Multiple Pages Project

Bu projede aşağıdaki web sitesini scrape edeceğiz öncekilerden farklı olarak göreceğimiz şey birden fazla page'i scrape etmek olacak.

https://www.tinydeal.com/specials.html

Bu sayfadaki special offers'ı scrape etmek istiyoruz currently 36 sayfalık special offer ürünü yer alıyor biz hepsinin original price'ını ve discounted price'ını scrape etmek istiyoruz.

Scrapy projemize başlamadan önce, sayfanın javascript olmadan nasıl göründüğünü ve çalıştığını anlamakta fayda var ayrıca robot.txt'ye bakarak scrape etmeye iznimizin olmadığı sayfa varsa bunu da görebiliriz.

Bizim scrape edeceğimiz url yukarıdaki gibi olduğu için aşağıdaki gibi ulaşacağımız robots.txt dosyasının içinde specials keyword'ünü aratabilirim, bu örnek için bir restriction bulunmuyor, burada yazmamasına ragmen scrapy kullanırken blocklanma şansımız var, bunu da proxy rotating service kullanarak aşabiliriz daha sonra göreceğiz. Zaten robots.txt'de restriction olsa da bunu da bu şekilde çözebiliyoruz sanırım.

https://www.tinydeal.com/robots.txt

Javascript'i disable ederek sayfayı görmek için:

F12 → CTRL+SHIFT+P → Type javascript →Disable javascript→Refresh

Ürün fotoğraflarının görünmediğini görüyoruz ama zaten bunlara ihtiyacımız yok. Pagination'ı denediğimizde ise istediğimiz şekilde next butonu ile sıradaki page'e geçebildiğimizi görüyoruz, pagination için bu next butonunu kullanacağız.

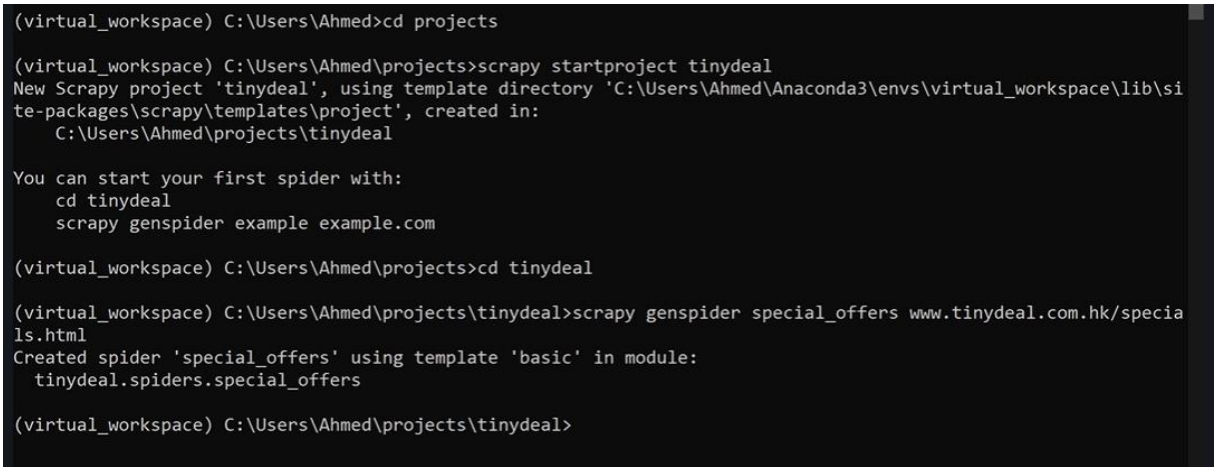
Yeni proje ve spider yarat:

Current directory'i proje klasörü olarak ayarladıktan sonra

scrapy startproject tinydeal diyerek tinydeal ismindeki project'imizi yaratmış oluyoruz.

Ardından cd tinydeal diyerek cd'yi ayarladıktan sonra

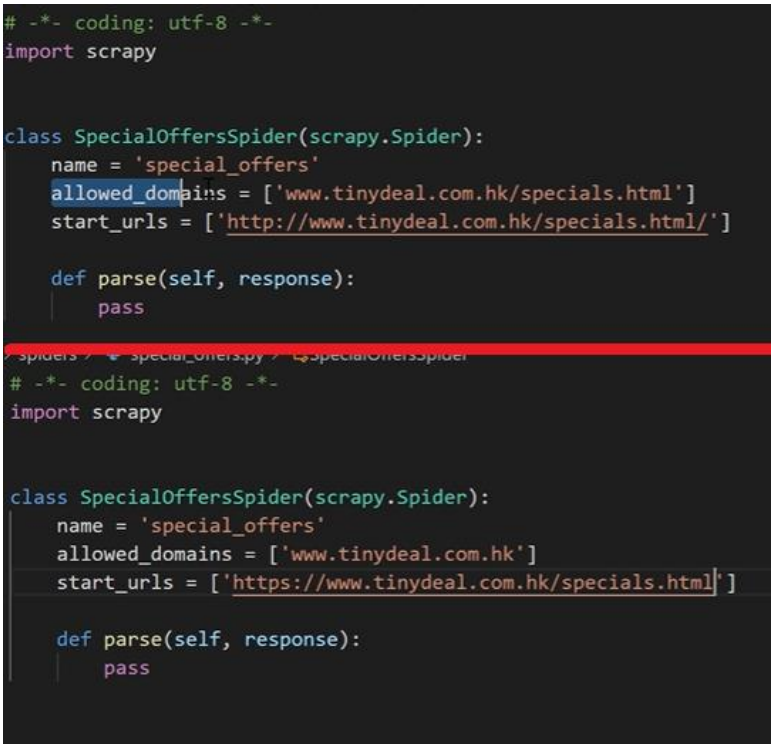
scrapy genspider special_offers www.tinydeal.com/specials.html diyerek ilgili sayfaya request göndrecek bir spider yaratıyoruz, burada baştaki http protokolünü sildik, çünkü zaten spider otomatik olarak yaratacak.



Fotoğraflarda site .com.hk olarak görünüyor bu değişti .com.hk yerine .com var olarak düşün.

Spider'ı birazcık değiştir:

VS Code içinden file>open folder ile projeyi açtıktan sonra special_offers.py spider'ını açarız, ilk hali aşağıdaki ilk ss'de görünüyor.



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


class SpecialOffersSpider(scrapy.Spider):
    name = 'special_offers'
    allowed_domains = ['www.tinydeal.com.hk/specials.html']
    start_urls = ['http://www.tinydeal.com.hk/specials.html/']

    def parse(self, response):
        pass
```

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


class SpecialOffersSpider(scrapy.Spider):
    name = 'special_offers'
    allowed_domains = ['www.tinydeal.com.hk']
    start_urls = ['https://www.tinydeal.com.hk/specials.html']

    def parse(self, response):
        pass
```

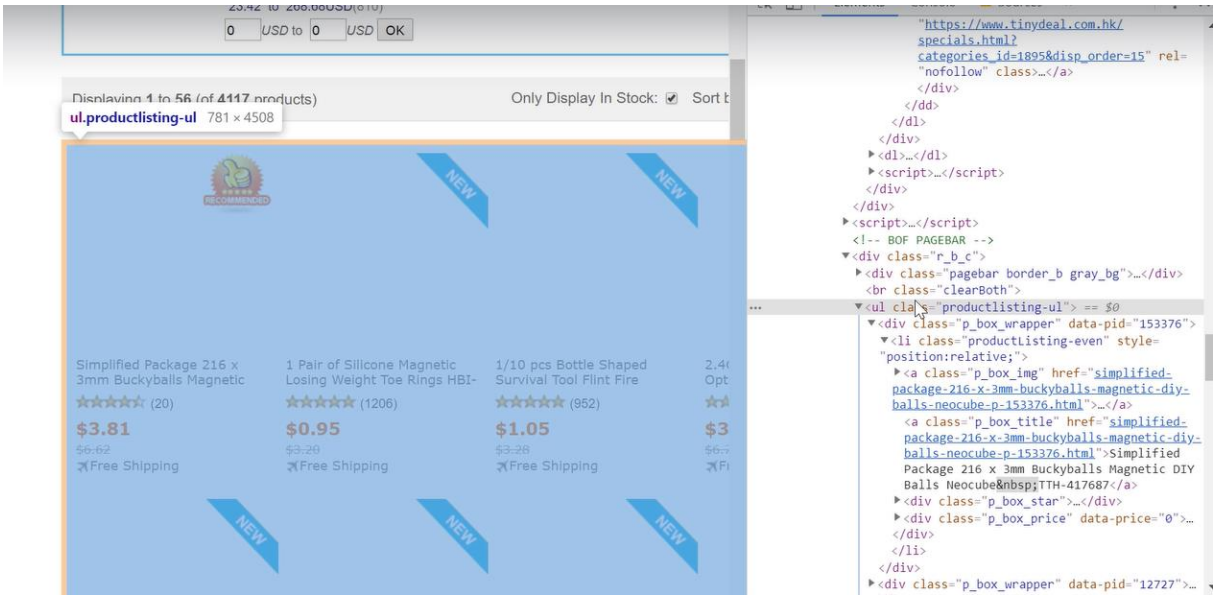
Burada yukarıdaki değişiklikler yapıldı yani, allowed domains genişletildi, ve starts url'nin sonundaki / silindi bu zaten otomatik konuluyor.

Sonuç olarak projemizi ve spider'ımızı initiate etmiş olduk!

Building the Spider 1

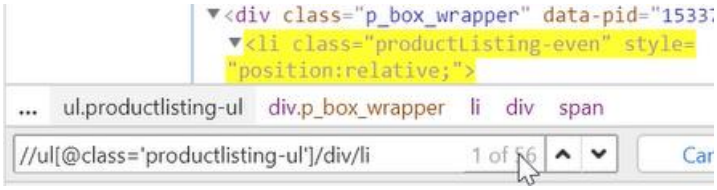
İlk yapılacak iş, spider kodunu düzenleyip, ilk sayfadaki ürünlerin hepsini scrape etmek.

Öncelikle sayfaya inspector'dan bakıyoruz ve ürünlerin html sayfada nasıl konumlandığını anlamaya çalışıyoruz:



Tüm ürünler bir ul altındaki bir div class=p_box_wrapper içinde konumlanmış, tek tek elemanlara ulaşmak için daha önce yaptığımız gibi bu yapıyı baz alarak xpath expressionlarımızı oluşturuyoruz.

Sonuçta aşağıdaki gibi bir expression ile bir sayfadaki 56 adet ürünü seçebildik, her ürün bir li elementi içinde yer alıyor, li'nin içinde ürünle ilgili header,original price, discounted price gibi bilgiler yer alıyor.



Xpath expression'ımızın çalıştığını gördükten sonra bu expression'ı spider içinde response'u scrape etmek için kullanalım:

Temelde yukarıdaki xpath expression'ı response.xpath() içerisinde kullanırsak 56 tane selector objesinden oluşan bir liste elde edeceğimizi biliyoruz her selector bir li elementini temsil edecek.

O zaman bu liste içinde dönüp her bir product için aşağıdaki özellikleri yield edebilirim.

Bu özellikleri elde etmek için daha önce yaptığımız gibi product selector'ı üzerinde tekrar xpath çalıştırdık.

Her bir xpath yine sayfada inspector'a bakılarak oluşturuldu, burada url xpath ifadesinin dışına response.urljoin metodunun geldiğine dikkat et, çünkü html içindeki link relative link olarak döndürülüyor, bu linkin başına original request linkini eklemek için response.urljoin() methodunu kullandık bunu daha önce de yaptımıştık.

```
class SpecialOffersSpider(scrapy.Spider):
    name = 'special_offers'
    allowed_domains = ['www.tinydeal.com.hk']
    start_urls = ['https://www.tinydeal.com.hk/specials.html']

    def parse(self, response):
        for product in response.xpath("//ul[@class='productlisting-ul']/div/li"):
            yield {
                'title': product.xpath(".//a[@class='p_box_title']/text()").get(),
                'url': response.urljoin(product.xpath(".//a[@class='p_box_title']/@href").get()),
                'discoutned_price': product.xpath(".//div[@class='p_box_price']/span[1]/text()").get(),
                'original_price': product.xpath(".//div[@class='p_box_price']/span[2]/text()").get()
            }
```

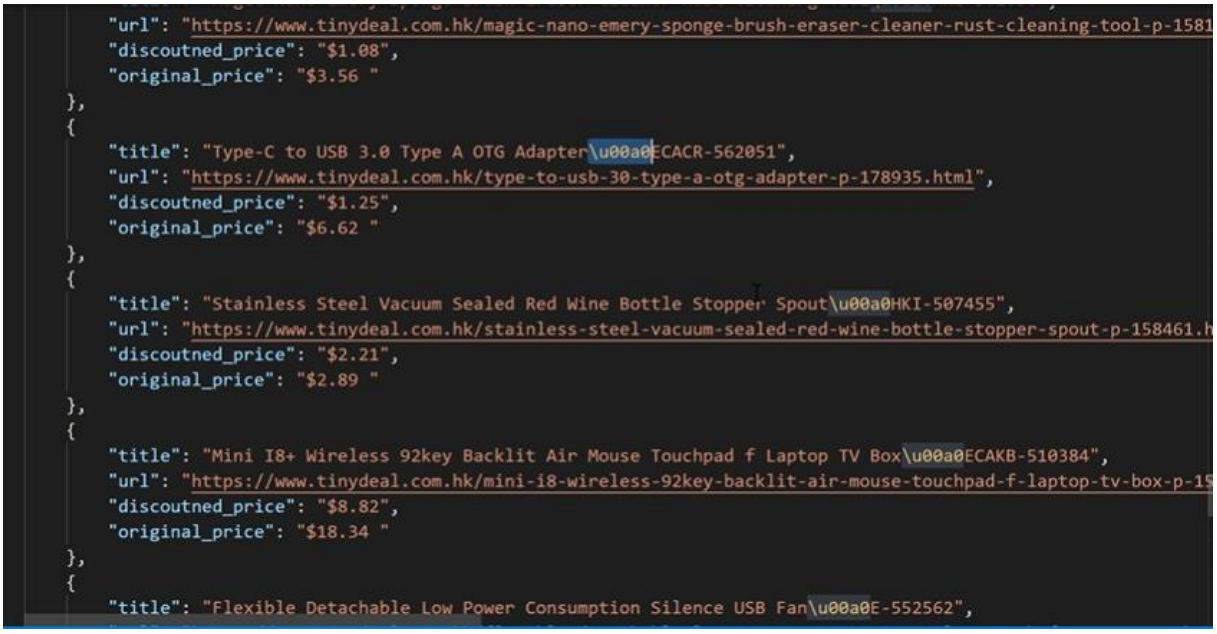
Sonuçta tek bir sayfayı scrape eden spider kodu yukarıdaki gibi hazırlanmış olur.

Running the spider above and Unicode Encoding Issue

scrapy crawl special_offers -o dataset.json

kodu ile yukarıdaki spider'ı çalıştırdık, ve sonuçta yield edilen verilen dataset.json file'ında kaydedildi.

Bu file'I açtığımızda şöyle bir görüntü görüyoruz:



Scrapy default olarak json file'ların encoding'i için utf8 kullanmadığı için yukarıdaki encoding issue ile karşılaşıyoruz, bunu düzeltmek kolay:

Proje içerisindeki settings.py file'ını açıyoruz daha sonra kodun en altına:

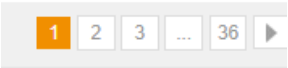
FEED_EXPORT_ENCODING = 'utf-8'

Satırını ekliyorum, kaydediyorum, spider'ı bir daha çalıştırdığımda yeni oluşan json dosyasında encoding sorunu çözülmüş olacak.

Dealing with Pagination

Önceki kısımda tek bir sayfa'yı nasıl scrape ettiğimizi gördük, bu videoda ise tüm sayfalardan nasıl scraping yapacağımızı göreceğiz.

Bir sayfadan diğerine navigate etmek için bir yöntem şu olabilir, geçeceğimiz page linkini elde edip bu linke request yollamak, ancak bu çok iyi bir yaklaşım değil çünkü total sayfa sayısı değiştiğinde bu approach işlevsiz kalacak.



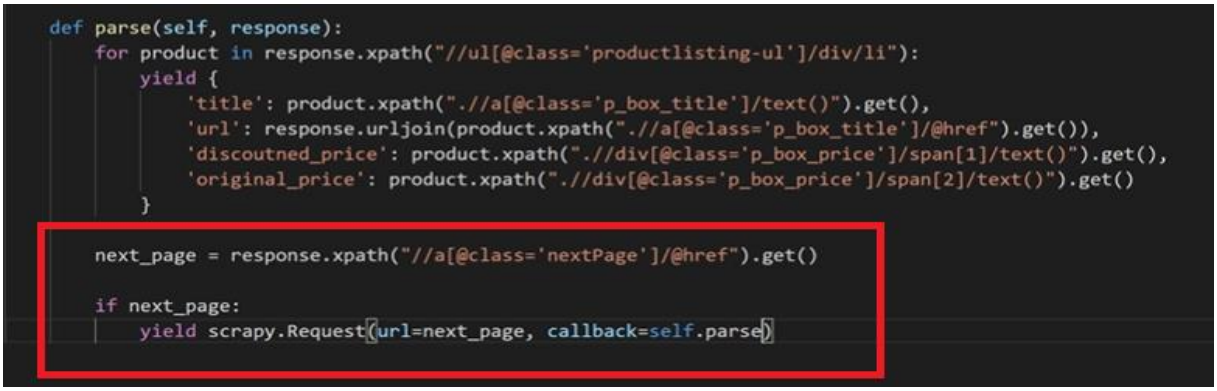
Bu yüzden navigation işlemini next butonunu kullanarak yapacağız böylece sayfa sayısından bağımsız olarak, bu butonla navigation'ı sağlayabileceğiz.

Dolayısıyla yapılacak iş şu: ilk sayfayı scrape et, daha sonra next page button'ın olup olmadığına bak, eğer next page butonu varsa bu buton elemanının içinden linki çek ve ilgili sayfaya request gönder ve yeni sayfada aynı işlemleri next page button kalmayana kadar yani tüm sayfalar için tekrarla.

Inspector yardımıyla next page butonunun içindeki linke nasıl ulaşabileceğimize bakıyoruz:

Bu link nextpage class'lı bir a elemanının içindeki href attribute'unda tutuluyor.

O halde spider'ımıza dönüp for loop'un dışına yani ilgili sayfadaki productları scrape ettikten sonrasına aşağıdaki satırları ekliyorum.



```
def parse(self, response):
    for product in response.xpath("//ul[@class='productlisting-ul']/div/li"):
        yield {
            'title': product.xpath(".//a[@class='p_box_title']/text()").get(),
            'url': response.urljoin(product.xpath(".//a[@class='p_box_title']/@href").get()),
            'discoutned_price': product.xpath(".//div[@class='p_box_price']/span[1]/text()").get(),
            'original_price': product.xpath(".//div[@class='p_box_price']/span[2]/text()").get()
        }

    next_page = response.xpath("//a[@class='nextPage']/@href").get()

    if next_page:
        yield scrapy.Request(url=next_page, callback=self.parse)
```

Burada yapılan ilgili sayfadaki productları scrape ettikten sonra, nextpage linkini scrape edip eğer bu link varsa ilgili linke bir request göndermek ve bu request'in response'unu da tekrar aynı parse methodu ile yakalamak, yani aynı işlemleri her sayfa için tekrarlamak.

Spoofing Request Headers

Browser'ın gönderdiği request'ler ile scrapy'nin gönderdiği request'ler arasında fark var ve bu fark botumuzun banlanmasına yol açabilir.

Şimdi bu requestlerin farklarını anlayalım, daha sonra scrapy ile gönderilen request'leri manipüle edip sanki browser'dan gönderiliyormuş gibi yapacağız, böylece banlanmayı engelleyeceğiz.

Request farkını anlamak için scrapy shell kullanalım:

```
C:\Users\Ahmed\projects\tinydeal>scrapy shell "https://www.tinydeal.com.hk/specials.html"
```

Yukarıdaki gibi, ilgili website sayfası için bir scrapy shell başlatalım, direkt olarak ilgili url'i fetch etmiş olduk yani request yollamış olduk.

Sonucunda aşağıdaki gibi bir ekranla karşılaşıyoruz.

```
[s] Available Scrapy objects:
[s]   scrapy     scrapy module (contains scrapy.Request, scrapy.Selector, etc)
[s]   crawler    <scrapy.crawler.Crawler object at 0x0000014704145348>
[s]   item       {}
[s]   request    <GET https://www.tinydeal.com.hk/specials.html>
[s]   response   <200 https://www.tinydeal.com.hk/specials.html>
[s]   settings   <scrapy.settings.Settings object at 0x0000014704145AC8>
[s]   spider     <SpecialOffersSpider 'special_offers' at 0x1470449a748>
[s] Useful shortcuts:
[s]   fetch(url[, redirect=True]) Fetch URL and update local objects (by default, redirects are followed)
[s]   fetch(req)                  Fetch a scrapy.Request and update local objects
[s]   shelp()           Shell help (print this help)
[s]   view(response)    View response in a browser
In [1]:
```

Available scrapy objects başlığı altında bir request objesi görülüyor, ve bu objenin GET olarak set edildiği görülüyor. Daha önce de söylediğimiz gibi hem GET hem de POST requestlerden söz etmek mümkün burada get request kullanılmış.

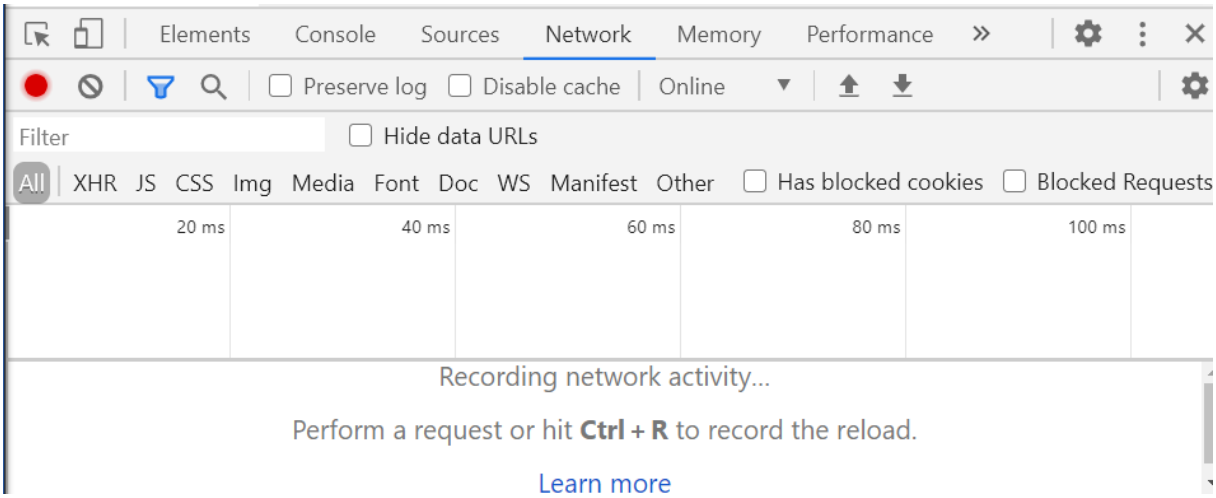
request.headers veya response.request.headers diyerek sayfaya gönderilen request'in headers'ını görebiliriz:

```
Out[1]:
{b'Accept': b'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
 b'Accept-Language': b'en',
 b'User-Agent': b'Scrapy/1.6.0 (+https://scrapy.org)',
 b'Accept-Encoding': b'gzip,deflate'}

In [2]: response.request.headers
Out[2]:
{b'Accept': b'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
 b'Accept-Language': b'en',
 b'User-Agent': b'Scrapy/1.6.0 (+https://scrapy.org)',
 b'Accept-Encoding': b'gzip,deflate'}

In [3]:
```

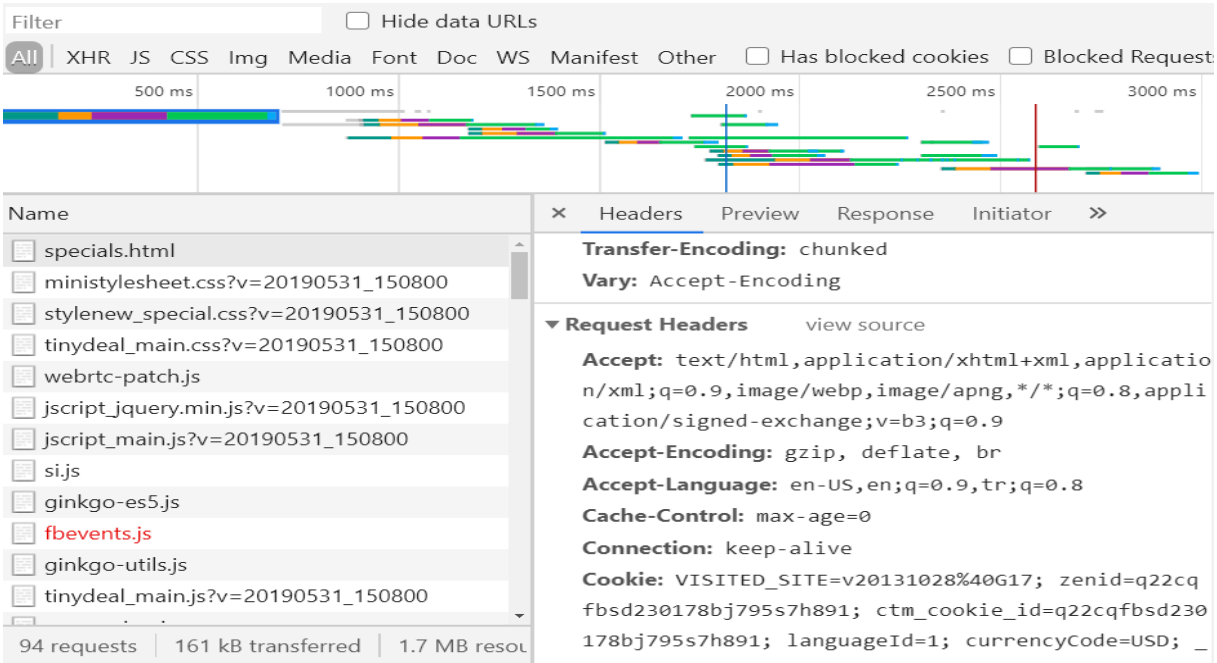
Bu komut output olarak, request ile gönderilen header'ları bir dictionary içerisinde verir, bu header'lardan önemli olan User-Agent bu header'ın içeriği request'i kimin gönderdiği ile ilgili bilgi taşıyor, şuanda bu değerin Scrapy/1.6.0 olarak set edildiğini görüyoruz, websitesi bu bilgiye dayanarak ilgili request'i ignore edebilir.

Bu header'ı override etmenin bazı yollarını göreceğiz. Ancak önce, browser'dan gönderilen bir request'e bakalım ve farkı anlayalım:

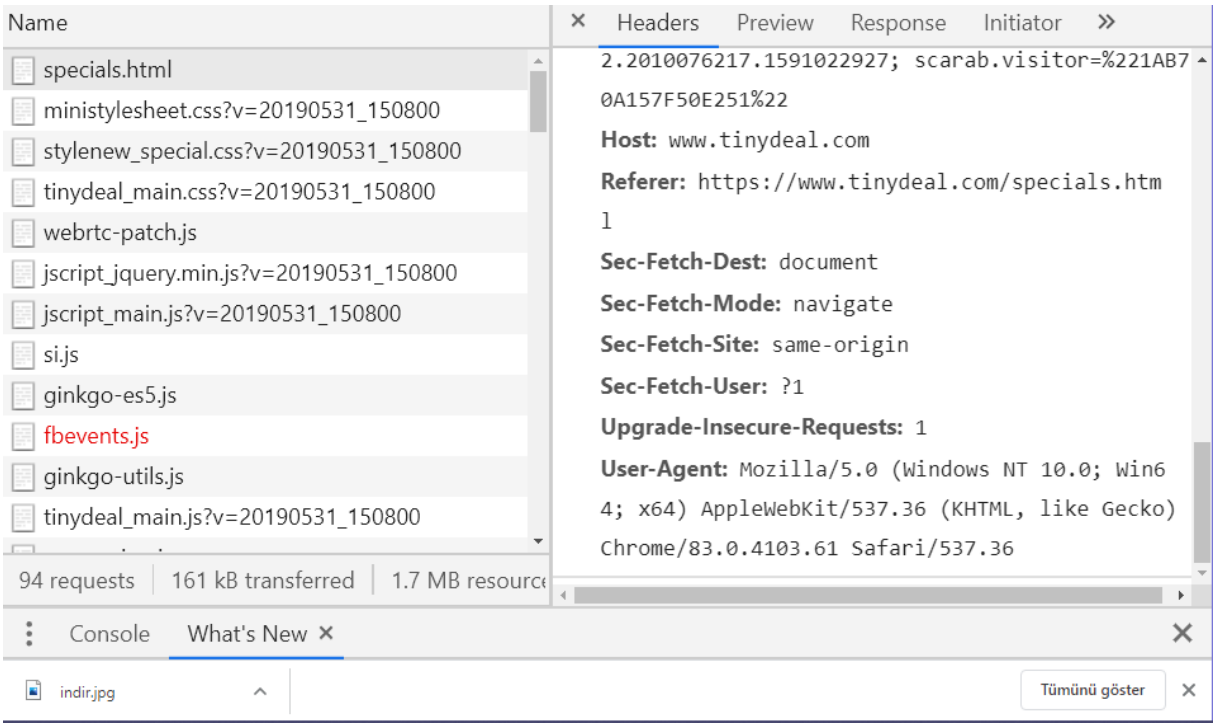
Bu amaç için developer tools'u kullanacağız, browser'da site linkindeyken F12'ye basıp inspector'ı açtıktan sonra Network tabına geliyoruz:



Şuanda sayfa boş, tüm requestlerin kaydedilmesi için sol üstteki All'ın seçili olduğundan emin olduktan sonra CTRL+R ile sayfayı yeniliyorum, böylece sayfa yüklenirken browser'ın gönderdiği tüm request'ler burada gösterilecek:



Specials.html request'inin headers'ına yukarıdaki gibi bakabiliyorum. User-Agent header'ını aşağıda görebiliyoruz,



Gördüğümüz gibi chrome request gönderirken böyle bir User-Agent header'ı kullanılıyor, safari veya firefox browserları farklı bir User-Agent header'ı ile request gönderirler.

Şimdi bu User-Agent header'ının içeriğini kopyalıyorum ve amacım scrapy'nin gönderdiği request'lerde bu User-Agent header'ı kullanması.

Bunu sağlamanın birden fazla yolu var, ilk ve en kolay yol, settings.py dosyasının içeriğini değiştirmek:

Bu dosyanın içeriğinde zaten USER_AGENT satırı bulunuyor ancak commentde.

```
# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'tinydeal (+http://www.yourdomain.com)'
```

Bu satırı uncomment edip değerini yukarıda kopyalanan değer olarak atıyoruz, böylece scrapy requestlerde bu User-Agent header'ı kullanacak.

Ancak bu pek iyi bir yol değil çünkü bu yöntemle sadece user_agent header'ını değiştirebiliyorum belki aynı zamanda başka bir request header'ı da değiştirmek istedim, bunu nasıl yaparım?

Eğer birden fazla header'ı override etmek istersek, kullanabileceğimiz iki yol var, ilk yol yine settings.py içerisindeki DEFAULT_REQUEST_HEADERS dictionary'sini kullanmak. Bunu kullanacaksak USER_AGENT değişkenini comment'e almayı unutma.

```
40
41  # Override the default request headers:
42  #DEFAULT_REQUEST_HEADERS = {
43  #   'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
44  #   'Accept-Language': 'en',
45  #}
46
```

Bu dictionary'i uncomment edip, içerisine key olarak aşağıdaki gibi User-Agent tanımlıyorum ve value olarak da yukarıda browserdan kopyalanan User-Agent value'sunu yazıyorum. Bunun yanında istersem diğer headerları da değiştirebilirdim.

```
# Override the default request headers:
DEFAULT_REQUEST_HEADERS = {
    'User-Agent': '
}
```

Multiple override için son yöntem ise spider içerisinde aşağıdaki gibi değişiklikler yapmak:

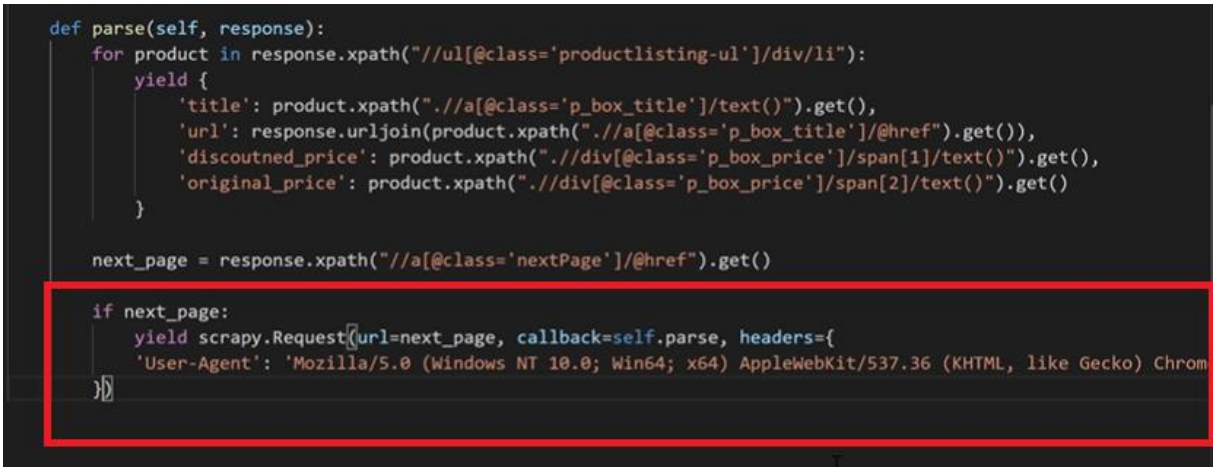
- Spider class'ının start_requests() adındaki methodunu override edeceğiz.

```
class SpecialOffersSpider(scrapy.Spider):
    name = 'special_offers'
    allowed_domains = ['www.tinydeal.com.hk']
    start_urls = ['https://www.tinydeal.com.hk/specials.html']

    def start_requests(self):
        yield scrapy.Request(url='https://www.tinydeal.com.hk/specials.html', callback=self.parse, headers={
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
        })

    def parse(self, response):
        for product in response.xpath("//ul[@class='productlisting-ul']/div/li"):
            yield {
```

- Yukarıda görüldüğü gibi bu method override edilir, yani bu methodu override ederek söylediğimiz şey şu sanırım: bu spider başlatılırken, method içine girilen url'ye, verilen header dictionary ile bir request yolla ve response'u da parse methoduna yolla.
- Bu method zaten vardı ve callback default olarak parse methoduydu ve ilgili url'yi de start_urls içine yazılan değer olarak alıyordu.

- Biz methodu override edip içine de url'mizi belirttiğimize göre artık start_urls attribute'una ihtiyacımız kalmadı, bu satırı silebiliriz.

Bu sayede, spider ilk çalıştığında yollanacak initial request'in methodunu override etmiş olduk ve parametreleri kendimiz verdik, iyi ama biz initial request'in response'unu alıp bu response'u scrape ettikten sonra parse içerisinde tekrar request yolluyoruz, bu request'ler bu parametreleri kullanmıyor, o yüzden o request methodunu da yeniden aşağıdaki gibi düzenlemeliyiz:



Tamam artık spider içinde gönderilen request'lerin hepsi belirtilen User-Agent header'ını kullanıyor, yani aynı bir choreme'dan request gönderilmiş gibi oluyor.

Son olarak bunu test etmek için aşağıdaki gibi yield içinde User-Agent headerını da yazdırıyorum ki emin olayım, bu user-agent header'ın değeri ilgili response'un içindeki request objesinden alınıyor.

Sonuçta User-Agent header'ını değiştirmenin üç farklı yolunu görmüş olduk!