

# ”Automatyczna analiza rynku pracy.”

Bartłomiej Sielicki

Łukasz Skarżyński

Praca inżynierska  
wersja 0.1

Promotor:  
Barbara Rychalska



Wydział Matematyki i Nauk Informatycznych  
Politechniki Warszawskiej

23.11.2017

# Rozdział 1

## Specyfikacja projektu

### 1.1 Słownik pojęć

- **back-end** - Odpowiada za operacje w tle, których przebiegu użytkownik nie widzi bezpośrednio. Zajmuje się przetwarzaniem, wykonywaniem zadań na podstawie otrzymanych danych. W projekcie słowem back-end określamy wszystkie moduły za wyjątkiem aplikacji internetowej z której korzysta użytkownik.
- **front-end** - Warstwa wizualna systemu - interfejs użytkownika. Głównym jego zadaniem jest pobieranie danych od użytkownika oraz przekazywanie ich do back-endu oraz ewentualne pokazanie odebranej odpowiedzi. W systemie którego dotyczy dana dokumentacja front-end jest stroną internetową.
- **web scraper** - program, którego głównym zadaniem jest zbierać określone dane ze stron internetowych. Gdy w dokumentacji używamy słowa “scraper” zawsze odnosi się ono właśnie do “web scraper’a”.
- **pipeline** - łańcuch przetwarzania danych (funkcji), w którym wejściem kolejnego etapu jest wyjście poprzedniego.
- **framework** - szkielet do budowy różnego rodzaju aplikacji. Definiuje on strukturę aplikacji oraz ogólny mechanizm jej działania, oraz dostarcza zestaw komponentów i bibliotek ogólnego przeznaczenia do wykonywania określonych zadań.
- **API** - (ang. Application Programming Interface) interfejs programowania aplikacji, jest to ściśle określony zestaw reguł i ich opisów, w jaki programy komputerowe komunikują się między sobą. Najczęściej używamy tego słowa w odniesieniu do WEB API - interfejsu komunikacji korzystającego z HTTP oraz formatu JSON do przesyłania danych.

## 1.2 Opis biznesowy

Głównym zadaniem aplikacji ma być automatyczna analiza rynku pracy. System zajmuje się agregacją oraz przetwarzaniem (analizą) ofert zebranych z serwisu zewnętrznego (Pracuj.pl) i przedstawianiem wyników użytkownikowi.

W szczególności proces działania systemu sprowadza się do:

1. Pobierania ofert pracy z serwisu zewnętrznego wyłuskując kluczowe elementy: tytuł ogłoszenia, opis, datę dodania, itp.
2. Zapisania tak zebranych ogłoszeń do własnej bazy danych
3. Przeprowadzenia analizy na treści ogłoszenia uzyskując zeń listę *kluczy*, tj:
  - obszaru branży IT którego dotyczy ogłoszenie
  - stanowiska którego dotyczy
  - wymaganych od pracownika opanowanych technologii / umiejętności
4. Udostępnienia użytkownikowi interfejsu do wyszukiwania zebranych w systemie ofert oraz wyświetlania statystyk przy użyciu wyżej wspomnianych kluczy.

Wymienione działania realizują odrębne komponenty systemu. Bardziej szczegółowy podział i opis znajduje się w sekcji wymagań funkcjonalnych oraz rozdziale drugim będącym dokumentacją każdego z modułów.

Użytkownik bezpośrednio będzie korzystał tylko z ostatniego modułu - aplikacji WWW będącej interfejsem dla zebranych i przetworzonych przez system danych.

Aplikacja będzie niosła korzyść dużej grupie odbiorców takich jak:

- studenci - dzięki aplikacji będą mogli znaleźć świetnie dopasowane stanowisko do ich umiejętności lub obserwując dane stanowisko określić jakie umiejętności są na nim elementarne, a także jakiej technologii powinni się nauczyć w najbliższym czasie.
- osoby szukające pracy - łatwo na podstawie swoich umiejętności znajdą stanowiska dla siebie.
- pracodawcy - na podstawie trendów wśród używanych technologii będą mogli podjąć decyzje dotyczące przyszłych projektów.
- pozostali użytkownicy zainteresowani analizą rynku pracy.

## 1.3 Wymagania funkcjonalne

Podstawą interakcji użytkownika jest strona WWW - użytkownik powinien być w stanie otworzyć ją na dowolnym komputerze z dostępem do internetu i wyposażonym w aktualną wersję jednej z wiodących na rynku przeglądarek.

- Google Chrome w wersji 49 lub wyższej
- Mozilla Firefox w wersji 52 lub wyższej
- Safari w wersji 10.1 lub wyższej

Posiadanie przeglądarki innej niż wymienione, lub w starszej wersji nie oznacza że strona nie będzie działać, jednak jako twórcy nie możemy zagwarantować że będzie to działanie w pełni poprawne.

Na stronie nie przewidujemy kont użytkowników, nawet administracyjnego. Każdy wchodzący na stronę będzie miał dostęp do tych samych danych oraz takie same możliwości.

Funkcjonalność strony rozbita jest na dwa moduły. Moduł wyszukiwarki oraz moduł statystyk. Możliwości użytkownika prezentuje załączony na końcu sekcji diagram przypadków użycia.

### 1.3.1 WYSZUKIWANIE OFERT WG KLUCZA

Jednym z podstawowych zastosowań strony jest wyszukiwanie ofert zebranych i umieszczonych w bazie przez nasz system. Jako kryterium wyszukiwania (przez mechanizm filtrowania) może zostać użyty tzw. *klucz*, czyli wyłuskana z opisu ogłoszenia jego cecha. Klucze dzielimy na trzy kategorie:

- **Obszary** (np. Mobile development, Helpdesk)
- **Stanowiska** (np. Software Developer, Data Scientist)
- **Technologie i umiejętności** (np. Java, Docker, AWS)

Pozwoli to na kompleksowe wyszukiwanie ofert ze względu na branżę czy pozycję którą interesuje się użytkownik oraz posiadane przez niego umiejętności. Możliwe jest podanie wielu kluczy jako kryterium.

### 1.3.2 WYŚWIETLENIE OFERTY

Wynikiem wyszukiwania jest lista ofert. Każdą z nich użytkownik może wyświetlić uzyskując dostęp do takich informacji jak:

- Tytuł oferty
- Data dodania i wygaśnięcia oferty w macierzystym serwisie
- Nazwa pracodawcy i miejsce pracy
- Wszystkie wyłuskane przez system klucze
- Odnośnik do oryginalnego ogłoszenia w macierzystym serwisie

### 1.3.3 WYŚWIETLENIE STATYSTYK SERWISU

W osobnej sekcji użytkownik ma dostęp do wyświetlenia zbiorczych statystyk dotyczących serwisu i dostępnych w nim danych. Zbiór dostępnych statystyk planowo zostanie rozwinięty podczas pracy nad ostatnimi dwoma modułami systemu. Bazowe, przewidziane już teraz to:

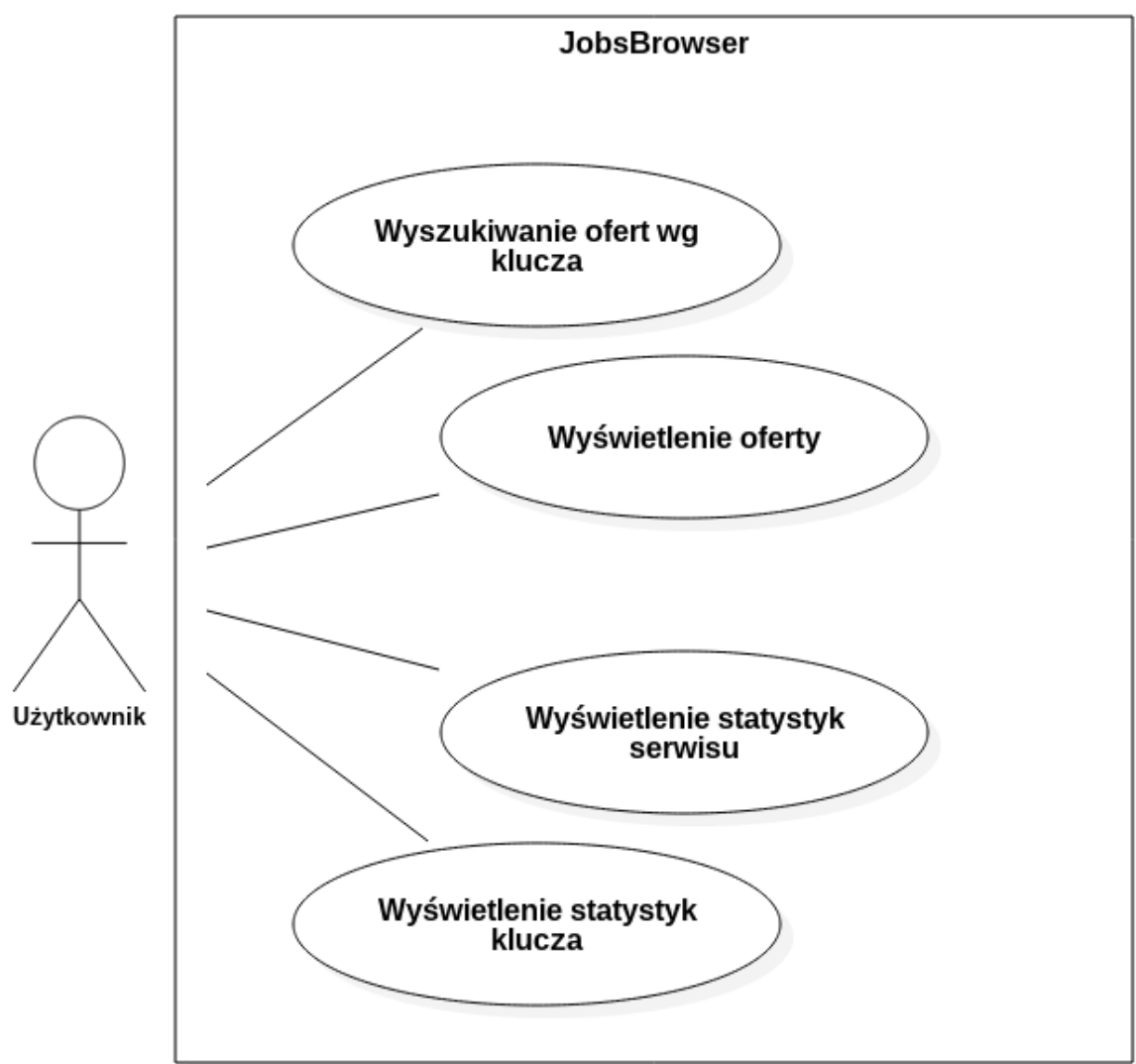
- Liczba wszystkich ofert w bazie systemu
- Wykres powyższej wartości względem czasu
- Datę ostatniego zebrania danych z serwisu macierzystego
- Listę wszystkich istniejących w systemie kluczy

### 1.3.4 WYŚWIETLENIE STATYSTYK KLUCZA

Jest to druga obok wyszukiwania podstawowa funkcjonalność strony. Pozwala ona użytkownikowi na wybór klucza oraz wyświetlenie:

- Wykresu ilości ofert zawierających ten klucz względem czasu
- Wykresu procentowej ilości ofert z systemu zawierających ten klucz
- Pogrupowanych w kategorie kluczy które występują z tym kluczem najczęściej w jednym ogłoszeniu

Możliwe jest podanie wielu kluczy. Wtedy pod uwagę przy generowaniu powyższych statystyk będą brane tylko ogłoszenia które zawierają każdy z nich.



Rysunek 1.1: Przypadki użycia.

## 1.4 Wymaganie niefunkcjonalne

W poniższej tabeli przedstawione zostały wymagania niefunkcjonalne tworzonej aplikacji.

Tabela 1.1: Wymagania niefunkcjonalne

Obszar wymagań	Opis
Używalność (Usability)	Wymagany dostęp do internetu w celu skorzystania z aplikacji. Aplikacja WWW jest intuicyjna w obsłudze dla użytkownika. Aplikacja WWW powinna być dostępna dla użytkownika w każdym wybranym dla niego momencie.
Niezawodność (Reliability)	Dostęp do aplikacji WWW powinien być możliwy przez 24 godziny, 7 dni w tygodniu. Za wyjątkiem prac serwisowych nie dłuższych niż 2 h w tygodniu przy założeniu stabilnego połączenia internetowego. Pozostałe moduły powinny działać bez problemów na oddzielnych serwerach.
Wydajność (Performance)	Aplikacja WWW powinna działać płynnie na każdym komputerze z dowolnym systemem operacyjnym na 1 z wcześniej wymienionych przeglądarek. Pozostałe moduły powinny uruchamiać się same co określony odstęp czasu. Początkowo planowane jest uruchamianie ich raz dziennie.
Wsparcie (Supportability)	W razie jakichkolwiek problemów w aplikacji WWW dostępny jest formularz kontaktowy.

## 1.5 Schemat architektury

Przewidziana architektura ma strukturę modułową. Schemat komponentów oraz ich połączenie przedstawia załączony na końcu sekcji diagram.

W systemie wyróżnić możemy cztery główne, niezależne od siebie (na tyle że mogą, a nawet powinny, być uruchamiane na różnych maszynach) moduły. Ta sekcja zawiera ich ogólny, wysokopoziomowy opis. Szczegóły dotyczące architektury i implementacji znajdują się w następnym rozdziale.

### 1.5.1 MODUŁ ZBIERAJĄCY DANE

Komponent ten odpowiada za automatyczne pobieranie ogłoszeń z serwisu zewnętrznego. Jest to program, który cyklicznie łączy się z udostępniającym ogłoszenia serwisem i automatycznie pobiera ich treść. Z pobranych ogłoszeń w najprostszy sposób (poprzez analizę kodu HTML) wyłuskuje podstawowe elementy, takie jak:

- Tytuł ogłoszenia
- Treść
- Data dodania
- Pracodawca
- Miejsce pracy

W kolejnym kroku tak “rozbite” ogłoszenie przesyła do kolejnego komponentu systemu.

Opisany proces nie obejmuje zapisu do żadnej bazy danych. W tej kwestii moduł zbierający dane polega całkowicie na komponencie, do którego przekazuje pobrane ogłoszenia. To samo dotyczy kwestii rozpoznawania duplikatów - program przed każdym rozpoczęciem skanowania prosi swojego “odbiorcę” o listę już zeskanowanych ogłoszeń aby uniknąć przetwarzania ich ponownie.

### 1.5.2 PIPELINE

Zebrane ogłoszenia trafiają do komponentu *Pipeline* (Łańcucha przetwarzania). Nazwa komponentu odnosi się do zasady jego działania. Odbierane ogłoszenia przekazywane są przez kolejne podmoduły, które wykonują na nich stosowne operacje.

Pierwszym elementem jest moduł przechowujący dane. Odbiera on nowe ogłoszenia od modułu zbierającego i zapisuje je w bazie danych. Jednocześnie oferuje usługę odczytania listy kluczy (adresów URL) dodanych już ogłoszeń, z czego moduł zbierający korzysta przed rozpoczęciem skanowania serwisu zewnętrznego.

Następnie ogłoszenia trafiają do modułu ekstrakcji kluczy. Tutaj zgodnie z wymaganiami funkcjonalnymi systemu z ogłoszenia wyłuskane są elementy z trzech kategorii:

- Obszar branży IT którego dotyczy ogłoszenie
- Stanowisko
- Technologie i umiejętności

Po ukończeniu procesu model obiekt ogłoszenia powiększa się o zestaw kluczy, a następnie zostaje wysłany do kolejnego komponentu.



### 1.5.3 BACKEND

Moduł ten zajmuje się dostarczaniem użytkownikowi (a konkretnie aplikacji WWW) informacji na podstawie przeanalizowanych ofert. Do bazy danych zapisywane są ogłoszenia wraz z kluczami, a podmoduły pozwalają na jej odpytywanie.

Moduł wyszukiwarki wystawia interfejs pozwalający przeglądarce na wykonanie zapytania o ogłoszenia zawierające jednej lub więcej kluczy.

Interfejs modułu statystyk pozwala na dostęp do statystyk systemu jako całości oraz statystyk poszczególnych kluczy. Bardziej szczegółowe wymagania tego interfejsu znajdują się w sekcji wymagań funkcjonalnych.

### 1.5.4 APLIKACJA WWW

Ostatnim elementem systemu, jedynym dostępnym bezpośrednio dla użytkownika jest aplikacja WWW. Nie posiada ona własnej bazy, a co za idzie kont użytkowników. Korzysta wyłącznie z interfejsu wyszukiwania oraz statystyk pozwalając użytkownikowi na przejrzysty i wygodny dostęp do informacji.



## 1.6 Model wytwórczy

Do wytworzenia opisywanego systemu wybraliśmy metodykę zwinną - Agile. Jest ona jedną z najszybszych metod wytwarzania oprogramowania i idealnie wpasowuje się w modułową strukturę architektury systemu. Wybór tej metodyki oznacza prowadzenie prac w modelu iteracyjnym - komponent po komponencie. W naszym przypadku oznacza to, że z każdą kolejną iteracją gotowy projekt będzie powiększał się o nowy, w pełni działający fragment. Po pierwszym etapie będzie to moduł zbierający dane z zewnętrznych serwisów, następnie moduł przechowujący i udostępniający zebrane dane, i tak dalej.

Wymagania funkcjonalne projektu zostały opracowane w pierwszej kolejności. Pozwoliło to na nakreślenie zarysu systemu i tego na co musi pozwalać, bez wdawania się w szczegóły implementacyjne. Po wyodrębnieniu komponentów systemu, uznaliśmy że podejście kaskadowe byłoby zbyt ryzykowne. Zaprojektowanie dokładnej struktury i szczegółowych wymagań implementacji każdego z nich już na wstępie, niosłoby za sobą ryzyko niedopasowania się do przewidzianego na implementację czasu. Zachodziłoby również niebezpieczeństwo przeoczenia błędów w planowaniu, które wyszłyby na powierzchnię dopiero w momencie implementacji bądź testów, kiedy metoda kaskadowa nie przewiduje już zmiany wymagań ani planu. Jakość końcowego produktu zdecydowanie bardzo by na tym ucierpiała.

Komponenty systemu są na tyle niezależne i wyizolowane, że mając opracowane ogólne ich założenia i przeznaczenie, podczas każdej z iteracji jesteśmy w stanie skupić się na nich indywidualnie, bez ingerowania w resztę projektu. Pozwoli to na dokładne ich dopracowanie, co jest szczególnie istotne biorąc pod uwagę, że część z nich opiera się w znacznym stopniu na zewnętrznych projektach i technologiach, które do udanej integracji będą wymagały głębszego poznania.

## 1.7 Harmonogram

Załączona tabela 1.2 przedstawia planowany harmonogram postępów w pracy nad projektem - przewidywane terminy ukończenia każdego z modułów. Wytłuszczone daty to terminy laboratoriów na których przewidziana jest kontrola postępów.

Tabela 1.2: Harmonogram

Rozpoczęcie iteracji	Ukończenie iteracji	Opis	Moduły
26.10.2017	<b>02.11.2017</b>	Przygotowanie harmonogramu oraz wstępnych wymagań	
02.11.2017	09.11.2017	Moduł zbierający dane Tydzień 1	
09.11.2017	<b>16.11.2017</b>	Moduł zbierający dane Tydzień 2	1/6
16.11.2017	23.11.2017	Moduł przechowujący dane	2/6
23.11.2017	<b>30.11.2017</b>	Moduł ekstrakcji kluczy Tydzień 1	
30.11.2017	07.12.2017	Moduł ekstrakcji kluczy Tydzień 2	
07.12.2017	<b>14.12.2017</b>	Moduł ekstrakcji kluczy Tydzień 3	3/6
14.12.2017	21.12.2017	Moduł statystyk Tydzień 1	
21.12.2017	28.12.2017	Moduł statystyk Tydzień 2	4/6
28.12.2017	04.01.2018	Moduł wyszukiwarki	5/6
04.01.2018	<b>11.01.2018</b>	Strona WWW	6/6

## Rozdział 2

# Specyfikacja komponentów systemu

### 2.1 Moduł zbierający dane

Kod źródłowy znajduje się pod adresem: [github.com/jobsbrowser/scrapper](https://github.com/jobsbrowser/scrapper).

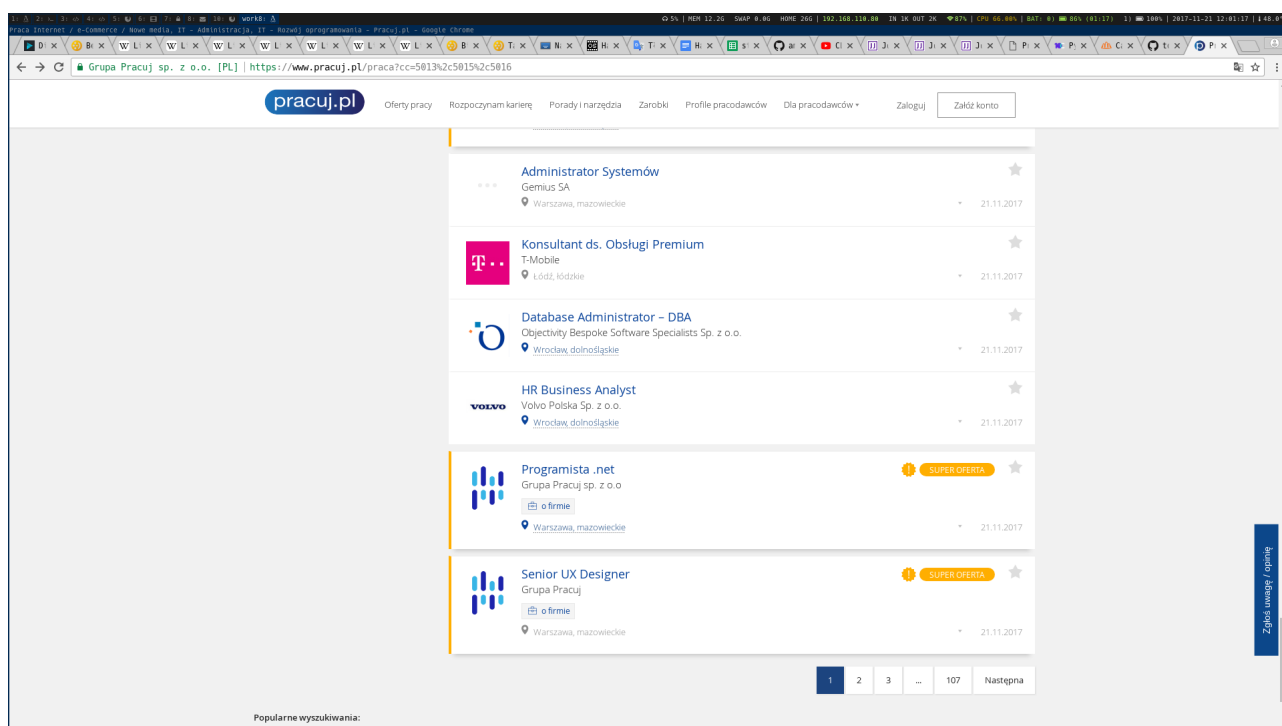
Komponent zajmuje się automatycznym pobieraniem ofert z serwisu zewnętrznego, ekstrakcją podstawowych informacji oraz przesłaniem tak przetworzonych ofert do kolejnego modułu. Zdecydowaliśmy się na serwis Pracuj.pl jako źródło ofert oraz na framework **Scrapy**(Scrapy n.d.) jako bazę naszego modułu.

#### 2.1.1 SERWIS ZEWNĘTRZNY

Pracuj.pl dzieli zamieszczone w nim oferty na kategorie. My, z racji tematyki pracy skupiamy się wyłącznie na trzech z nich:

- Internet / e-Commerce / Nowe media
- IT - Administracja
- IT - Rozwój oprogramowania

Widok listy ogłoszeń w serwisie umożliwia wybór kategorii, z których ogłoszenia chcemy zobaczyć. Skorzystaliśmy z tej możliwości, aby uzyskać bazowy link od którego zaczniemy pobieranie ofert.



Pracuj.pl przy zbiorczym wyświetlaniu ofert używa paginacji. Oznacza to, że scraper musi poradzić sobie nie tylko z pobieraniem podstron poszczególnych ofert, ale też z poruszaniem się pomiędzy ponumerowanymi stronami listy.

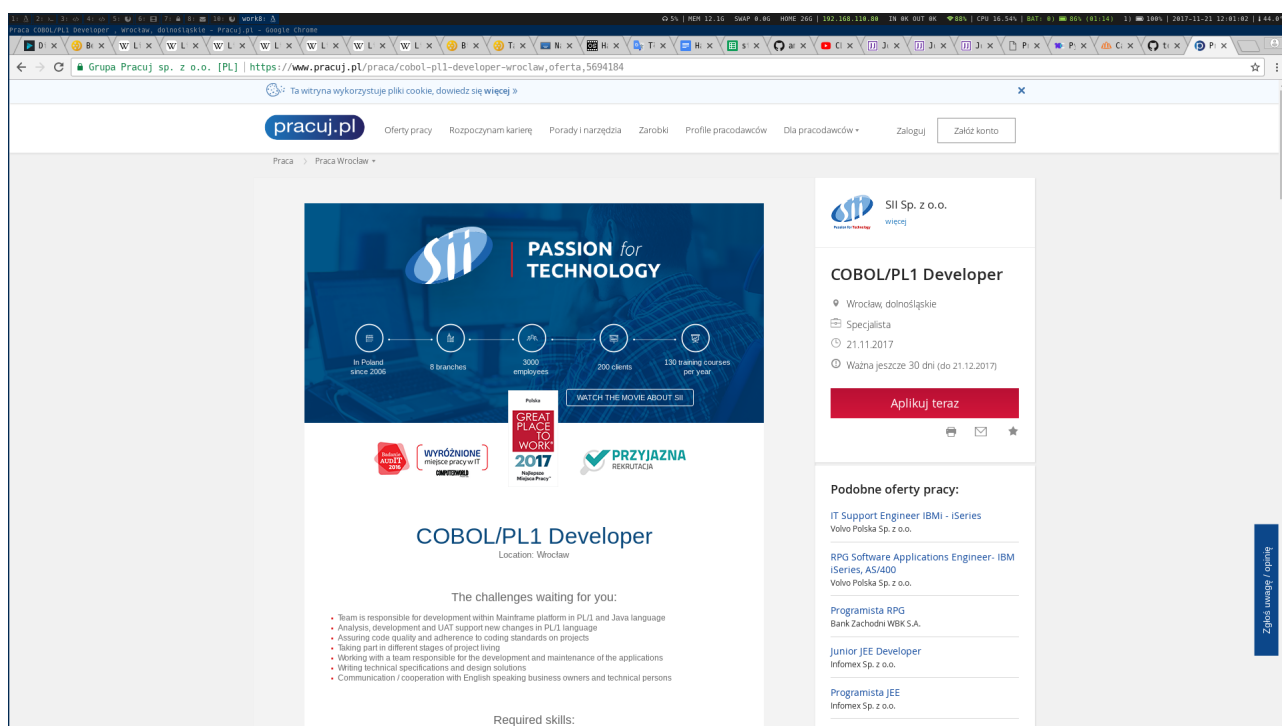
Do każdej z ofert na stronie (których znajduje się ok. 50) prowadzi bezpośredni link, który można wyłuskać z kodu HTML listy. Podstrona pojedynczej oferty zawiera wszystkie interesujące nas informacje również możliwe do wyłuskania z kodu HTML przy użyciu odpowiednich selektorów CSS. Dostępne w przystępny sposób informacje to:

- Data dodania oferty
- Data wygaśnięcia
- Nazwa dodającego (pracodawca)
- Lokalizacja (miasto i województwo)
- Tytuł oferty
- Treść oferty

Ponadto, w łatwy sposób można uzyskać również dwie wartości jednoznacznie identyfikujące ofertę:

- Adres URL oferty
- ID (będące częścią adresu URL)

Te dwie wartości z powodzeniem mogą służyć za klucz pozwalający np. szybko sprawdzać czy oferta jest już w bazie systemu - czyli czy została już kiedyś przetworzona.



Rysunek 2.2: Widok oferty w witrynie pracuj.pl.



### 2.1.2 NAPOTKANE PROBLEMY

Zadaniem stojącym przed scraperem jest automatyczne przejście po wszystkich dostępnych stronach listy, wyłuskanie zeń adresów poszczególnych ofert, a stamtąd wszystkich potrzebnych informacji. Pobrane ogłoszenie z wydzielonymi fragmentami powinno trafić do innego komponentu systemu, który zajmie się jego zapisem czy dalszym przetwarzaniem. Podczas prac nad modułem natrafiliśmy na kilka kwestii które wymagały opracowania konkretnego rozwiązania.

#### 2.1.2.1 Aktualizacja ofert w serwisie

Ofert na stronie wraz z upływem czasu będzie przybywać - i dla naszego systemu jest to bardzo istotne. Aby zapewnić stały przypływ ogłoszeń z serwisu Pracuj.pl scraper został tak przygotowany, aby mógł być uruchamiany cyklicznie. Przewidzianym interwałem jest 12 godzin, choć nie jest to wartość zdefiniowana w programie. To od uruchamiającego program na komputerze zależy jak ją ustawi.

#### 2.1.2.2 Utrzymywanie stanu scrapera

Uruchomienie cykliczne wraz z brakiem stanu (bo przecież wszystkie przetworzone ogłoszenia trafiają do osobnego modułu) wiąże się z możliwością niechcianego przetwarzania jednej oferty wielokrotnie - przy każdym kolejnym uruchomieniu programu. Zdecydowaliśmy się na rozwiązanie tego programu przez zaimplementowanie w scraperze możliwości pobrania z modułu do którego trafiają dane kluczy (adresów URL) tych danych które już tam są. Oznacza to, że scraper przy każdym uruchomieniu przetworzy wszystkie dostępne strony, ale nie będzie pobierał ani przetwarzał podstron ofert które już ma na liście. Próba pobrania listy wykonywana jest po uruchomieniu programu, przed rozpoczęciem skanowania. Jeżeli nie uda się jej otrzymać (serwer nie odpowie, lub odpowie z błędem), program wypisze w konsoli stosowny komunikat ostrzegający i przejdzie do przetwarzania wszystkich ofert.

#### 2.1.2.3 Zabezpieczenia serwisu przed zbieraniem danych

Wartym wspomnienia jest fakt stosowania przez serwis Pracuj.pl zabezpieczeń utrudniających automatyczne zbieranie danych. Podczas pierwszych testów naszego modułu wszystkie wychodzące żądania HTTP miały nagłówek `User-Agent` ustawiony na nazwę naszego projektu - `jobsbrowser`. Nie zauważyliśmy wtedy żadnych utrudnień ani trudności związanych z uzyskaniem przez scrapera dostępu do zasobów serwisu. Po kilku dniach jednak, sytuacja się

zmieniała. Okazało się, że wszystkie nasze żądania (nawet z innych adresów IP) podpisane jako `jobsbrowser` były odrzucane przez serwer. Konieczne było wprowadzenie poprawki w kodzie modułu, tak żeby nagłówkiem `User-Agent` imitował przeglądarkę internetową. Wybór padł na Google Chrome w wersji 41 i to rozwiązało problem.

### 2.1.3 SCRAPY

Do napisania scrapera zdecydowaliśmy się na framework Scrapy. Jest to framework napisany w języku Python, przy wykorzystaniu biblioteki Twisted. Dzięki temu działa całkowicie asynchronicznie, co czyni go niezwykle wydajnym przy relatywnej łatwości pisania własnych scraperów. Wydany jest na licencji BSD3.

### 2.1.4 URUCHOMIENIE I UŻYTKOWANIE

W katalogu `jobsbrowser` repozytorium znajduje się wykonywalny skrypt `run.sh`. Nie przyjmuje on żadnych parametrów, a jego uruchomienie powoduje uruchomienie procesu scraper'a. W celu cyklicznego uruchamiania, zalecamy skorzystanie z menadżera *cron*.

Za pomocą parametru `--log-level=LOG_LEVEL`, gdzie `LOG_LEVEL` może przyjąć jedną z poniższych wartości (wartości wymienione są od najmniej restrykcyjnej do najbardziej):

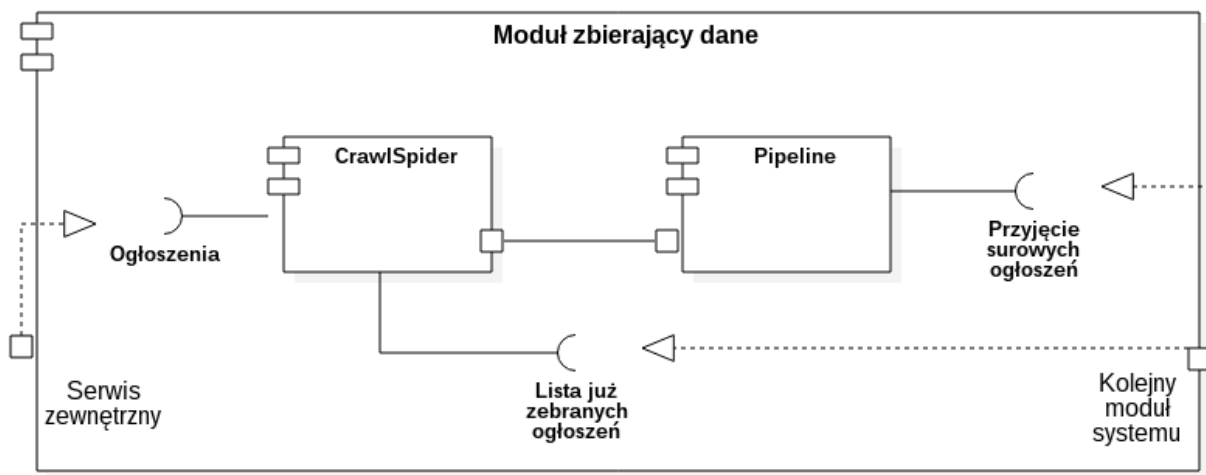
- `DEBUG`
- `INFO`
- `WARNING`
- `ERROR`
- `CRITICAL`

Innym użytecznym parametrem podczas weryfikowania działania scraper'a jest `-o/--output filename.EXTENSION`, gdzie `EXTENSION` może przyjąć jedną z niżej wymienionych wartości:

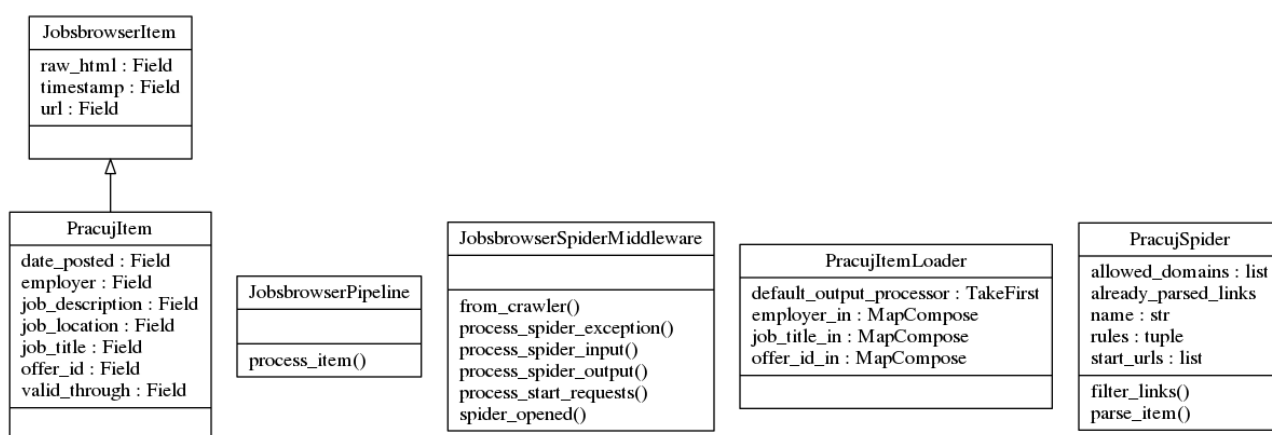
- `json`
- `jl` (json lines) każda linia jest oddzielnym obiektem JSON
- `csv`

Po podaniu tego parametru wszystkie dane zebrane przez scraper zostaną zapisane do podanego pliku w wybranym na podstawie rozszerzenia formacie.

W pliku `jobsbrowser/jobsbrowser/settings.py` znajdują się ustawienia programu. Większość z nich to ustawienia framework'a Scrapy. Ich opis znaleźć można w jego dokumentacji(Scrapy n.d.). Ustawieniami dotyczącymi konkretnie tego projektu są `STORAGE_SERVICE_ADD_URL` oraz `STORAGE_SERVICE_RETRIEVE_URL`, które oznaczają adresy URL pod który program może wykonać żądania w celu odpowiednio, dodania nowo przetworzonej strony oraz uzyskania listy adresów URL ofert których nie powinien przetwarzać.



Rysunek 2.3: Schemat architektury modułu Scraper'a.



Rysunek 2.4: Diagram klas modułu Scraper'a.

### 2.1.5 STRUKTURA KODU ŹRÓDŁOWEGO

Poniżej prezentujemy drzewo katalogów oraz plików projektu wraz z krótkim omówieniem:

```
jobsbrowser
  jobsbrowser
    __init__.py
    items.py          # definicje klas przechowujących zebrane dane
    loaders.py        # definicje klas ładujących zebrane dane
    middlewares.py
    pipelines.py      # definicje kolejnych etapów przetwarzania danych
    settings.py       # ustawienia pajków
    spiders           # katalog z definicjami pajków zbierających dane
      __init__.py
      pracuj.py       # definicja pajka zbierającego dane ze strony pracuj.pl
  tests/             # katalog z testami projektu
  run.sh             # plik wykonywalny uruchamiający proces zbierania danych
  scrapy.cfg         # konfiguracja całego projektu
  requirements.txt   # plik z wymaganiami projektu
```

### 2.1.6 GŁÓWNE KLASY MODUŁU

- **JobBrowserItem** - klasa bazowa definiująca pola które powinny być dostarczane przez wszystkie spider'y wykorzystywane w projekcie.
- **PracujItem** - klasa przechowująca pola które muszą zostać wypełnione przez scraper'y zbierające dane ze strony pracuj.pl.
- **JobsBrowserPipeline** - klasa która po zebraniu danych ze stron wysyła dane do modułu zapisy do bazy danych.
  - `process_item` - metoda w której za pomocą zapytania HTTP wysyłany jest aktualnie przetwarzany element(Item) do modułu bazy danych.
- **JobsBrowserSpiderMiddleware** - klasa wygenerowana automatycznie podczas tworzenia projektu za pomocą Scrapy'ego.
- **PracujItemLoader** - klasa przetwarzająca w prosty sposób zebrane dane.
  - `load_item` - wykonuje wszystkie operacje zdefiniowane w deskryptorach na aktualnie przetwarzanym elemencie

- **PracujSpider** - klasa wykonująca zapytania do serwisu pracuj.pl oraz zbierająca dane.
  - `start_urls` - adresy URL od których zaczynane jest zbieranie danych.
  - `rules` - zasady definiujące jakie elementy są ofertami oraz jak przejść do następnej strony z ofertami.
  - `already_parsed_links` - linki do ofert znajdujących się już w bazie.
  - `filter_links` - metoda sprawdzająca czy ogłoszenia z danej strony nie są już w bazie. Zwraca nowe ogłoszenia.
  - `parse_item` - zajmuje się wyciągnięciem potrzebnych danych z aktualnie przetwarzanej strony za pomocą `PracujItemLoader`'a.

### 2.1.7 TESTY

W kodzie źródłowym modułu znajdują się testy jednostkowe pozwalające na przetestowanie poprawności zaimplementowanych metod i funkcji. W tej sekcji znajduje się również opis przykładowego scenariusza testów akceptacyjnych. Testy integracyjne na tym etapie nie są jeszcze przewidziane. Moduł zbierania danych jest pierwszym modułem i bez implementacji pozostałych ich wykonanie jest niemożliwe.

#### 2.1.7.1 Testy jednostkowe

Aby uruchomić testy jednostkowe w konsoli należy wpisać polecenie `pytest`.

Poniżej przedstawiamy listę plików z testami jednostkowymi oraz opis poszczególnych funkcji lub metod:

- `test_pracuj_item_loader.py` - plik z testami klasy `PracujItemLoader`.
  - `test_taking_first_from_each_field` - test sprawdza czy żaden z przetworzonych pól nie jest listą (`Scrapy` domyślnie zwraca wszystkie elementy jako listy, nawet tylko gdy znajduje się w nich jeden element).
  - `test_offer_id_properly_extracted` - test sprawdza czy pole `offer_id` jest odpowiednio wydobywane z adresu URL strony.
  - `test_remove_html_tags_from_employer_and_job_title_fields` - test sprawdza czy znaczniki HTML zostały poprawnie usunięte z wartości pól `employer` oraz `job_title`.
- `test_jobsbrowser_pipeline.py` - plik z testami klasy `JobsBrowserPipeline`.

- `test_jobsbrowser_pipeline_process_item_send_request_to_db_module` - test sprawdza czy w metodzie `process_item` wykonywane jest zapytanie HTTP POST do serwera z działającym modulem bazy danych.
- `test_pracuj_spider.py` - plik z testami klasy `PracujSpider`.
  - `test_parse_item` - test sprawdza czy metoda prawidłowo wyciąga dane z adresu URL oraz treści strony, a następnie zwraca je w atrybutach obiektu typu `PracujItem`.
  - `test_parse_on_page_with_multiple_next_pages` - test sprawdza czy metoda `parse` prawidłowo znajduje link do następnej strony z ofertami. Testowany w tej metodzie jest przypadek gdy istnieją kolejne strony.
  - `test_parse_on_last_page` - test sprawdza czy metoda `parse` prawidłowo zachowuje się na ostatniej stronie z listingami ofert, czyli nie znajduje żadnych nowych linków, tym samym kończąc zbieranie danych.

### 2.1.7.2 Testy akceptacyjne

Moduł zbierania danych zajmuje się, jak mówi sama nazwa, jedynie ich zbieraniem. Domyślnie nie są one nigdzie przechowywane, ani zapisywane. Podejście takie utrudnia nieco przygotowanie testów akceptacyjnych obejmujących wyłącznie ten komponent, ale nie uniemożliwia, co zaraz wykażemy.

Zasada działania programu (uruchamianego przez `run.sh`) jak opisano już wcześniej sprowadza się do zbierania ofert i wysyłania ich do kolejnego modułu systemu (z wypisaniem stosownego komunikatu, jeśli ten nie odpowiada). Bez tego komponentu, nie zobaczymy nigdzie zebranych ofert, ani też nie dostarczymy scraperowi listy już zebranych, co będzie skutkowało zebraniem wszystkich. Nie są to warunki idealne na testy akceptacyjne - gdzie przecież chcemy upewnić się że komponent faktycznie działa. Wykorzystamy jednak fakt że skrypt uruchamiający przekazuje swoje parametry do procesu scrapera, co pozwala na nadpisanie jego ustawień na czas uruchomienia. Dzięki temu ograniczymy zbiór przetwarzanych ofert (do jednej strony) i zapiszemy je na dysku, aby przekonać się że interesujące nas elementy faktycznie zostały z ofert wyłuskane.

Aby wykonać takie polecenie testujące sprawność scrapera, do skryptu musimy przekazać kilka dodatkowych argumentów:

```
./run.sh -s DEPTH_LIMIT=1 -o oferty.json
```

Oznaczają one odpowiednio:

- `-s DEPTH_LIMIT=1` - nadpisanie ustawień scrapera dotyczących maksymalnej “głębokości” na jaką może się zapuścić. W naszym przypadku oznacza to liczbę przetworzonych

stron

- `-o oferty.json` - wymusza zapis przetworzonych obiektów do pliku `oferty.json`

Ponadto uruchomieniu towarzyszyć będą wypisywane w terminalu komunikaty informujące o przetworzeniu danej oferty oraz próbie wysłania jej do sąsiedniego komponentu. Mówią one użytkownikowi czym program aktualnie się zajmuje i na jakie trafia problemy. Po zakończeniu w katalogu w którym znajduje się wywołany skrypt znajdziemy plik `oferty.json` który zawiera zebrane oferty. Z powodu przechowywania w obiekcie również surowej wersji przetwarzanej strony (w postaci kodu HTML) nie jest on szczególnie czytelny dla człowieka, jednak bez problemu można wykonywać na nim dowolne operacje, np. z poziomu innego programu.



```

2017-11-23 00:42:11 bsdell pracuj[17637] INFO Offer 5570127 scraped. Sending to storage service...
2017-11-23 00:42:11 bsdell pracuj[17637] WARNING Sending offer 5570127 failed. Service storage is unavailable.
2017-11-23 00:42:11 bsdell pracuj[17637] INFO Offer 5570005 scraped. Sending to storage service...
2017-11-23 00:42:11 bsdell pracuj[17637] WARNING Sending offer 5570005 failed. Service storage is unavailable.
2017-11-23 00:42:13 bsdell pracuj[17637] INFO Offer 5570061 scraped. Sending to storage service...
2017-11-23 00:42:13 bsdell pracuj[17637] WARNING Sending offer 5570061 failed. Service storage is unavailable.
2017-11-23 00:42:13 bsdell scrapy.core.engine[17637] INFO Closing spider (finished)
2017-11-23 00:42:13 bsdell scrapy.extensions.feedexport[17637] INFO Stored json feed (47 items) in: oferty.json

```

Rysunek 2.5: Komunikaty w oknie terminala

```

In [18]: with open("oferty.json") as f:
...:     offers = json.load(f)
...:

In [19]: assert len(offers) == 47

In [20]: for offer in offers[:5]:
...:     print(offer['job_title'])
...:
...:
Programista ASP.NET MVC 6
Specjalista ds. Marketingu
Programme and Project Services Officer
Analityk Hurtowni Danych - Konsultant BI
Oracle ERP Cloud PM

```

Rysunek 2.6: Przykładowy dostęp do wynikowego pliku JSON

## 2.2 Moduł przechowujący dane

Kolejnym komponentem systemu jest moduł odpowiedzialny za przechowywanie danych. Przez dane rozumiemy w tym przypadku oferty w postaci “pół-surowej”. Tzn. będące już po wstępnym, prostym procesie przetwarzania w module zbierania danych, ale jeszcze przed najbardziej znaczącym procesem ekstrakcji kluczy. Są to więc dane które są bazą dla przyszłych działań systemu, ale w obecnej formie nie dostarczają wielu informacji.

### 2.2.1 WYMAGANIA

Moduł z pozostałymi komponentami systemu łączy się dwiema drogami:

- Przez interfejs HTTP wykorzystywany przez moduł zbierania danych (poprzedni komponent)
- Poprzez dodawanie nowych zadań do kolejki, zbieranych i wykonywanych przez moduł ekstrakcji kluczy (następny komponent)

Wymagania funkcjonalne modułu sprowadzają się więc do obsłużenia obydwu kierunków komunikacji. Interfejs HTTP powinien pozwalać na dwie operacje:

- Dodanie oferty do bazy (nadpisując jeśli oferta z takim adresem URL już istnieje)
- Pobranie listy adresów URL ofert zapisanych już w bazie

Natomiast na drodze komunikacji z kolejnym modułem:

- Rozpoczęcie nowego zadania Celery z dokumentem oferty przekazany jako argument, po zapisie oferty do bazy.

Wymagania niefunkcjonalne modułu sprowadzają się natomiast do:

- **niezawodności** - usługa powinna być dostępna możliwie cały czas. Nie jest to jednak kwestia kluczowa, ponieważ stosunkowo krótkie braki w dostępności ( rzędu maksymalnie kilku dni) nie ciągną za sobą konsekwencji. Jeżeli moduł zbierający dane nie uzyska odpowiedzi od modułu zajmującego się ich przechowywaniem, informacje o tej ofercie nie zostaną nigdzie zapisane. Kiedy usługa przechowywania będzie ponownie dostępna, na zapytanie scrapera o listę ofert będących już w bazie zwróci tę sprzed awarii, wszystkie pominięte oferty zostaną więc ostatecznie dodane.

- **wydajności** - liczba nadchodzących ofert może być potencjalnie duża, proces zapisu do bazy powinien być więc jak najmniej skomplikowany i efektywny, aby uniknąć spadków na wydajności z powodu niewydajnych zapytań. Podobnie jak w kwestii niezawodności, nie jest to jednak wymaganie kluczowe. Spadki w wydajności nie będą bowiem objawiać się wolniejszym działaniem aplikacji przeznaczonej dla użytkownika końcowego, a jedynie późniejszym pojawianiem się w niej nowych ofert.

### 2.2.2 INTERFEJS

Interfejs HTTP zaimplementowany jest przy użyciu micro-frameworka Flask(Flask n.d.). Flask jest wykorzystywany do tworzenia stron internetowych oraz REST API. Zdecydowaliśmy się na niego ze względu na to, że jest to framework dojrzały, idealny do małych lub średnich projektów, a w razie wzrostu skali projektu umożliwia wygodne skalowanie. Flask zawiera również wiele świetnych rozszerzeń, np. rozszerzenie integrujące go z bazą MongoDB z której korzystamy w projekcie.

### 2.2.3 BAZA DANYCH

Wykorzystanym silnikiem bazy danych jest MongoDB(MongoDB n.d.). Zdecydowaliśmy się na bazę relacyjną z kilku powodów. Po pierwsze tak naprawdę jedynym typem trzymanych w niej danych są same oferty. Oznacza to że w bazie relacyjnej mielibyśmy tylko jedną tabelę - bez żadnych relacji czy potrzeby zachowania spójności. Nie ma potrzeby również stosowania mechanizmu transakcji, czy skomplikowanych zapytań. Tym czego faktycznie oczekujemy od bazy jest wydajność, dostępność oraz ewentualna skalowalność. Wybór był więc prosty. Struktura dokumentów przechowywanych w bazie jest identyczna jak struktura zebranego ogłoszenia. Dla przypomnienia, pola jakie wyróżniamy w dokumencie oferty to:

- Adres URL
- Czas w którym pobrano ofertę
- Kod HTML strony z ofertą
- ID oferty w systemie pracuj.pl
- Data dodania
- Data ważności
- Podmiot dodający
- Tytuł oferty
- Miejsce pracy
- Kod HTML treści oferty (rozbity na opis, kwalifikacje oraz benefity - wg struktury Pracuj.pl)

## 2.2.4 INTEGRACJA Z KOLEJNYM MODUŁEM

Początkowe plany zakładały użycie w tym miejscu (po dodaniu oferty do bazy) frameworka *Luigi*. Jest to stworzone przez twórców aplikacji *Spotify* narzędzie do łączenia ze sobą kolejnych funkcji / etapów tworząc łańcuch przetwarzania (wspomniany w kilku miejscach tzw. *Pipeline*). Okazało się jednak, że przewidziane zastosowania narzędzia obsługują etapy nieco innego typu niż te do zaimplementowania w module ekstrakcji kluczy. Luigi przeznaczony jest bowiem do łączenia bardzo wymagających zadań, angażujących wiele zewnętrznych usług czy języków programowania i wykonujących się nawet kilka dni. W efekcie nie udostępnia chociażby tak podstawowej w mniejszych zastosowaniach możliwości, jak uruchamianie zadania z poziomu kodu źródłowego. W grę wchodzi jedynie linia poleceń. Zdecydowaliśmy się więc na porzucenie go, na rzecz frameworka *Celery* którego obsługa zadań jest tym czego potrzebujemy. Stracimy co prawda na odporności na awarie (Luigi zapisuje stan po każdym zadaniu, i wraca do niego po awarii), lecz zdecydowanie zyskamy na łatwości implementacji.

Użycie frameworka Celery jest częścią kolejnego modułu, tj. modułu ekstrakcji kluczy, więc to tam znajdzie się dotycząca tej kwestii dokumentacja.

## 2.2.5 URUCHOMIENIE

Do uruchomienia API korzystamy z polecenia `python manage.py runserver`. Aby uruchomić usługę z odpowiednimi ustawieniami trzeba ustawić zmienną środowiskową `APP_CONFIG` na wartość `PRODUCTION`. Możemy to zrobić korzystając z 1 z poniższych poleceń:

- `export APP_CONFIG="production" && python manage.py runserver`
- `APP_CONFIG="PRODUCTION" python manage.py runserver`

## 2.2.6 STRUKTURA KODU

Poniżej prezentujemy drzewo katalogów oraz plików modułu wraz z krótkim omówieniem.

```
jobsbrowser
├── api                                # katalog modułu API
│   ├── __init__.py
│   ├── resources.py                # definicja endpointów API
│   ├── settings.py                 # ustawienia API
│   └── spec.yml                    # specyfikacja w standardzie OpenAPI(swagger)
```

```

manage.py          # skrypt z pożytecznymi komendami dotyczącymi API
tests              # katalog z testami modułu
    __init__.py
    test_api_resources.py  # testy endpointów API
requirements.txt   # wymagania modułu

```

## 2.2.7 GŁÓWNE KLASY MODUŁU

- `add_offer` - funkcja implementująca dodawanie otrzymanej oferty do bazy danych.
- `get_offers` - funkcja implementująca pobieranie aktualnych ofert (takich których data w polu *valid\_through* jest większa od daty dzisiejszej) znajdujących się w bazie danych.
- `init_app` - funkcja tworząca aplikację z podaną konfiguracją (jako parametr *config\_name* lub zmienna środowiskowa `APP_CONFIG`).
- `BaseConfig` - klasa z bazową, fundamentalną konfiguracją.
- `DevConfig` - klasa przechowująca konfigurację developerską.
- `ProductionConfig` - klasa przechowująca konfigurację produkcyjną.
- `TestingConfig` - klasa przechowująca konfigurację testową.

## 2.2.8 TESTY

### 2.2.8.1 Testy jednostkowe

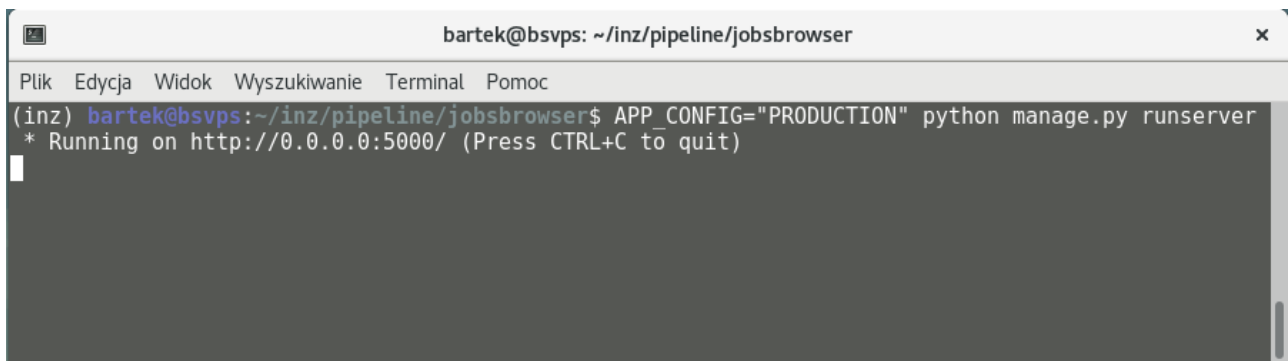
Aby uruchomić testy jednostkowe, w konsoli należy wpisać polecenie `pytest`. Poniżej przedstawiamy opis testów znajdujących się w pliku `jobbrowser/tests/test_api_resources.py`:

- `test_ping_resource_returns_pong` - test sprawdza czy endpoint */pong* (który jest wykorzystywany do sprawdzania czy usługa API jest aktywna) zwraca odpowiedź ze statusem 200 oraz wartością "pong".
- `test_add_offer_resource_try_add_offer_to_mongo_db` - test sprawdza czy wysłana poprzez zapytanie POST oferta próbuje być zapisana do bazy danych.
- `test_get_offers_resource_query_mongo_db` - test sprawdza czy po wykonaniu zapytania HTTP GET na endpoint */offers* wykonywana jest próba pobrania danych z bazy danych.

### 2.2.8.2 Testy akceptacyjne

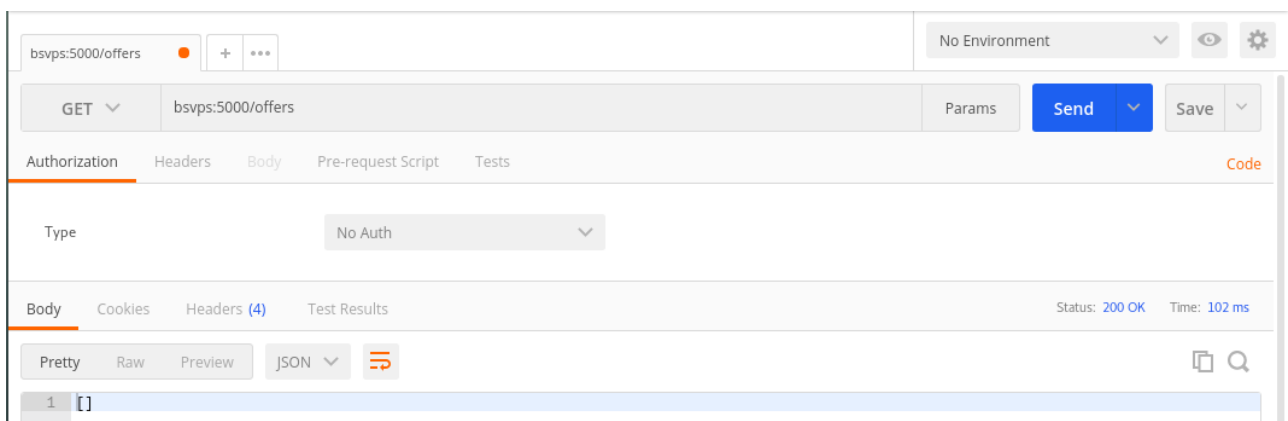
Interfejs modułu zbierania danych jest dość ubogi. Zajmuje się on bowiem jedynie dodawaniem ofert oraz zwracaniem informacji o adresach URL tych już istniejących. Test jaki możemy przeprowadzić aby upewnić się moduł działa poprawnie może więc polegać na:

- Uruchomieniu usługi wg instrukcji z dokumentacji
- Wykonaniu zapytania o listę ofert (powinna być pusta)
- Wykonaniu żądania dodającego nową ofertę
- Wykonaniu zapytania o listę ofert raz jeszcze. Powinniśmy otrzymać adres URL przesłany w poprzednim kroku.

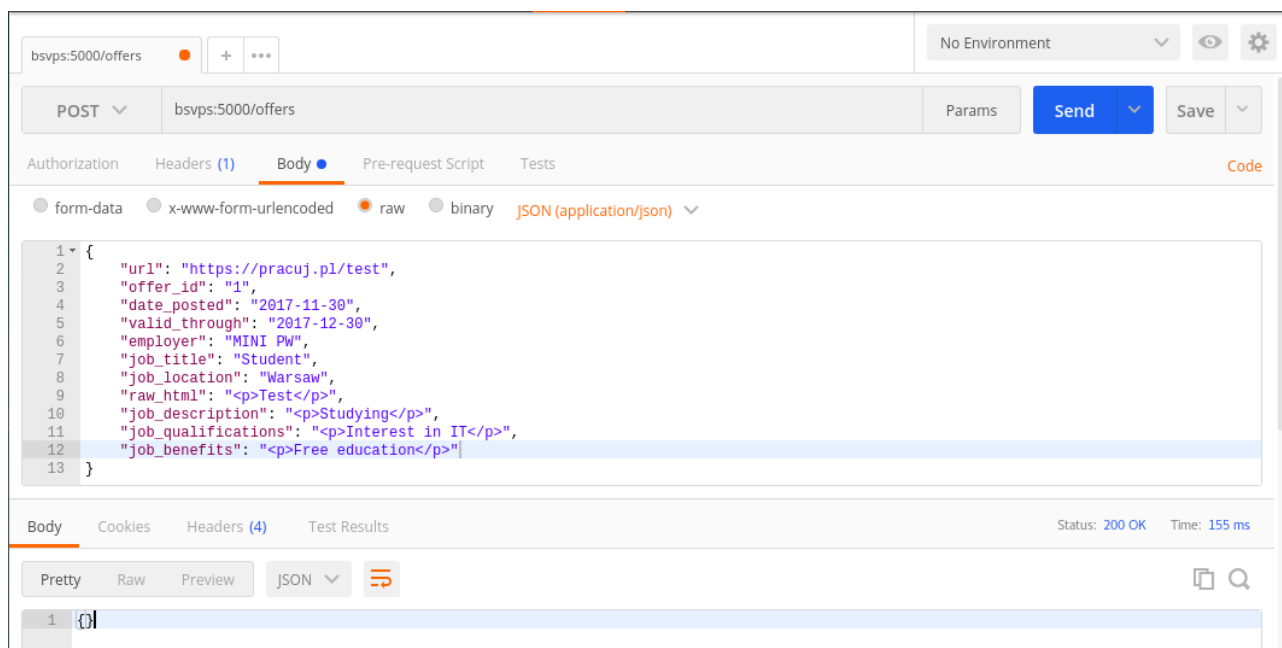


A terminal window titled "bartek@bsvps: ~/inz/pipeline/jobbrowser". The window contains a menu bar with "Plik", "Edycja", "Widok", "Wyszukiwanie", "Terminal", and "Pomoc". The command prompt shows the user entering the command: `(inz) bartek@bsvps:~/inz/pipeline/jobbrowser$ APP_CONFIG="PRODUCTION" python manage.py runserver`. The output of the command is: `* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)`. A cursor is visible on the line following the output.

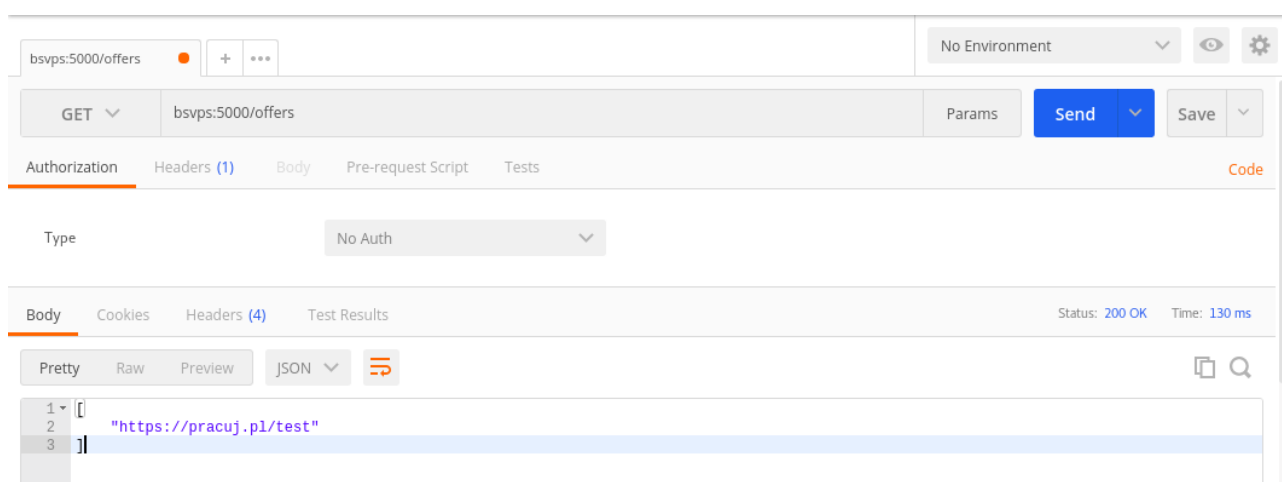
Rysunek 2.7: Uruchomienie serwera



Rysunek 2.8: Pierwsze zapytanie o listę



Rysunek 2.9: Dodanie nowej oferty



Rysunek 2.10: Ponowne zapytanie o listę



## 2.3 Moduł ekstrakcji kluczy

# Historia zmian dokumentu

Tabela 2.1: Historia Zmian

Data	Autor	Opis zmian	Wersja
23.11.2017	Bartłomiej Sielicki	Rozdział 1	0.1
	Łukasz Skarżyński	Rozdział 2 - moduł zbierania danych	

# Bibliografia

Flask, Framework wykorzystywany do tworzenia stron internetowych oraz api. Dostępne pod adresem: <http://flask.pocoo.org/>.

MongoDB, Nierelacyjna baza danych, przechowująca rekordy w formacie bson. Dostępne pod adresem: <https://www.mongodb.com/>.

Scrapy, Zestaw narzędzi umożliwiający szybką, wydajną ekstrakcję informacji ze stron internetowych. Dostępne pod adresem: <https://scrapy.org>.