python barcelona

Basic web scraping with Beautiful Soup

2023-05-31

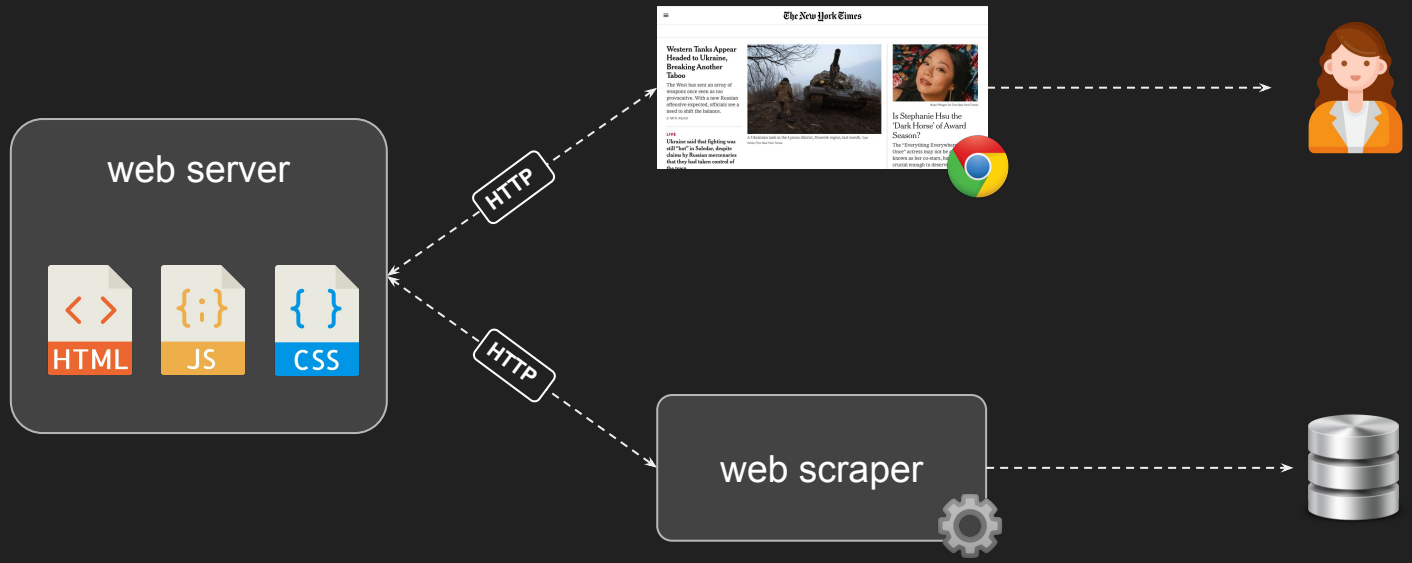
Noé Casas

Speaker

Noé Casas

- PhD in artificial intelligence for machine translation.
- 15 year of experience as software dev, 2 years as data scientist, 5 years with NLP and deep learning.
- Solo founder of Langtern, a language learning app for teachers and students.

What is web scraping?



Web scraping use cases

- Build databases / indexes from website information:
  - Information aggregation services.
  - Search engines.
  - Machine learning datasets.
- Web testing
- …

Types of web pages from the scraper point of view

- Static web pages / server-side rendered pages:

  → the client receives HTML code directly.

- Dynamic web pages / single-page applications:

  1. The client receives javascript code.
  2. The client executes the javascript code.
  3. The javascript code requests data from the server API.
  4. The javascript code creates the page dynamically based on the retrieved data.

Anti-scraping protections

- Dynamic content (javascript)
- Legal: terms of service (contract), copyright
- Rate limiting checks
- IP range blocks
- User agent checks
- Other header checks (e.g. Referrer)
- Request regularity checks (interval, IP, user agent)
- Captchas

Note: differences between dev / staging / production can lead to different behaviors (different IPs, different scraping time, different access time ranges)

Advanced techniques to bypass protections

- Dynamic content (javascript):
  - Mimic API calls by web page:
    - Analyze API calls with DevTools (Network tab).
    - Use in Python directly the discovered backend endpoints (e.g. with **`requests`**).
  - Browser-based scraping with **`selenium`** or **`splinter`**
    - They open a real web browser.
    - They allow to interface with the web site from Python.
- Rate limit checks: use [random] wait times.
- IP checks: use proxies.
- Captchas: use AI-driven solutions to break them.

Basic scraping of static websites in Python

- Requests (download HTML) + Beautiful Soup (parse HTML)
- Scrapy:
  - Larger scope:
    - Download HTML
    - Parse HTML
    - Scheduling
    - Parallelism
    - Pipelines
    - …
  - Framework structure (rather than a library).

Basic web scraping with requests + beautifulsoup

1. Analyze web page with Chrome DevTools
2. Get web pages with **`requests`**
3. Parse HTML with **`beautifulsoup (bs4)`**
4. ???
5. Profit

Getting the web pages: **`requests`** (1/3)

```
> pip install requests
```

```
import requests


try:
   url = "https://..."
   page = requests.get(url)

   ...

except requests.exceptions.RequestException:
   pass  # TODO: handle error
```

Getting the web pages: **`requests`** (2/3)

```
import requests


HTTP_TIMEOUT = 5  # seconds
USER_AGENT = "..."


try:
   url = "https://..."
   page = requests.get(
       url,
       timeout=HTTP_TIMEOUT,
       headers={'User-Agent': USER_AGENT},
       allow_redirects=True,
   )

   ...

except requests.exceptions.RequestException:
   pass  # TODO: handle error
```

Getting the web pages: **`requests`** (3/3)

```
import requests
from requests.adapters import HTTPAdapter, Retry

USER_AGENT = ...
HTTP_TIMEOUT = ...

session = requests.Session()
session.mount('https://', HTTPAdapter(max_retries=Retry(total=5, backoff_factor=0.1)))

url = ...

try:
  page = session.get(
      url,
      allow_redirects=True,
      timeout=HTTP_TIMEOUT,
      headers={'User-Agent': USER_AGENT},
  )
except requests.exceptions.Timeout:
  pass  # TODO: handle error

except requests.exceptions.RequestException:
  pass  # TODO: handle error
```

Parsing static web pages: **beautifulsoup (bs4)**

```
> pip install beautifulsoup4
```

```
import bs4


soup = bs4.BeautifulSoup(page.content, 'html.parser')

panel_body_elem = soup.find('div', class_='list_article')

if not panel_body_elem:
   return

for article_row in panel_body_elem.find_all('li'):

   title_elem = article_row.find('div', class_='title')

   link_elem = title_elem.find('a')

   url = BASE_URL + link_elem['href']

   ...
```

Live coding session

Goal:

Scrape website of a podcast that has audio files and transcripts to index them for Langtern's online content selection.

Hope you enjoyed

contact@noecasas.com

Langtern

https://langtern.com

Bonus

Legal precedents (USA)

- HiQ Labs v. LinkedIn case (2019 - 2022)
  - Publicly accessible pages can be scraped.
  - Breach of contract (Terms of Service) accepted by HiQ Labs staff.
- Associated Press v. Meltwater U.S. Holdings (2013)
  - Meltwater scraping was Ok under the “fair use” US doctrine.
  - Meltwater violated AP’s copyright.
- Facebook v. Power Ventures (2009 - 2011)
  - Power ventures created a “super social network” by scraping other social networks.
  - Power’s scraping of user data was Ok, because they had permission from the users, who have the rights over such data, not Facebook.

Disclaimer: I am not a lawyer. Here I present info that can be found online. Check these matters with a lawyer.

Legal precedents (United Kingdom)

- PR Consultants Association (+ Meltwater) v Newspaper Licensing Agency (2011 - 2013)
  - Parallel to US case Associated Press v. Meltwater U.S. Holdings, but opposite result.
  - Website indexing of copyrighted material is copyright infringement as an entire copy of the infringed work is made and stored for future use.

Disclaimer: I am not a lawyer. Here I present info that can be found online. Check these matters with a lawyer.

Legal precedents (European Union)

- Ryanair v. PR Aviation (2015)
  - Ryanair data was not subject to copyright.
  - Breach of contract (Terms of Service) accepted by PR Aviation.

Disclaimer: I am not a lawyer. Here I present info that can be found online. Check these matters with a lawyer.