A web scrapper case

Rafael Campos Nunes

19 de Abril de 2019

1 Requirements

The objective of the web scrapper is to list every product with the following characteristics encoded as a JSON:

1. Name of the product
2. Link to one or more of its images
3. Price of the product

The page from which all of this information will be scrapped is: `https://nerdstore.com.br/categoria/especiais/game-of-throne/s`

2 Inspection

When inspecting the page I found that the elements are enclosed in the following structure:

```
<ul class="products ...">
  <li class="... product ...">
    <a class="woocommerce-LoopProduct-link woocommerce-loop-product
      <img class="attachment-woocommerce_thumbnail ..." src="">
      <img class="attachment-woocommerce_thumbnail ..." src="">
      <h2 class="woocommerce-loop-product__title">
        <span class="ellip">text ellipsed?</span>
        <span class="ellip-line">text ellipsed 2?</span>
      </h2>
      <span class="price">
        <span class="woocommerce-Price-amount amount">
          <span class="woocommerce-Price-currencySymbol">R$</span>
          "49,90"
        </span>
      </span>
    </a>
  </li>
  ...
</ul>
```

By inspection I see that a product may or may not have more than one image and that are specific classes to grab the specified element, though this shouldn't be enough I think because they (the developers) might change on the long run. I should return a JSON that looks somewhat like this:

```
o = {
   'name'  : '',
   'img'   : ['', '...', ''],
   'price' : '44,90'
}
```

The price comes with a currency tag and that should be important to have but it's not asked by the head hunters. What I should do now is to take the big block of products that are contained inside the *ul* tag with the *products* class and work from there.

3 Solution

This section of the document explains a thing in chronological order and if something doesn't make sense when you read it might make sense after a few paragraphs. It is just how my mind went when solving this problem. I hope that you, reader, to comprehend that.

Well, I'll be working with scrapy and django. First I will write the scrapper because I think that's the most important thing to do and then I write the interface to access all of the things.

Scrapy is quite a handy tool, I could use to scrapy the page with a shell that they created. Using expressions such as the following I can pick every image that is inside a product item.

```
response.css('img.attachment-woocommerce_thumbnail::attr(src)').get
```

These expressions below makes the job complete without much work:

```
response.css('h2.woocommerce-loop-product__title::text').get
response.css('span.price > span::text, span.woocommerce-Price-
currencySymbol')
```

Now that I have everything I might as well implement it right? mm, not yet. The images should be returned in an array because there might be more than one image per product. I think that this could work:

```
response.css('li.product > img.attachment-woocommerce_thumbnail::att
```

Lets proceed to that then. I there where more pages there's a bit more of work that I should do but anyway, this will suffice. Some time later I got it working, I had to modify the expressions a bit but the scrapper is returning very good results, still not perfect quite yet. I still need to scrap over all the images inside a product item and the price isn't being returned with the currency symbol.

The second image on a product has different class names, how dull I was to think it was the same not even looking for it, I will have to add a simple thing and it will work.

```
response.css('img.secondary-image:attr(src)').getall()'
```

I guess it should be more easy to select every image inside it with xpath instead. Lets do it now. No success after quite a while trying, as it is ready I will proceed to other parts of the task as requested and then I comeback if I have time.

The next step is to integrate this with Django. Well, I kind of fixed the problem of getting the images, kudos for me! The structure of the scrapper was also changed and it's noted on the $6a338c$ commit.

Right, I kind of did the integration but is quite clumsy. I don't like the way this stuff is integrated, the scraper code is not fully integrated and it looks more like a glue than an integration. A way I would do this integration is to pipe the applications (just like unix tools), the json output of the scraper would be used by another application to fill the database and Django would be only responsible for displaying the information.

4 Hard time

As I didn't have a proficiency on the frameworks used on this challenge I was quite slow to implement everything. I ended up with a application which looks like a duck with eyes of a cat and legs of a dog, a chimera.

I also was unable to detail a bit more of the development as I had to full fill other objectives of the challenge. I didn't see that I need to write a README and I was writing this already.