Scrapy Fake User Agents: How to Manage User Agents When Scraping

After you've learned the basics of web scraping (how to send requests, crawl websites and parse data from the page), one of the main challenges we face is avoiding our requests getting blocked.

The two keys we can achieve this is be using proxies and managing the user-agents we send to the website we are scraping.

In this guide, we will go through:

- What Are User-Agents & Why Do We Need To Manage Them?

- How To Set A User Agent In Scrapy
- How To Rotate User Agents
- How To Manage Thousands of User Agents

First, let's quickly go over some the very basics.

What Are User-Agents & Why Do We Need To Manage Them?

User Agents are strings that let the website you are scraping identify the application, operating system (OSX/Windows/Linux), browser (Chrome/Firefox/Internet Explorer), etc. of the user sending a request to their website. They are sent to the server as part of the request headers.

Here is an example User agent sent when you visit a website with a Chrome browser:

```
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.82 Safari/537.36
```

When scraping a website, you also need to set user-agents on every request as otherwise the website may block your requests because it knows you aren't a real user.

In the case of Scrapy. When you use Scrapy with the default settings, the user-agent your spider sends is the following by default:

```
Scrapy/VERSION (+https://scrapy.org)
```

This user agent will clearly identify your requests as coming from a web scraper, so the website can easily block you from scraping the site.

That is why we need to manage the user-agents Scrapy sends with our requests.

How To Set A Fake User-Agent In Scrapy

There are a couple of ways to set new user agent for your spiders to use.

1. Set New Default User-Agent

The easiest way to change the default Scrapy user-agent is to set a default user-agent in your `settings.py` file.

Simply uncomment the `USER_AGENT` value in the `settings.py` file and add a new user agent:

```
## settings.py

USER_AGENT = 'Mozilla/5.0 (iPad; CPU OS 12_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Mobile/15E148'
```

---

2. Add A User-Agent To Every Request

Another option is to set a user-agent on every request your spider makes by defining a user-agent in the headers of your request:

```
## myspider.py

def start_requests(self):
    for url in self.start_urls:
        return Request(url=url, callback=self.parse,
                       headers={"User-Agent": "Mozilla/5.0 (iPad; CPU OS 12_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Mobile/15E148"})
```

Both of these options work, however, you will have the same user-agent for every single request which the target website might pick up on and block you for. That is why we need to have a list of user-agents and select a random one for every request.

How To Rotate User Agents

Rotating through user-agents is also pretty straightforward, and we need a list of user-agents in our spider and use a random one with every request we make using a similar approach to option #2 above.

```
## myspider.py

import random

user_agent_list = [
    'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/93.0.4577.82 Safari/537.36',
    'Mozilla/5.0 (iPhone; CPU iPhone OS 14_4_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0.3 Mobile/15E148 Safari/604.1',
    'Mozilla/4.0 (compatible; MSIE 9.0; Windows NT 6.1)',
    'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
```

```
Chrome/87.0.4280.141 Safari/537.36 Edg/87.0.664.75',
    'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/70.0.3538.102 Safari/537.36 Edge/18.18363',
]

def start_requests(self):
    for url in self.start_urls:
        return Request(url=url, callback=self.parse,
                       headers={"User-Agent": user_agent_list[random.randint(0,
len(user_agent_list)-1)]})
```

This works but it has 2 drawbacks:

1. We need to manage a list of user-agents ourselves.
2. We would need to implement this into every spider, which isn't ideal.

A better approach would be to use a Scrapy middleware to manage our user agents for us.

How To Manage Thousands of Fake User Agents

The best approach to managing user-agents in Scrapy is to build or use a custom Scrapy middleware that manages the user agents for you.

You could build a custom middleware yourself if your project has specific requirements like you need to use specific user-agents with specific sites. However, in most cases using a off-the-shelf user-agent middleware is enough.

Developers have realised of user-agent middlewares for Scrapy, however, for this guide we will use ScrapeOps Fake User-Agent API as it is one of the best available.

ScrapeOps Fake User-Agent API

The ScrapeOps Fake User-Agent API is a **free user-agent API**, that returns a list of fake user-agents that you can use in your web scrapers to bypass some simple anti-bot defenses.

To use the **ScrapeOps Fake User-Agents API** you just need to send a request to the API endpoint to retrieve a list of user-agents.

You first need an **API key** which you can get by signing up for a free account here.

```
http://headers.scrapeops.io/v1/user-agents?api_key=YOUR_API_KEY
```

Example response from the API:

```
{
  "result": [
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15) AppleWebKit/605.1.15 (KHTML, like
Gecko) Version/13.0.5 Safari/605.1.15",
    "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/103.0.5060.53 Safari/537.36",
    "Mozilla/5.0 (Windows NT 10.0; Windows; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/103.0.5060.114 Safari/537.36",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/603.3.8 (KHTML, like
Gecko) Version/10.1.2 Safari/603.3.8",
    "Mozilla/5.0 (Windows NT 10.0; Windows; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/103.0.5060.114 Safari/537.36",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like
Gecko) Version/14.0 Safari/605.1.15",
    "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/103.0.5060.53 Safari/537.36",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/605.1.15 (KHTML, like
Gecko) Version/14.0 Safari/605.1.15",
    "Mozilla/5.0 (Windows NT 10.0; Windows; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/103.0.5060.114 Safari/537.36",
    "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/103.0.5060.53 Safari/537.36"
  ]
```

```
}
```

The best way to integrate the **Fake User-Agent API** is to create a Downloader middleware and have a fake user-agent be added to every request. Here is an example middleware you can use:

```
## middlewares.py

from urllib.parse import urlencode
from random import randint
import requests

class ScrapeOpsFakeUserAgentMiddleware:

    @classmethod
    def from_crawler(cls, crawler):
        return cls(crawler.settings)

    def __init__(self, settings):
        self.scrapeops_api_key = settings.get('SCRAPEOPS_API_KEY')
        self.scrapeops_endpoint = settings.get('SCRAPEOPS_FAKE_USER_AGENT_ENDPOINT',
'http://headers.scrapeops.io/v1/user-agents?')
        self.scrapeops_fake_user_agents_active =
settings.get('SCRAPEOPS_FAKE_USER_AGENT_ENABLED', False)
        self.scrapeops_num_results = settings.get('SCRAPEOPS_NUM_RESULTS')
        self.headers_list = []
        self._get_user_agents_list()
        self._scrapeops_fake_user_agents_enabled()

    def _get_user_agents_list(self):
        payload = {'api_key': self.scrapeops_api_key}
        if self.scrapeops_num_results is not None:
            payload['num_results'] = self.scrapeops_num_results
        response = requests.get(self.scrapeops_endpoint, params=urlencode(payload))
        json_response = response.json()
        self.user_agents_list = json_response.get('result', [])

    def _get_random_user_agent(self):
        random_index = randint(0, len(self.user_agents_list) - 1)
```

```
        return self.user_agents_list[random_index]

    def _scrapeops_fake_user_agents_enabled(self):
        if self.scrapeops_api_key is None or self.scrapeops_api_key == '' or
self.scrapeops_fake_user_agents_active == False:
            self.scrapeops_fake_user_agents_active = False
        self.scrapeops_fake_user_agents_active = True

    def process_request(self, request, spider):
        random_user_agent = self._get_random_user_agent()
        request.headers['User-Agent'] = random_user_agent
```

Note: This middleware example requires the installation of **Python Requests** via `pip install requests`.

And then enable it in your project in the `settings.py` file. Remembering to swap the `YOUR_PROJECT_NAME` for the name of your project (`BOT_NAME` in your `settings.py` file):

```
## settings.py

SCRAPEOPS_API_KEY = 'YOUR_API_KEY'
SCRAPEOPS_FAKE_USER_AGENT_ENABLED = True

DOWNLOADER_MIDDLEWARES = {
    'YOUR_PROJECT_NAME.middlewares.ScrapeOpsFakeUserAgentMiddleware': 400,
}
```

Or in the spider itself using the `custom_settings` attribute.

```
## your_spider.py

import scrapy
from demo.items import QuoteItem


class QuotesSpider(scrapy.Spider):
```

```
    name = "demo"
    start_urls = ["http://quotes.toscrape.com/"]

    ## Enable ScrapeOps Fake User Agent API Here
    custom_settings = {
        'SCRAPEOPS_API_KEY': 'YOUR_API_KEY',
        'SCRAPEOPS_FAKE_USER_AGENT_ENABLED': True,
        'DOWNLOADER_MIDDLEWARES': {
            'YOUR_PROJECT_NAME.middlewares.ScrapeOpsFakeUserAgentMiddleware': 400,
        }
    }

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

When activated, the **ScrapeOpsFakeUserAgentMiddleware** will download a list of the most common user-agents from the API and use a random one with every request, so you don't need to create your own list.

To see all the configuration options, then check out the docs here.

More Scrapy Tutorials

So that's why you need to use user-agents when scraping and how you can manage them with Scrapy.

Managing user-agents is only half the battle when it comes to not getting blocked whilst web scraping. The more important part is using proxies. If you want to learn how you can integrate proxies into your spiders then check out our Scrapy Proxy Guide here.

If you would like to learn more about Scrapy in general, then be sure to check out The Scrapy Playbook.