ını kullandık, bu request API'a gönderilen bir request idi.



Şimdi sayfada scroll down yaptıkça, başka başka API request'lerin ortaya çıktığını görüyoruz:



- Yukarıda her bir request başka bir url'ye gönderilmiş, ve her birinin response'u başka bir sayfayı temsil ediyor, her sayfada birçok quotes yer alıyor.
- Yukarıda page=2'nin içine bakarsak daha önce olduğu gibi has_next, page, ve quotes gibi özellikleri görebiliyoruz.
- Burada has_next: true demek, bu sayfa son sayfa değil demek!
- 10. Sayfa request'inin preview'ine bakarsak, bunun son sayfa olduğunu anlarız:



Bu noktada bizim amacımız, tüm sayfaları gezip her sayfanın içindeki quote'ları elde etmek.

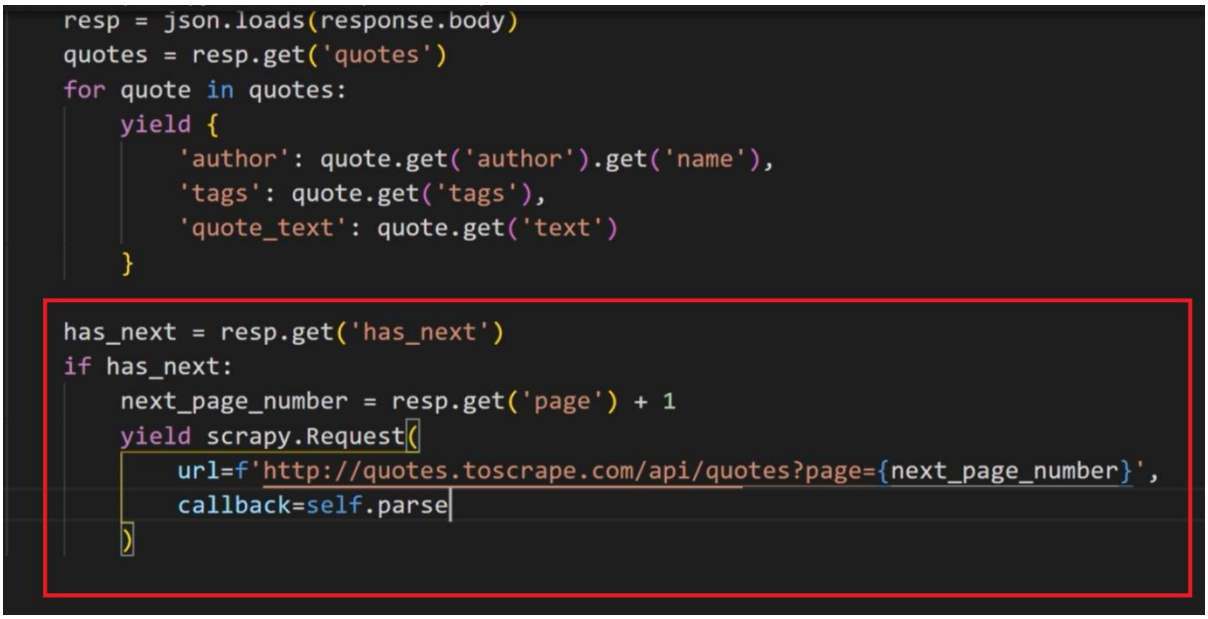
- İşte bunu gerçekleştirmek için bu has_next property'sinden yararlanacağız.
- İlk request'i page=1'a gönderdikten ve scraping'i yaptıktan sonra has_next property'sini check edeceğiz eğer bu değer false ise spider'ı sonlandırabiliriz. Yok eğer bu değer true ise, next page exist demek o halde next page'i elde edecek api url'ini construct edeceğiz ve ilgili url'e request'imizi yollayacağız.

Sadece ilk sayfayı scrape eden kodu hatırlayalım, page=1 api url'ine request yollanıyordu ve response.body aşağıdaki gibi json.loads ile dictionary'e cast ediliyordu, ve bu dictionary içerisinden quotes çekilip, quote'lar tek tek dönülüyor ve ilgili elemanlar yield ediliyordu.

```python
# -*- coding: utf-8 -*-
import scrapy
import json


class QuotesSpider(scrapy.Spider):
    name = 'quotes'
    allowed_domains = ['quotes.toscrape.com']
    start_urls = ['http://quotes.toscrape.com/api/quotes?page=1']

    def parse(self, response):
        resp = json.loads(response.body)
        quotes = resp.get('quotes')
        for quote in quotes:
            yield {
                'author': quote.get('author').get('name'),
                'tags': quote.get('tags'),
                'quote_text': quote.get('text')
            }
```


İlk sayfa yukarıdaki gibi scrape edildikten sonra,

For loop'unun dışına bir has_next property'si tanımlıyoruz, ve eğer bu property exists ise next_page'in url'ini consturct ediyoruz ve bu url'e request gönderiyoruz, response'u da tekrar aynı parse methodunda yakalıyoruz.



```
resp = json.loads(response.body)
quotes = resp.get('quotes')
for quote in quotes:
    yield {
        'author': quote.get('author').get('name'),
        'tags': quote.get('tags'),
        'quote_text': quote.get('text')
    }

has_next = resp.get('has_next')
if has_next:
    next_page_number = resp.get('page') + 1
    yield scrapy.Request(
        url=f'http://quotes.toscrape.com/api/quotes?page={next_page_number}',
        callback=self.parse
    )
```

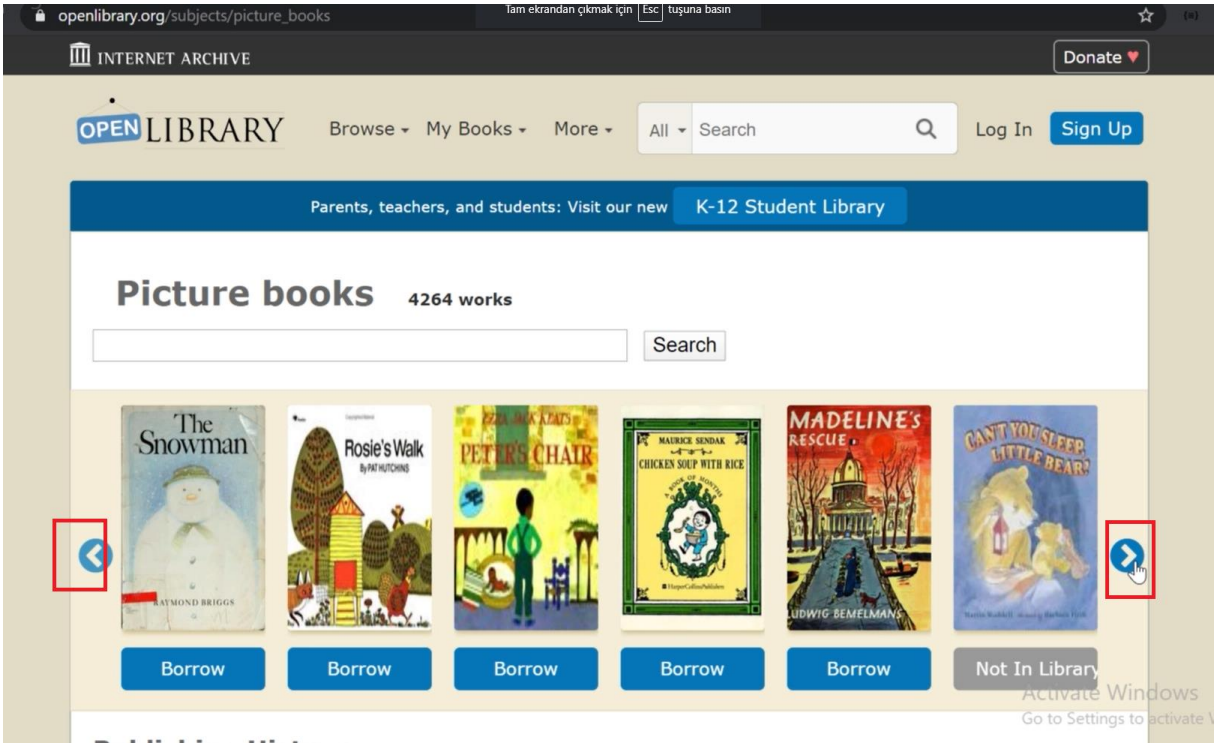
- Yeni url'yi construct ederken, current page'in page property'sini kullandığımıza dikkat et.
-

Sonuçta bu şekilde, her sayfa için quotes scrape edilecek! Böylelikle örneği bitirmiş olduk.

ANCAK UNUTMA Kİ BÜTÜN API'LAR AYNI ŞEKİLDE ÇALIŞMAZ, AYNI MANTIK HEPSİNDE İŞLEMEYEBİLİR BU YÜZDEN ÖNCELİKLE BROWSER ÜZERİNDEN API'YI KEŞFETMEKTE FAYDA VAR.

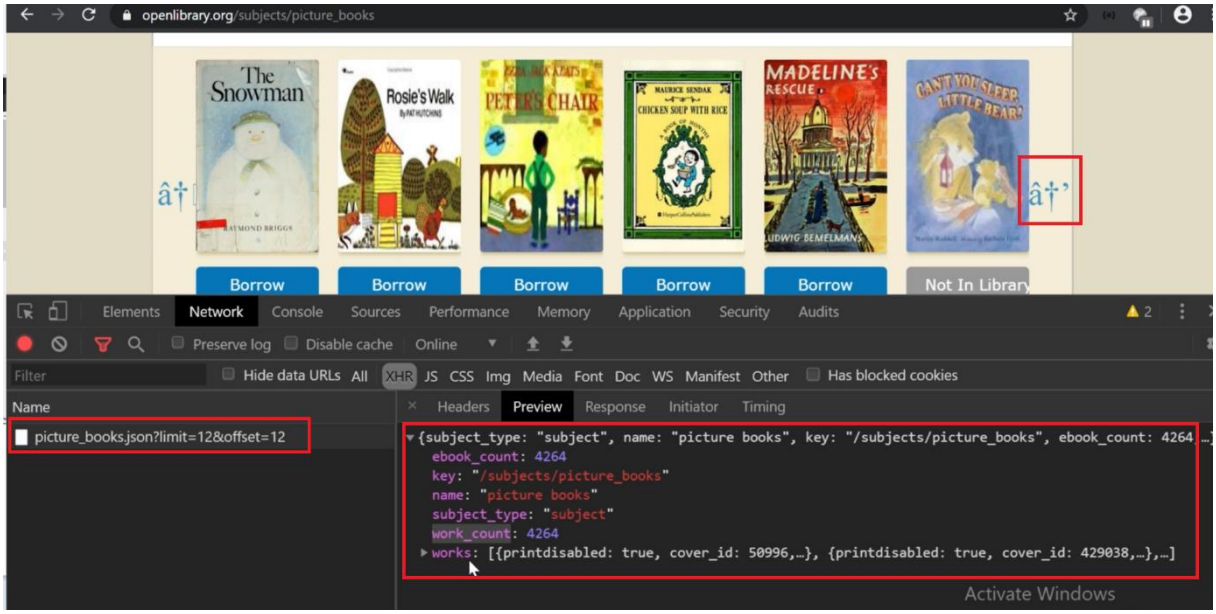
SCRAPING APIs PART 4 – PROJECT: SCRAPE THE FIRST PAGE

Bu örnek için scrape edilecek page:
https://openlibrary.org/subjects/picture_books



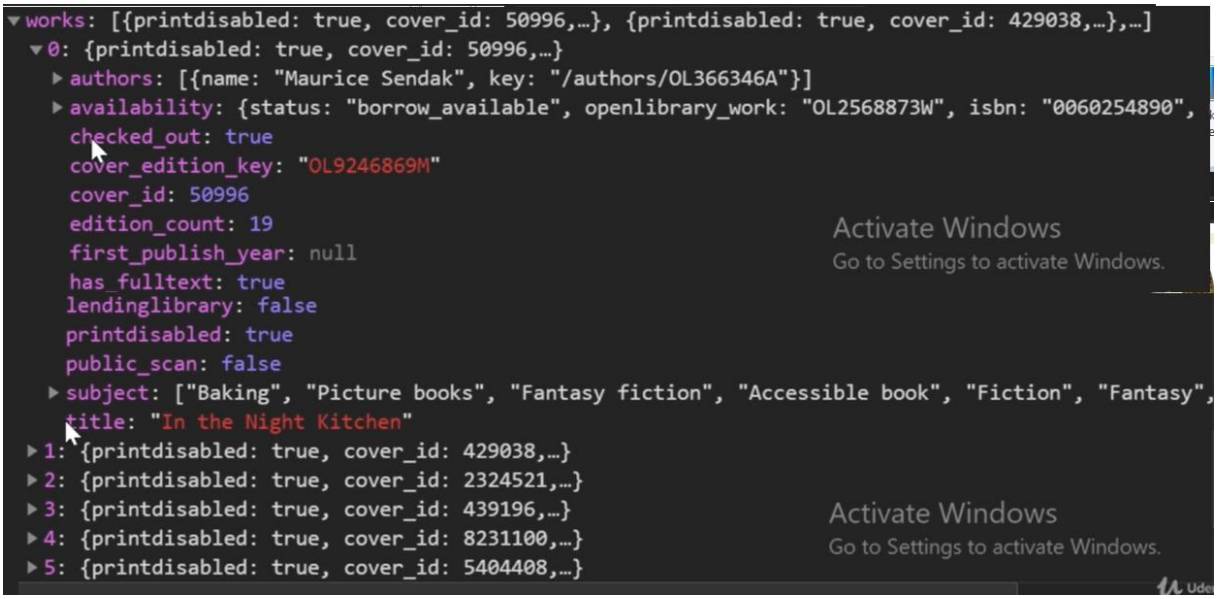
Yukarıda kırmızı ile işaretli butonlara tıkladığımızda yeni picture books görünüyor, projede amacımız bu kitapları scrape etmek.

Öncelikle sayfayı keşfetmek için developer tool'u kullanıyoruz:



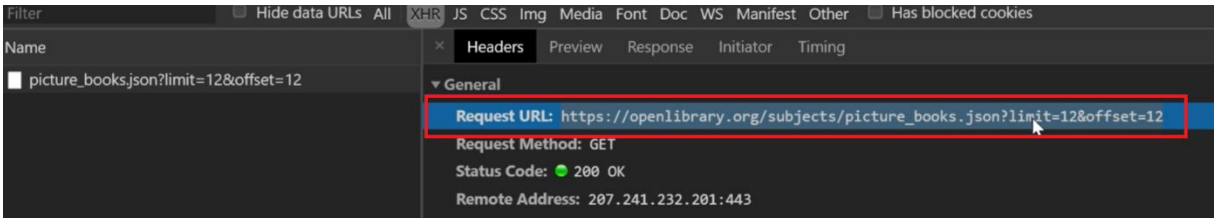
Developer'ı açıp Network'den XHR'ı işaretleyip, sayfada ileri tuşuna bastığımda yeni kitapların ekrana gelmesini sağlayan XML http request'i görüyorum, bu reqest'in preview'ine baktığımızda, bize gönderilen response'u görürüyoruz.

- ebook_count, key, name, subject_type, work gibi değişkenleri görüyoruz.
- work'ün içine bakarsak da 0, 1 , 2 objelerini görüyoruz. Bunların her biri bir kitabı temsil ediyor sanıyorum, sonuçta burada değerleri scrape ediyor olacağız.

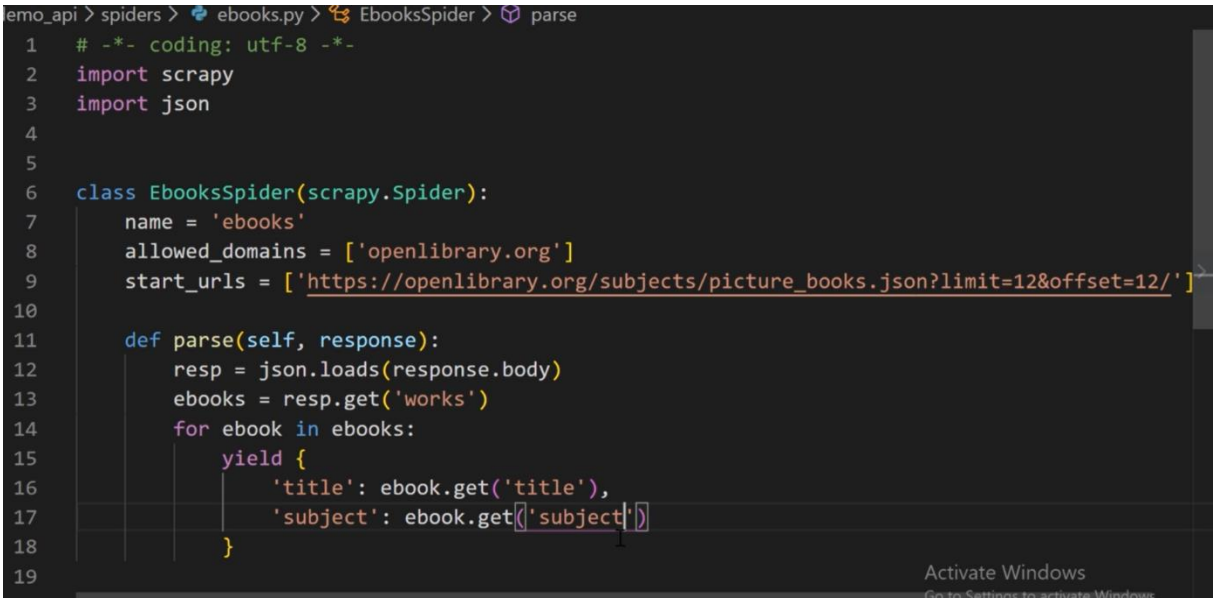


Spider’a Geçelim:

Bu bağlamda geçen örnekteki projenin içine yeni bir spider oluşturduk, ve initial request için aşağıdaki gibi api request’in url’ini browserdan kopyaladık.



Spider’ımızı aşağıdaki gibi oluşturduk:



- sonuçta initial request browser’dan elde edilen api url’e gönderiliyor.
- Response alınıyor ve response.body daha önceki gibi bir dictionary’e çeviriliyor, daha sonra bu dictionary içerisinden works’e ulaşıyoruz, biliyoruz ki bu works’ün içindeki her bir object bir book’a karşılık geliyor.
- Bu yüzden loop ile her book’u elde ediyoruz ve her book için title ve subject’i yield ediyoruz.

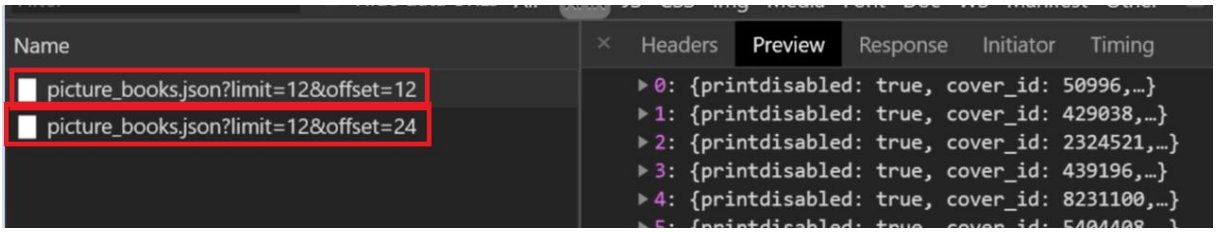
SCRAPING APIs PART 5 – PROJECT: PAGINATION

Tek bir sayfayı yukarıdaki başlıkta scrape ettik, şimdi yapacağımız şey diğer sayfalar için de ebookları elde etmek.

İleri geri tuşlarına her bastığımda yeni bir api url'e request gönderildiğini biliyorum, yapmam gereken tek şey bu api url'ler arasındaki ilişkiyi bulmak, önceki örnekte page no artıyordu o kadar, burada da o ilişkiyi çözersem:

- Next page var mı check et
- Yoksa programı sonlandır
- Varsa next url'i oluştur ve request yolla, response'u da aynı parse'da catch et diyebilirim.

Görüyoruz ki her page için offset 12'şer 12'şer artıyor, ilk sayfa için 0 idi, 2. Sayfa için 12, daha sonra 24 şeklinde gidiyor.



Öncelikle start_urls'in sonundaki offset'i ya 0'a eşitleriz ya da o kısmı tamamen sileriz yine 0'a denk gelir, yani ilk request'i ilk sayfaya gönderiyoruz.

```
# -*- coding: utf-8 -*-
import scrapy
import json


class EbooksSpider(scrapy.Spider):
    name = 'ebooks'

    INCREMENTED_BY = 12
    offset = 0

    allowed_domains = ['openlibrary.org']
    start_urls = ['https://openlibrary.org/subjects/picture_books.json?limit=12']
```

Ayrıca spider'a yukarıdaki gibi attribute'lar tanımlıyoruz, bunları next url'i elde etmek istediğimizde kullanacağız.

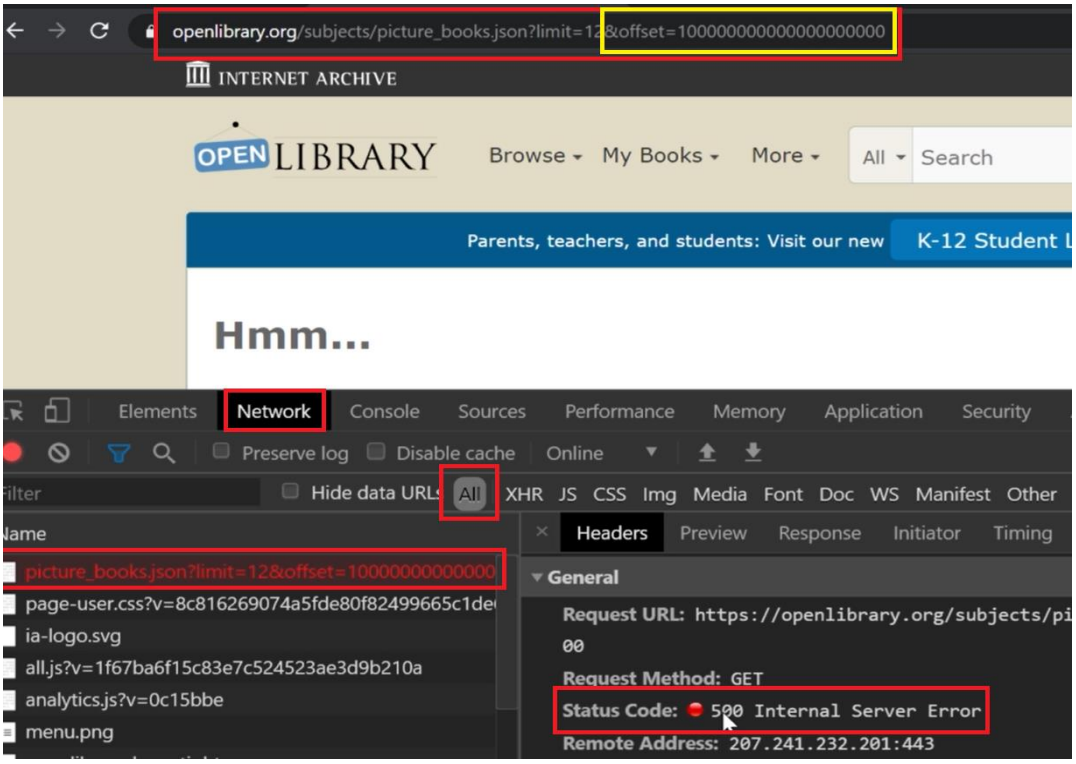
Aşağıda görüyoruz ki current page scrape edildikten sonra önce offset 12 artırılacak, ardından oluşturulan url'ye bir request gönderilip response yine parse methoduna yönlendirilecek.

```
allowed_domains = ['openlibrary.org']
start_urls = ['https://openlibrary.org/subjects/picture_books.json?limit=12']

def parse(self, response):
    resp = json.loads(response.body)
    ebooks = resp.get('works')
    for ebook in ebooks:
        yield {
            'title': ebook.get('title'),
            'subject': ebook.get('subject')
        }

    self.offset += self.INCREMENTED_BY
    yield scrapy.Request(
        url=f'https://openlibrary.org/subjects/picture_books.json?limit=12&offset={self.offset}',
        callback=self.parse
    )
```

Her şey iyi güzel ama, bu spider ne zaman duracak? Current page'den elde edilen Json objesi içinde has_next gibi bir property olmadığı için current page'in son sayfa olup olmadığını bilmiyoruz.

Bu yüzden şöyle bir trick yapacağız:

- Browser'a gidelim ve sayfaya çok yüksek bir offset değeri verelim mesela 100000000 gibi, ve daha sonra bu url'ye gönderilen request'in response'larını inceleyelim:



- Yukarıdaki gibi offsett'i yüksek bir url için gödnerilen request'lere bakıyoruz, kırmızı request'e gelip header'ına baktığımızda görüyoruz ki, status code 500: internal server error diyor.
- Yani eğer spider'ın içinde alınan response'un içinde status code 500 görürsek anlayacağız ki, sayfalar bitti, spider'ı kapatalım.

Şimdi spider'a dönelim, aşağıdaki kırımızılı satırları koda ekliyoruz böylece eğer response status'ü 500 olursa spider'ı sonlandır ve last page hatası ver diyoruz:

```
import scrapy
from scrapy.exceptions import CloseSpider
import json


class EbooksSpider(scrapy.Spider):
    name = 'ebooks'

    INCREMENTED_BY = 12
    offset = 0

    allowed_domains = ['openlibrary.org']
    start_urls = ['https://openlibrary.org/subjects/picture_books.json?limit=12']

    def parse(self, response):

        if response.status == 500:
            raise CloseSpider('Reached last page...')

        resp = json.loads(response.body)
        ebooks = resp.get('works')
        for ebook in ebooks:
            yield {
                'title': ebook.get('title'),
                'subject': ebook.get('subject')
```

Sonuçta böylelikte kodumuz kitaplar bitene kadar, yani status 500'ü görene kadar, scrape işlemine devam edecektir!