

Studium licencjackie

Kierunek: Informatyka

Specjalność: Bazy danych i technologie internetowe

Michał Polakowski

Nr albumu: 203446

Karol Chmielewski

Nr albumu: 246668

Michał Pydyszewski

Nr albumu: 246631

Bartłomiej Pysiak

Nr albumu: 251191

Praktyczne zastosowanie web scrapingu do tworzenia użytkowych aplikacji internetowych

Praca licencjacka

napisana w Instytucie Matematyki,

Fizyki i Informatyki

pod kierunkiem naukowym

dra Włodzimierza Bzyla

Gdańsk, 2019

Spis treści

1	Wprowadzenie	5
2	Django	8
2.1	Popularność	8
2.2	Modularność	9
2.2.1	Jedna funkcjonalność, jedna aplikacja	9
2.2.2	Gotowe paczki	10
2.2.3	Bierz co chcesz	10
2.3	ORM	10
3	Moduł Bramy	12
3.1	Load balancing	12
3.2	Caching	13
3.3	Anonimizacja	14
3.4	Zwiększenie wydajności	14
3.5	Skalowalność	15
4	Scraper	16
4.1	Opis etapów	17
4.1.1	Etap 1	17
4.1.2	Etap 2	17
4.1.3	Etap 3	17
4.2	Dlaczego Scrapy	19
4.2.1	Selektory	19
5	Wizualna prezentacja projektu	21

<i>SPIS TREŚCI</i>	3
6 Etyczne i prawne aspekty web-scrapingu	24
6.1 Wprowadzenie	24
6.2 Scraping w życiu codziennym, regulacje prawne	24
6.3 Podsumowanie	25
7 Diagramy	26
7.1 Wprowadzenie	26
7.2 Diagram sekwencji	27
7.3 Diagram klas	28

Streszczenie

Nasza praca opisuje aplikację webową służącą jako pomoc w dopasowaniu za pomocą web-scrapingu elementu garderoby do najlepszych propozycji dostępnych w sklepach internetowych. Web-scraping oparty jest na frameworku Scrapy, aplikacja zbudowana jest przy użyciu frameworka webowego Django, anonimizacja użytkownika i optymalizacja wydajnościowa jest możliwa dzięki serwerowi NGINX. Wszystkie zadania są obsługiwane przy użyciu kolejek dzięki Celery, a zbierane informacje są zapisywane w MongoDB oraz PostgreSQL. W pracy opisujemy działanie każdego z modułów, oraz przedstawiamy moralne aspekty web-scrapingu.

Rozdział 1

Wprowadzenie

Pomysł na naszą aplikację narodził się podczas odbywanych przez nas licznych wizyt w galeriach handlowych, gdzie zauważyliśmy jak dużo czasu, wysiłku i pieniędzy ludzie poświęcają aby do posiadanej odzieży dopasować nowe elementy garderoby.

Na taki stan rzeczy składa się kilka czynników. Pierwszym z nich jest dostępność produktów. Wybór konsumenta ograniczony jest przez wyposażenie sklepów na terenie galerii. Ponadto, konsument ma tendencję do kupienia produktu, który w jego mniemaniu jest najbliższy temu, czego oczekiwał, lecz często nie dokładnie taki. To pokazuje nam kolejny problem - konsument nie pożytkuje swoich pieniędzy najlepiej, jak to możliwe. Nie sposób jest porównać wszystkich przedmiotów z puli potencjalnych dopasowań, ponieważ są one rozproszone po różnych sklepach, często galeriach, a nawet miastach.

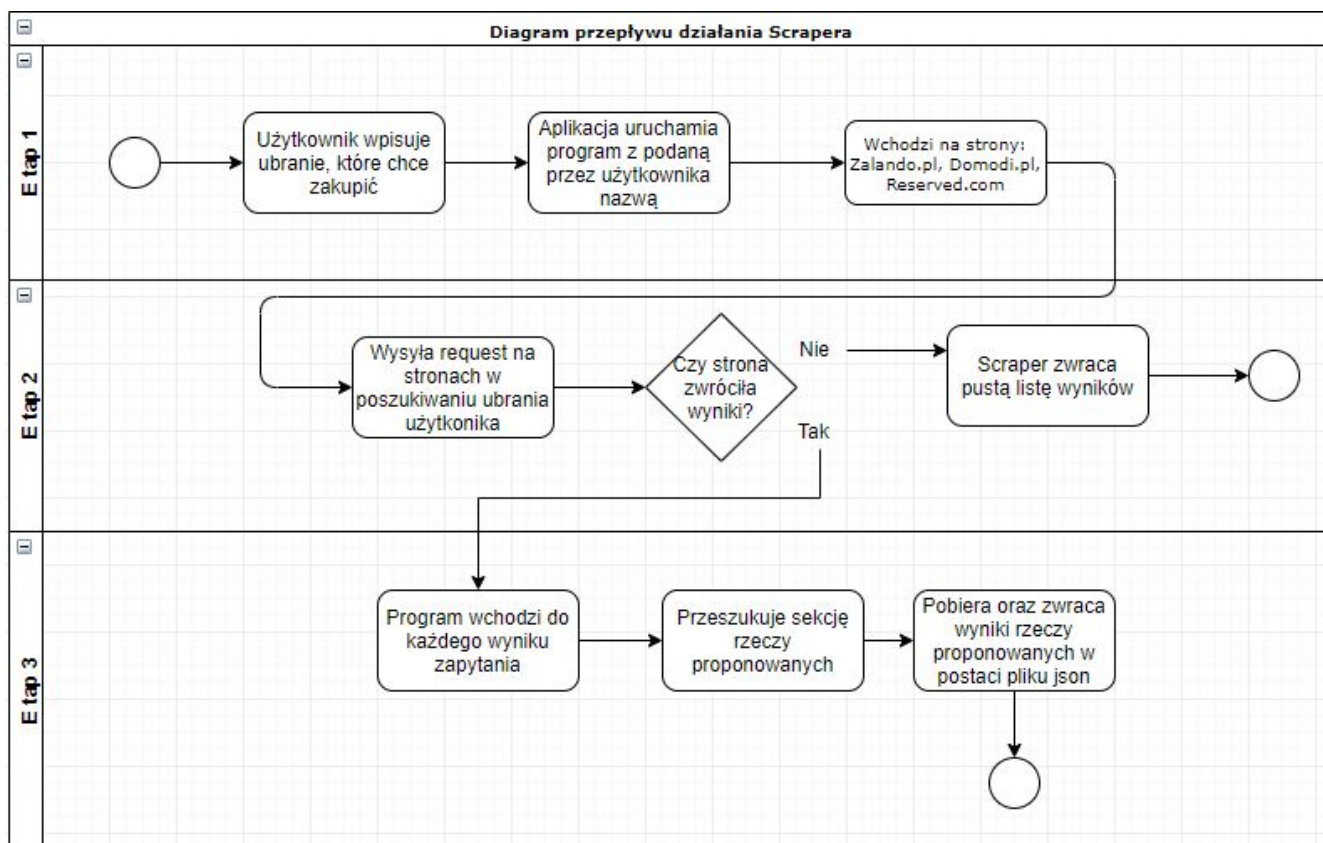
Trzecim czynnikiem jest czas poświęcony na wybieraniu potencjalnych dopasowań, ponieważ koniecznym jest fizycznie udać się do sklepów.

Jednym z rozwiązań problemu braku czasu jest zatrudnienie stylisty, który podpowie, które dopasowanie jest najlepsze, lecz jest to kosztowna usługa, na którą nie każdy może sobie pozwolić.

Odpowiedzią na te problemy jest nasza aplikacja. Po wprowadzeniu w pasku wyszukiwarki produktu, prezentowana jest użytkownikowi lista najlepiej skomponowanych do niego artykułów wraz z linkiem do sklepu, zdjęciem oraz ceną. Wystąpienie na liście propozycji dopasowania podyktowane jest jego popularnością oraz częstotliwością doboru wśród innych użytkowników systemu. Pozwala to na oszczędzenie czasu poprzez możliwość porównania bez wycho-

dzenia z domu. Aplikacja przyczynia się również do oszczędności pieniędzy, ponieważ niejako eliminuje potrzebę zatrudniania stylisty.

Produkt działa z wykorzystaniem web-scrapingu opartego na pythonowym frameworku Scrapy. Poniższy diagram przedstawia po krótko proces działania wspomnianego modułu.



Rysunek 1.1: Diagram działania scraper'a. Wykonanie własne

Wyszukiwania są cachowane na serwerze pośredniczącym, co pozwala użytkownikowi na szybsze wyszukiwanie i płynniejsze korzystanie z aplikacji. Zasoby statyczne takie jak zdjęcia pobierane są z serwera pośredniczącego, a nie z domeny docelowej.

Dodatkowym atutem korzystania z serwera pośredniczącego jest ukrycie prawdziwego adresu logicznego użytkownika, dzięki czemu utrudni to reklamodawcom prezentowanie mu reklam, co miałyby miejsce w przypadku bezpośrednich wizyt w poszczególnych domenach sklepów internetowych.sklepach

W projekcie wykorzystujemy load balancing, dzięki któremu ruch będzie rozpraszany na wszystkie węzły, a użytkownik nie odczuje spadku wydajności w przypadku wzmożonego ruchu.

Słowa kluczowe: web-scraping, dopasowanie ubrań, stylizacja modowa

Rozdział 2

Django

Python jest aktualnie jednym z najpopularniejszych 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 najczęściej 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 funkcjonalność, jedna aplikacja

Przy tworzeniu podstawowego schematu djangoowej 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

Przed wszystkim już w samym kodzie frameworka znaleźć możemy moduły, które możemy, choć wcale nie musimy wykorzystać. Wspomniany już 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 aplikacja 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 występujących między poszczególnymi encjami naszej aplikacji. Obsługa wielu różnych baz danych jest wspierana dzięki paczkom przygotowanym przez społeczność.

Rozdział 3

Moduł Bramy

Moduł bramy ma za zadanie optymalizację i dystrybucję ruchu sieciowego. Używa NGINX jako serwera pośredniczącego aby zapewnić możliwie najkrótszy czas odpowiedzi dla użytkownika aplikacji.

W związku z tym możemy wyszczególnić 5 główne zadania, jakie serwer bramy wykonuje:

- Load balancing
- Cachowanie
- Anonimizacja użytkownika
- Zwiększenie wydajności
- Skalowalność

3.1 Load balancing

Load balancing to technika redukowania obciążenia wynikającego z ilości zapytań jakie serwer musi obsłużyć poprzez dystrybuowanie ruchu na więcej niż jedną instancję, która jest w stanie takie zapytania obsłużyć. Nasz system musi zapewniać użytkownikowi poczucie lekkości niezależnie od ilości aktualnie nawiązanych połączeń. Aby rzeczywiście znalezienie najlepszego dla wybranej rzeczy dopasowania wiązało się z oszczędnością czasu, z perspektywy infrastruktury musimy zapewnić możliwie najkrótszy czas odpowiedzi. Na serwerze bramy do

dystribucji ruchu wybrany został algorytm karuzelowy - wszystkie procesy mają ten sam priorytet.

```
http {  
    upstream app1 {  
        server srv1.example.com  
        server srv1.example.com  
        server srv1.example.com  
    }  
}
```

3.2 Caching

Funkcjonalność aplikacji oraz korzyści jakie daje ona użytkownikowi opierają się między innymi na możliwości szybkiego porównania i wyciągnięcia wniosków dotyczących potencjalnego dopasowania. Możliwe jest to dzięki wyświetleniu zdjęcia znalezionej produktu. Niezadko może się zdarzyć, że propozycje dopasowań będą się powtarzać. Daje to możliwość cachowania plików statycznych, dzięki czemu kolejne zapytanie HTTP po taki zasób będzie serwowane bezpośrednio z dysku twardego serwera bramy, a nie z serwera docelowego. Co więcej, NGINX daje udostępnić interfejs do ustawienia cachowania po stronie klienta, dzięki czemu zasoby statyczne w razie zapytania HTTP będą pobierane bezpośrednio z pamięci podręcznej przeglądarki.

```
location ~* \.(jpg|jpeg|png|gif|ico)$ {  
    expires 30d;  
}  
  
location ~* \.(css|js)$ {  
    expires 7d;  
}
```

3.3 Anonimizacja

Niezbędnym jest zapewnienie bezpieczeństwa użytkownika i systemu. Bezpieczeństwo użytkownika jest rozumiane przez nas jako anonimizacja jego wyszukiwań. W przypadku korzystania z NGINX jako serwera pośredniczącego anonimizacja wynika naturalnie z architektury systemu. Rządania HTTP po zasoby udostępniane przez sklepy internetowe są wykonywane z adresem logicznym bramy, a nie użytkownika końcowego. Dzięki temu sklepy pozbawione są możliwości targetowania reklam dla użytkownika biorąc pod uwagę jego poprzednie wizyty. Dzięki temu użytkownikowi aplikacji nie będą prezentowane reklamy powiązane z wyszukiwanym produktem - nie zapewniają one w żaden sposób, że reklamy będą dotyczyć rzeczy pasujących, a jedynie podobnych, lub nawet tych samych.

3.4 Zwiększenie wydajności

Pośród ogromu możliwości, jakie serwer NGINX daje w kontekście optymalizacji ruchu, wybraliśmy kilka, które w najbardziej bezpośredni sposób wpływają na wydajność naszej infrastruktury:

1. Ustawienie liczby procesów potomnych serwera na podstawie liczby CPU oraz dysków twardych

```
worker_processes auto;
```

2. Zwiększenie liczby operacji weścia/wyjścia na dysku twardym przez wyłączenie logowania

```
error_log /var/log/nginx/error.log crit;  
access_log off;
```

3. Wymuszenie na wysłaniu danych z bufora tak szybko jak to możliwe, pomijając algorytm Nagle'a, według którego pakiet, który jest za mały, czeka według standardowej implementacji UNIX 200 milisekund na wypełnienie.

```
tcp_nodelay on;
```

Wymuszenie na pakiecie blokady zanim osiągnie maksymalny rozmiar

```
tcp_nopush on;
```

3.5 Skalowalność

Wraz ze wzrostem zainteresowania systemem, musimy przewidzieć, że w pewnym momencie obecna architektura może nie być wystarczająca, aby zapewnić stabilność i wysoką wydajność. Serwer NGINX jest zorientowany wokół jednego procesu głównego i wielu procesów potomnych zależnie od liczby rdzeni CPU i dysków twardych. W miarę potrzeby możemy zwiększyć liczbę rdzeni procesora, a przez to zwiększyć też liczbę procesów potomnych. Infrastruktura jest zorientowana zarówno wokół konieczności skalowalności wertykalnej, ale również horyzontalnej. Zwiększenie liczby procesów na które dystrybuowany jest ruch w procesie load-balancingu również jest możliwe i zapewni stabilność w przypadku potrzeby skalowania.

Rozdział 4

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 pythonowym 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>

4.1 Opis etapów

4.1.1 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.

4.1.2 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.

4.1.3 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 `<div>`, a znacznik ten zaczyna się od klasy, która ma nazwę *portrait*. Dla Domodi jest to każdy znacznik ``.
- 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 przedostawiania 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 Lua⁴:

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

⁴<https://splash.readthedocs.io/en/stable/scripting-tutorial.html>

`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).

4.2 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 strukturze HTML. Jego największą zaletą jest to, że służy jako crawler – wystarczy podać adres początkowy, a framework będzie w stanie przeszukiwać wgląd 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.

4.2.1 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 scra-

pera, który jest używany w naszym programie:

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

Powyższa linijka oznacza wybranie wszystkich znaczników <a>. Następnym krokiem jest znalezienie takich znaczników , 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(),'zł')]/text()").extract_first()
    itemName =
        item.xpath('./*/img[contains(@src,.)]/@alt').extract()
```

Kod ten napisałem sam, za pomocą dokumentacji Xpath oraz Scrapy. 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 *zalando.pl*".

Finalnie, jeśli zdajemy sobie sprawę które rzeczy Scrapy powinien szukać oraz mamy otwartą stronę z dokumentacją⁵, to napisanie owego crawlera od podstaw, nie znając wcześniej składni, okazuje się stosunkowo proste i przyjemne.

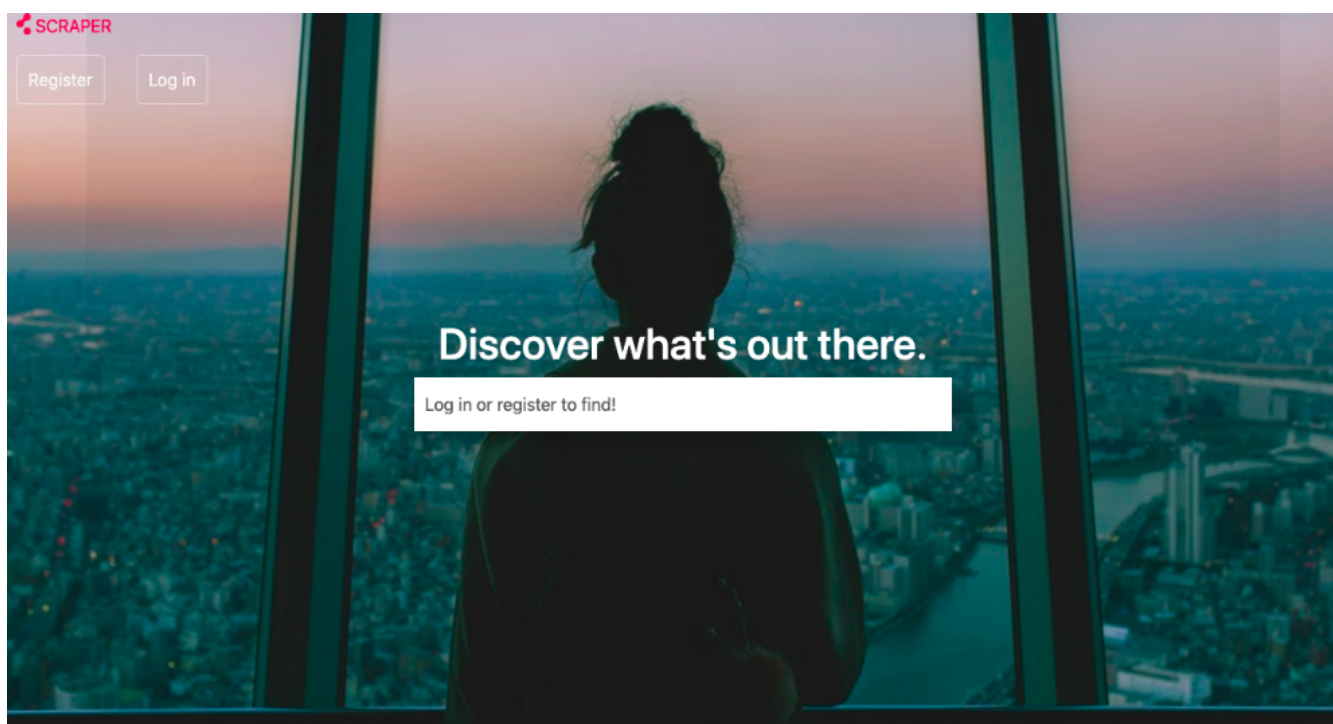
⁵<https://doc.scrapy.org>

Rozdział 5

Wizualna prezentacja projektu

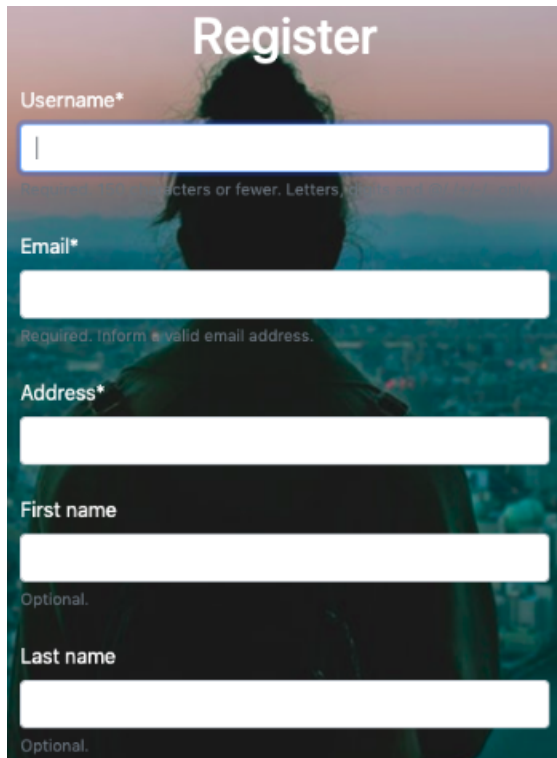
Niniejszy rozdział poświęcony jest przedstawieniu projektu ze strony wizualnej.

Wchodząc na naszą aplikację pierwszym obrazkiem, który nas powita jest strona tytułowa wraz z zablokowanym oknem formularza, w którym będziemy mogli wpisać nazwę ubrania, do którego program będzie szukać dopasowań.



Rysunek 5.1: Strona tytułowa

Użytkownik, żeby móc korzystać z aplikacji musi się zarejestrować. W tym celu klika przycisk *Register*, a następnie wypełnia formularz.



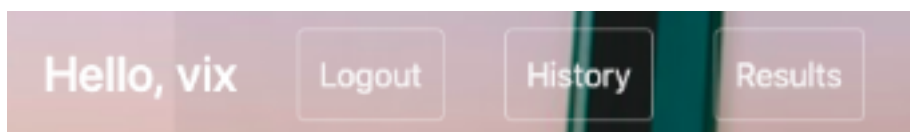
The image shows a 'Register' form with a background of a person's silhouette. The form contains the following elements:

- Register** (Title)
- Username*** (Label) with a text input field. Below it, a small error message: "Minimum 6 characters or fewer. Letters, numbers and underscores only."
- Email*** (Label) with a text input field. Below it, a small error message: "Required. Inform a valid email address."
- Address*** (Label) with a text input field.
- First name** (Label) with a text input field.
- Last name** (Label) with a text input field. Below it, a small error message: "Optional."

Rysunek 5.2: Formularz rejestracji

Następnym krokiem po rejestracji jest logowanie. Używając danych wpisanych podczas rejestracji można zalogować się do systemu.

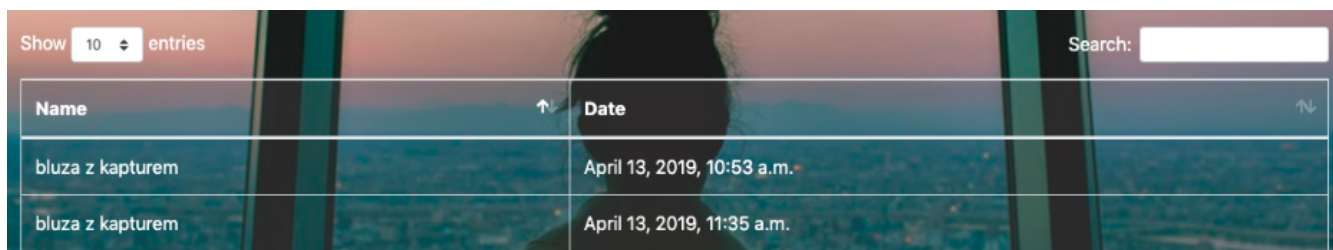
Od tej chwili użytkownik może korzystać z aplikacji w stu procentach. Pojawiło się okna *History* oraz *Results*, w których odpowiednio można przeglądać swoją historię wyszukiwań oraz wyniki ostatniego wyszukania.



Rysunek 5.3: Wszystkie okna użytkownika

Jeśli przejdziemy do podstrony z historią wyszukiwań, otrzymamy widok z tabelą, która ma dwie kolumny: *Name* i *Date*.

Tabela jest sortowalna, jest możliwość wyszukania na podstawie frazy wpisanej przez użytkownika.



Show 10 entries	Search:
Name ↑	Date ↑↓
bluza z kapturem	April 13, 2019, 10:53 a.m.
bluza z kapturem	April 13, 2019, 11:35 a.m.

Rysunek 5.4: Historia wyszukiwań

Na stronie głównej zostało odblokowane okno z wyszukiwaniem. W tym celu można wpisać interesującą użytkownika rzecz, a następnie przejść do zakładki results, w której pojawi się tabela z nazwą ubrania, ceną, linkiem do sklepu oraz linkiem do zdjęcia. Tabela jest w pełni sortowalna, można wyszukiwać danej frazie.

Show

10

entries

Search:

Name	Price	Link	Photo
adidas Buty Tubular Shadow	740	Go to shop!	Preview
Nike Tiempo - Sneakers Course	290	Go to shop!	Preview

Rysunek 5.5: Wyniki wyszukiwań

Rozdział 6

Etyczne i prawne aspekty web-scrapingu

6.1 Wprowadzenie

Web scraping ze względu na swoje działanie jak i późniejsze wykorzystywanie pozyskanych informacji traktowany jest zwykle jako procedura wątpliwa nie tylko moralnie, ale często również jako działanie nie do końca zgodne z prawem. Mamy tu przecież do czynienia z pozyskiwaniem, zwykle bez wiedzy właściciela strony, portalu czy serwisu, informacji, które mogą być objęte prawami autorskimi, naruszającymi czyjąś prywatność lub wręcz tajnymi (ale nie dostatecznie zabezpieczonymi).

6.2 Scraping w życiu codziennym, regulacje prawne

Podstawy prawne, które mogą regulować zasady działania web-scrapingu różnią się w zależności od naszego położenia na kuli ziemskiej. Ogólnie rzecz biorąc, takie wytyczne mogą być ujęte w zasadach korzystania z danej strony internetowej. Przestrzeganie tychże zasad i dochodzenie swoich praw ze strony „poszkodowanego” portalu nie zawsze jest jasne i łatwe do przeprowadzenia. Biorąc pod uwagę powyższe, niektóre kraje zdecydowały się wprowadzić regulacje, które wprost decydowałyby o tym, co jest, a co nie jest legalne. Jednakże, ze względu na różnorodność jaka charakteryzuje działania związane ze scrapingiem ostateczna decyzja należy zwykle do sądu. Mowa tu oczywiście o działaniach, których nie można wprost określić jako nielegalnych np. kradzieży danych i wykorzystywania ich w negatywnych celach. Spory kończące

się na ławach sądowych dotyczą zwykle wielkich korporacji działających przede wszystkim w sieci takich jak Facebook, Google, LinkedIn, ale również firm działających w różnych gałęziach gospodarki, ale w obecnych czasach nierozzerwalnie związanych już z Internetem.

Ponieważ scraping wykorzystywany jest przez strony dzięki którym można na przykład porównywać ceny biletów lotniczych, sprawdzać dostępność hoteli, ceny pokoi czy chociażby porównywać ceny dostępnych produktów w bezmiarze sklepów internetowych istniejących obecnie. Z punktu widzenia konsumenta działania takie są jak najbardziej pozytywne, jednak z punktu widzenia sprzedawcy już nie tak bardzo. Te same dane, zamiast służyć dobru konsumenta, wykorzystane mogą być również przez inne firmy do nie do końca czystych zagrań, jak np. celowe obniżanie cen.

Inny negatywny aspekt zauważyć można chociażby w przypadku odpłatnych imprez masowych. Dotyczy to zjawiska „automatycznego podkupowania” biletów przez inne podmioty oczywiście w celach uzyskania dodatkowych korzyści z późniejszej ich sprzedaży po swoich cenach. Aby temu przeciwdziałać wprowadzono ograniczenia w ilości kupowanych biletów przez pojedynczego użytkownika. Nie jest to rozwiązanie idealne, ale też nie uderza zbyt mocno w konsumenta zainteresowanego kupieniem biletu w najlepszej możliwej cenie.

6.3 Podsumowanie

Scraping w różnych formach istniał praktycznie od momentu rozpowszechnienia się internetu i zapewne istniał będzie aż po jego kres (albo i dłużej). Biorąc pod uwagę zalety, które przynoszą takie działania ciężko stwierdzić czy zaistnieje kiedykolwiek wykładnia, która raz na zawsze zabroni jednym podmiotom korzystać z danych zamieszczanych w sieci przez innych.

Jak pokazuje historia, każdy przypadek wchodzenia na drogę sądową przez firmy „dotknięte” działaniem scrapingu jest inny i bez ścisłych uregulowań prawnych nie do przewidzenia jest decyzja jaka zostanie podjęta przez sąd rozpatrujący daną sprawę.

Rozdział 7

Diagramy

7.1 Wprowadzenie

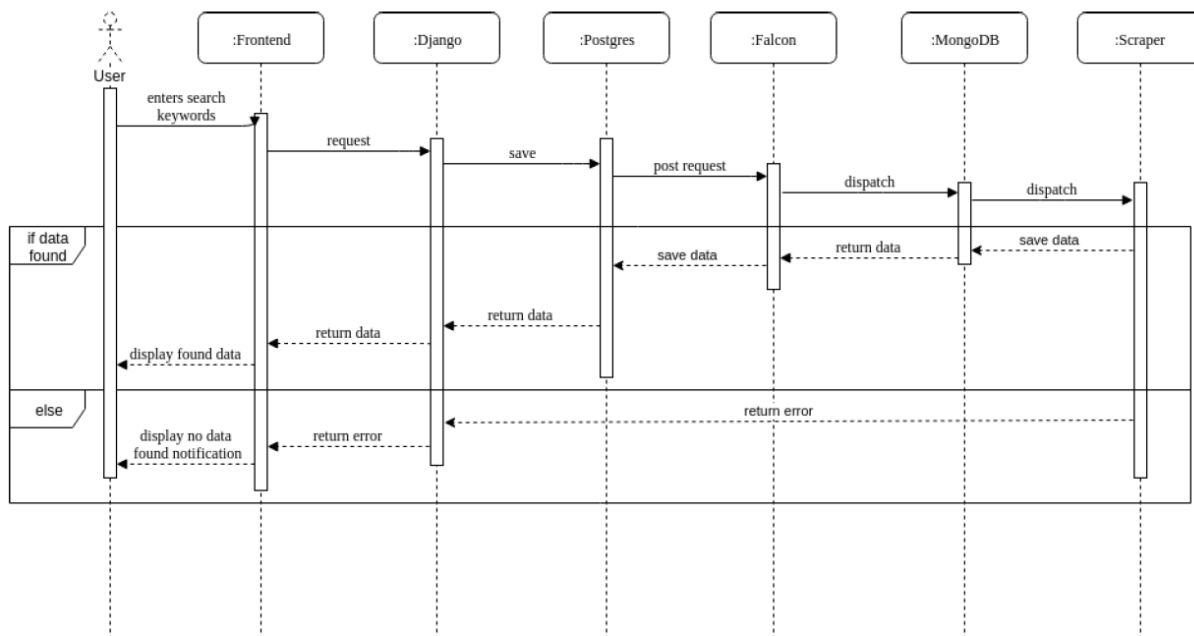
W poniższym rozdziale przedstawione zostały diagramy sekwencji i klas opisujące tworzoną przez nas aplikację.

Diagramy sekwencji przedstawiają zależności czasowe pomiędzy obiektami oraz służą do modelowania systemów czasu rzeczywistego i złożonych scenariuszy¹. Natomiast diagramy klas pokazują określony fragment struktury systemu. Diagramów klas używa się do modelowania statycznych aspektów perspektywy projektowej. Wiąże się z tym silnie modelowanie słownictwa systemu, kooperacji lub schematów. Diagramy klas pozwalają na sformalizowanie specyfikacji danych i metod. Mogą także pełnić rolę graficznego środka pokazującego szczegóły implementacji klas².

¹https://pl.wikipedia.org/wiki/Diagram_interakcji

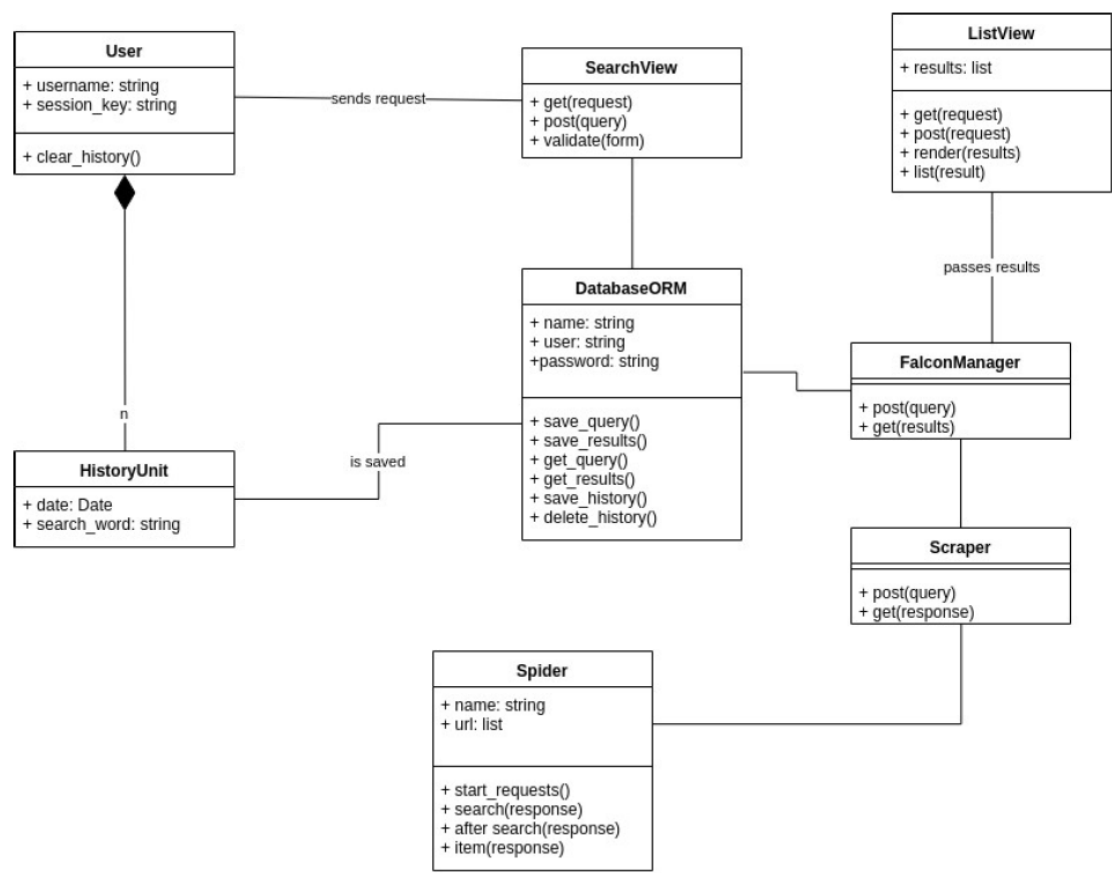
²https://pl.wikipedia.org/wiki/Diagram_klas

7.2 Diagram sekwencji



Rysunek 7.1: Diagram sekwencji. Opracowanie własne

7.3 Diagram klas



Rysunek 7.2: Diagram klas. Opracowanie własne

Bibliografia