Studium licencjackie

Kierunek: Informatyka

Specjalność: Bazy danych i technologie internetowe

Michał Polakowski
Nr albumu: 203446
Karol Chmielewski
Nr albumu: 246668

Scraping w praktycznych aplikacjach webowych

Praca licencjacka
napisana w Instytucie Matematyki,
Fizyki i Informatyki
pod kierunkiem naukowym
dra Włodzimierza Bzyla

Gdańsk, 2019

Spis treści

1 Wprowadzenie **3**

2 Django **4**

2.1 Popularność 4

2.2 Modularność 5

2.2.1 Jedna funkcjonalość, jedna aplikacja 5

2.2.2 Gotowe paczki 6

2.2.3 Bierz co chcesz 6

2.3 ORM 6

3 Scraper **8**

3.1 Opis etapów 9

3.1.1 Etap 1 9

3.1.2 Etap 2 9

3.1.3 Etap 3 9

3.2 Dlaczego Scrapy 11

3.2.1 Selektory 11

Rozdział 1

Wprowadzenie

Celem niniejszej pracy jest ...

Rozdział 2

Django

Python jest aktualnie jednym z najpopularnijeszych języków programowania na świecie, a co za tym idzie liczba frameworków, które się na nim opierają jest dość pokaźna. Nasz wybór w kwestii obsługi backendu aplikacji padł na Django z kilku powodów, które chciałbym w tym rozdziale pokrótce przedstawić.

2.1 Popularność

Pierwszym czynnikiem wpływającym na nasz wybór była popularność frameworka. Django bez wątpienia jest jednym z najczęściej używanych pythonowych narzędzi do obsługi serwerowej strony aplikacji internetowych. Oprócz popularności wśród pythonowych frameworków jest on niewątpliwie również jednym z njaczęściej wybieranych w ogóle. Na jego popularność z pewnością wpływa podejście "batteries included", to znaczy koncepcja frameworka, który najważniejsze elementy współczesnych aplikacji internetowych ma niejako wbudowane. Możliwość obsługi frontendu poprzez wbudowany system template'ów, formularze tworzone bezpośrednio po stronie django i w prosty sposób renderowane na froncie, czy system autentykacji, który praktycznie nie wymaga modyfikacji (chyba, że ich potrzebujemy) - to wszystko również kierowało milionami użytkowników przy wyborze tego narzędzia do budowy ich projektów.

Z pewnością początki Django sięgające okolic 2006 roku pozwoliły na uformowanie się bardzo dużej społeczności, która sukcesywnie budowała, i wciąż buduje niezliczone ilości narzędzi wspomagających proces tworzenia i obsługi kolejnych aplikacji internetowych.

2.2 Modularność

Wymieniając w pierwszej części tego rozdziału powody popularności obiektu naszego zainteresowania, siłą rzeczy zahaczyłem również o powody, dla których Django stało się naszym wyborem jako silnik backendu budowanej przez nas aplikacji. Jednym z nich jest właśnie jego modularność - aspekt, który jest niejako bezpośrednią konsekwencją popularności. Mianowicie przez słowo modularność mam na myśli technikę budowania systemów, która kładzie nacisk na podzielenie poszczególnych funkcji programu w niezależne, wymienne moduły w taki sposób, aby dana funkcja mogła być w pełni wykonana przez dany moduł.

Modularność Django można zauważyć w kilku aspektach:

2.2.1 Jedna funkcjonalość, jedna aplikacja

Przy tworzeniu podstawowego schematu djangowej aplikacji zauważyć można, że cli frameworka tworzy strukturę folderów zbliżoną do następującej

```
projectname
├── projectname
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── app1
│   ├── models.py
│   └── views.py
└── app2
    ├── models.py
    └── views.py
```

Pokazuję oczywiście jedynie najważniejsze z technicznego punktu widzenia pliki. Założeniem frameworka jest wyabstrahowanie każdej funkcjonalności aplikacji do oddzielnych folderów (na schemacie: app1, app2). Każdy z nich powinien zawierać oddzielne modele oraz widoki stricte odnoszące się do danej części budowanego systemu. Pozwala to na łatwe przyłączanie kolejnych, nowych elementów do gotowej aplikacji, a także tworzenie modułów wielokrotnego użytku.

2.2.2 Gotowe paczki

W związku z liczną społecznością zgromadzoną wokół Django bez większych problemów możemy znaleźć gotowe rozwiązania napotkanych problemów czy uzupełnienia frameworka o funkcjonalności, których wprowadzenia do rdzenia nie przewidzieli twórcy. Jednym z największych tego typu rozwiązań jest Django Rest Framework - narzędzie służące do budowy API w stylu architektury REST.

2.2.3 Bierz co chcesz

Przede wszystkim już w samym kodzie frameworka znaleźć możemy moduły, które możemy, choć wcale nie musimy wykorzystać. Wspomniany juz w tym rozdziale moduł autentykacji zawiera podstawowe modele użytkownika naszego systemu, formularze rejestracji i logowania, a nawet wbudowane widoki odzyskiwania hasła.Nie ma żadnego przymusu korzystania z nich, jednak mało która apliakcji webowa nie korzysta z tego typu funkcjonalności. Sprawa ulega komplikacji, gdy wbudowane funkcjonalności nie odpowiadają w pełni naszym potrzebom, jednak z Django nie stanowi to większego problemu - nadpisywanie, dopisywanie i ogólna modyfikacja gotowych elementów jest intuicyjna i szybka.

2.3 ORM

Jednym z elementów, który wyróżnia Django jest jego ORM:

```
from django import models

class AuthorModel(models.Model):
  name = models.CharField('Name', max_length=50)

class BookModel(models.Model):
  title = models.CharField('Title', max_length=100)
  author - models.ForeignKey(Author, on_delete=models.CASCADE)
```

Przedstawiony fragment kodu przedstawia reprezentację schematu tabel zawierających dane

na temat książek i ich autorów. Atrybuty klasy definiują kolumny i ich typy. Taki sposób planowania bazy danych pozwala na niemal obrazowe przedstawianie jej przyszłej budowy, a także zwiększa czytelność zależności wystepujących między poszczególnym encjami naszej aplikacji.Obsługa wielu różnych baz danych jest wspierana dzięki paczkom przygotowanym przez społeczność.

Rozdział 3

Scraper

W niniejszym rozdziale zajmę się omówieniem użytych w projekcie zagadnień dotyczących webscrapingu.

Scrapy jest jedną z najpopularniejszych bibliotek pythonowych, które mają za zadanie zbierać informacje ze stron internetowych. Webscraping oparty jest na pyhtonowym frameworku *Scrapy*, wspieranym przez biblioteki *Splash* oraz *XPath*. Uzyskiwałem wiedzę posługując się dokumentacjami bibliotek *Scrapy*[1], *Splash*[2] oraz *XPath*[3].

Skrypt uruchamiany jest na trzech stronach sklepów internetowych: *https://reserved.com, https://zalando.pl, https://domodi.pl*. Wykonywanie programu jest podzielone na trzy etapy:

1. Po uzyskaniu słów kluczowych program wprowadza w wyszukiwarce strony daną frazę wpisaną przez użytkownika.
2. Przekierowuje do wszystkich wyników wyszukiwania.
3. Zbiera oraz zwraca informację z rubryki wygenerowanej dynamicznie przez stronę, która ma na celu poinformowanie kupującego o rzeczach zakupionych przez innych użytkowników wraz z wyszukiwanym ubraniem.

[1] https://doc.scrapy.org

[2] https://splash.readthedocs.io/en/stable/api.html

[3] https://doc.scrapy.org/en/xpath-tutorial/topics/xpath-tutorial.html

3.1 Opis etapów

3.1.1 Etap 1

Aplikacja włącza program scrapingowy za pomocą następującej komendy: *scrapy crawl clothing -a tag="fraza" -o results.json*. Oznacza to uruchomienie programu o nazwie *clothing* z parametrem do wyszukania (tag), a rezultat będzie przechowywany do pliku *results.json*. Po uruchomieniu, do każdej strony internetowej inicjowana jest metoda *search*, która za pomocą *XPath* odszukuje formularz na stronie głównej sklepu, wpisuje interesującą użytkownika nazwę ubrania i wysyła request z żądaniem wyszukania.

3.1.2 Etap 2

Strona sklepu internetowego zwraca wyniki wyszukiwania. Program zbiera wszystkie linki do wyszukanych ubrań za pomocą *XPath*, który przeszukuje w kodzie HTML odnośniki mające w adresie przekierowującym rodzaj ubrania. Następnie przechodzi do tych stron odpalając kolejną metodę, która zależnie od sklepu szuka w kodzie HTML strony innych struktur.

3.1.3 Etap 3

Będąc na tym etapie program jest już w dokładnie jednej ofercie zakupu ubrania. Zależnie od strony, skrypt zachowuje się w inny sposób:

- dla stron *reserved* oraz *domodi* od razu następuje przeszukiwanie kodu HTML w celu znalezienia proponowanych rzeczy poprzez znalezienie odpowiedniej struktury
  - dla sklepu Reserved każda oferta przeszukiwania jest umieszczona w znaczniku , a znacznik ten zaczyna się od klasy, która ma nazwę *portrait*. Dla Domodi jest to każdy znacznik <li>.
- dla sklepu Zalando wyszukiwany jest najpierw tekst *zobacz więcej*. Po znalezieniu tekstu aplikacja przechodzi na tą podstronę, żeby móc wylistować wszystkie ubrania proponowane przez sklep. W przypadku, gdy tekst *zobacz więcej* nie zostanie znaleziony, aplikacja nie zwróci żadnych propozycji ze strony Zalando.

Program pobiera następujące informacje z propozycji kupna:

- adres URL
- zdjęcie ubrania
- cenę
- nazwę ubrania

Informacje zapisywane są tylko wtedy, jeśli jest ich cały komplet – powoduje to uniknięcie przedostawania się do wyników niepotrzebnych danych bądź innych przypadkowo pobranych rzeczy. Zapisywanie odbywa się w formacie json o strukturze słownikowej:

{ClothesName: { "image": imageURL, "price": PRICE, "url": URL}}

ClothesName – przechowywany w formacie string; jako wartość zawiera słownik ze szczegółami

imageURL – przechowywany w formacie string; zawiera adres do zdjęcia ubrania

PRICE – przechowywany w formacie string; zależnie od strony może być z końcówką „zł” lub „PLN”

URL – przechowywany w formacie string; zawiera adres do strony sklepu z daną rzeczą

Program na bieżąco przesyła taki słownik do pliku wynikowego, z którego dalsza część programu może na bieżąco czytać.

Program, żeby mógł działać na dynamicznych stronach używa biblioteki Splash. Splash używa funkcji napisanej w języku Lua:

```
function main(splash, args)
        assert(splash:go(args.url))
        assert(splash:wait(0.5))
        assert(splash:set_viewport_full())
        return {
          html = splash:html()}
          end
```

Splash:go(args.url) przechodzi na stronę podaną w argumencie.

Splash:wait(0.5) – określa jak długo program czeka na załadowanie strony

Splash:set_viewport_full() sprawia, że wczytywana jest cała strona HTML

Wysyłanie requestów do stron również odbywa się za pomocą metody z biblioteki Splash – *SplashRequest*, która jako argumenty przyjmuje adres strony, metodę do której ma przejść w następnym kroku oraz argumenty (przekazanie całej funkcji z języka Lua jako parametr).

3.2 Dlaczego Scrapy

Scrapy jest stosunkowo dużym frameworkiem skupionym na webscrapingu. Nie jest trudno opanować podstawy – framework ten jest bardzo dobrze udokumentowany oraz posiada dużo poradników dotyczących pierwszych kroków. Wszystkie te udogonienia pozwalają poznać podstawowe fundamenty Scrapy, które w dużej mierze powinny wystarczyć do większości zadań związanych z webscrapingiem. Posiada własne selektory, dzięki którym można swobodnie poruszać się po stukturze HTML. Jego największą zaletą jest to, że służy jako crawler – wystarczy podać adres początkowy, a framework będzie w stanie przeszukiwać wgłąb stronę wedle chęci osoby piszącej program.

Przy wielkości opisywanego przez nas projektu oraz przy daleko idących planach rozbudowy aplikacji pisanie crawlera we frameworku w pełni skupionym na scrapingu jest niezbędne do uniknięcia dalszych problemów związanych z obsługą coraz bardziej skomplikowanych stron internetowych.

3.2.1 Selektory

Scrapy implementuje własne selektory pozwalające przeszukiwanie kodu HTML. Można wydobywać części kodu za pomocą języka XPath oraz poprzez wybieranie kodu CSS, który stylizuje HTML.

W opisywanej aplikacji zdecydowałem się na posługiwanie się językiem XPath – był dla mnie wygodniejszy i łatwiejszy w nauce. Przykładowy selektor z napisanego przeze mnie scrapera:

```
response.xpath("//a[./img[contains(@src,.)]]/@href").extract()
```

Powyższa linijka oznacza wybranie wszystkich znaczników <a>. Następnym krokiem jest znalezienie takich znaczników <img>, które posiadają w sobie odnośniki do innych stron. *Extract()* powoduje, że wszystkie wyniki wyszukiwania pojedynczo będą formatowane jako typ string, który będzie można przekształcić wedle uznania. Ważnym elementem selektorów jest możliwość wyszukania danej informacji znajdującej się wewnątrz wybranych znaczników. Za przykład posłuży mi fragment kodu. Ma on za zadanie zebrać wszystkie niezbędne informacje na temat strony *zalando.pl*:

```
    for item in
        response.xpath("//div[./a[contains(@href,'zalando.pl')]]"):
        url = item.xpath('./a/@href').extract()
        itemImg =
            item.xpath('.//*/img[contains(@src,.)]/@src').extract()
        price =
            item.xpath(".//span[contains(text(),'zl')]/text()").extract_first(
        itemName =
            item.xpath('.//*/img[contains(@src,.)]/@alt').extract()
```

Xpath zaczynający się od kropki oznacza, że program zaczyna przeszukiwać od – podanego w przykładzie wyżej, <a>. Wybierane są tylko hiperłącza posiadające odnośnik do strony zawierającej podstring źalando.pl".

Finalnie, jeśli zdajemy sobie sprawę które rzeczy Scrapy powinien szukać oraz mamy otwartą stronę z dokumentacją[4], to napisanie owego crawlera od podstaw, nie znając wcześniej składni, okazuje się stosunkowo proste i przyjemne.

[4]https://doc.scrapy.org