Politechnika Warszawska

WYDZIAŁ MATEMATYKI
I NAUK INFORMACYJNYCH

Praca dyplomowa inżynierska

na kierunku Informatyka

Automatyczna analiza rynku pracy

Bartłomiej Sielicki

Numer albumu 268807

Łukasz Skarżyński

Numer albumu 268808

promotor

mgr Barbara Rychalska

konsultacje

dr hab. inż. Przemysław Biecek

WARSZAWA 2018

....

podpis promotora

....

podpis autora

Streszczenie

Automatyczna analiza rynku pracy

Rynek pracy w branży IT jest bardzo dynamiczny. Szczególnie jest to widoczne w ostatnich latach, kiedy to nieustannie zauważamy wzajemne wypieranie się technologii. Na zmiany reagują wszyscy - począwszy od firm, które- aby dorównać konkurencji- stale wdrażają nowe rozwiązania, przez pracowników chcących nieustannie poszerzać własne kompetencje, po uczelnie wyższe, które starają się przekazać swoim przyszłym absolwentom jak najbardziej aktualną wiedzę i umiejętności. Istotna staje się zatem możliwość wglądu w trendy i oparta na niej analiza zachodzących zmian.

Aplikacja, której poświęcono tę pracę, jest próbą udowodnienia, że proces taki da się zautomatyzować i- bez wymogu nadmiernej ingerencji użytkownika- zbierać, analizować, wizualizować oraz udostępniać najbardziej istotne (z punktu widzenia autorów) informacje. W ramach pracy wykonano system informatyczny składający się z dwóch komponentów. Pierwszym z nich jest zaplecze analityczne (back end), zajmujące się automatycznym pobieraniem zamieszczanych na jednym z największych polskich serwisów rekrutacyjnych (Pracuj.pl) ofert z branży IT oraz ich późniejszą analizą pod kątem umieszczanych w nich technologii. Drugim wykonanym komponentem jest aplikacja WWW (front end) prezentująca wyniki użytkownikowi oraz umożliwiająca eksport zebranych danych do samodzielnej analizy.

Słowa kluczowe: *Rynek pracy IT*, *Automatyzacja*, *Web Scraping*

Abstract

Automatic job market analysis

The job market in the IT industry is very dynamic. It has been especially visible in the recent years, when we have constantly noticed the mutual displacement of technologies. Everyone reacts to those changes – starting from companies, which are constantly implementing new solutions to match the competition, through the employees, who want to expand their own competences, and ending up with universities, who are trying to give their future graduates the most up-to-date knowledge and skills. Therefore, the possibility of having an insight into trends and basing on it an analysis of occurring changes become very important.

The application, on which this thesis is concentrated, is an attempt to prove that such a process can be automated and that it is possible - without the need for an excessive user interference – to collect, analyse, visualise, and share the most important (from the point of view of the authors) information. As part of the thesis, an IT system consisting of two components was created. The first of them is an analytical back-end, automatically downloading the IT industry job offers posted on one of the largest Polish recruitment websites (Pracuj.pl) and their subsequent analysis in terms of technologies placed in them. The second component is a web application (front end) presenting the results to the user, and enabling the export of the collected data for an independent analysis.

Keywords: IT job market, Automation, Web Scraping

Warszawa, dnia ……

Oświadczam, że moją część pracy inżynierskiej (zgodnie z podziałem zadań opisanym w treści pracy) pod tytułem „Automatyczna analiza rynku pracy”, której promotorem jest mgr Barbara Rychalska wykonałem samodzielnie, co poświadczam własnoręcznym podpisem.

.....

Spis treści

Wstęp

W ostatnich latach rozwój na wielu płaszczyznach technologii informacyjnych, zaowocował powstaniem nowych języków programowania (Kotlin, Swift), frameworków (React, Vue), narzędzi (PyTorch, Tensorflow) czy standardów. Często zauważyć można wzajemne wypieranie się wiodących technologii i narzędzi czy gwałtowne wzrosty popularności nowych, dotąd jeszcze szerzej nieznanych. Dobrymi przykładami są tutaj branże takie jak programowanie aplikacji WWW, Data Science czy świat aplikacji mobilnych. Ciekawym wydaje się być wpływ takich zmian na rynek pracy oraz to, czy są one tam zauważalne; jeśli tak - to w jakim stopniu.

W obserwacji takiego wpływu mógłby pomóc proces długotrwałej analizy pojawiających się na rynku ofert pracy - szczególnie pod kątem technologii, których znajomość jest wymagana przez pracodawców.

Aplikacja, której poświęcono tę pracę, jest próbą udowodnienia, że proces taki da się zaprojektować, zautomatyzować i - bez wymogu nadmiernej ingerencji użytkownika - zbierać, analizować, wizualizować oraz udostępniać najbardziej istotne (z punktu widzenia autorów) informacje dotyczące zmian na rynku pracy w branży IT.

W ramach pracy wykonano system informatyczny składający się z dwóch komponentów. Pierwszym z nich jest zaplecze analityczne (back end), zajmujące się automatycznym pobieraniem zamieszczanych na jednym z największych polskich serwisów rekrutacyjnych (Pracuj.pl) ofert z branży IT oraz ich późniejszą analizą pod kątem umieszczanych w nich technologii. Drugim wykonanym komponentem jest aplikacja WWW (front end) prezentująca wyniki użytkownikowi oraz umożliwiająca eksport zebranych danych do samodzielnej analizy.

Rozdział pierwszy pracy zawiera ogólną specyfikację wykonanego systemu - opis biznesowy, spis wymagań czy przewidziane grupy odbiorców.

W rozdziale drugim znajduje się opis każdego z wyodrębnionych komponentów systemu. Opisuje on ich architekturę, zasadę działania oraz efekt końcowy, ale też napotkane problemy, zastosowane algorytmy czy użyte narzędzia.

Rozdział trzeci zawiera instrukcję obsługi aplikacji WWW będącej efektem końcowym prac nad systemem.

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 pracy terminem tym określone jest przede wszystkim moduł analityczny z którym komunikuje się aplikacja WWW.
- **front end** - warstwa wizualna danego systemu - interfejs użytkownika. Pełni również rolę pośrednika między użytkownikiem a zapleczem systemu (back endem). W systemie którego dotyczy dana dokumentacja front end jest stroną internetową.
- **scraper** - program, którego głównym zadaniem jest zbierać określone dane ze stron internetowych.
- **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. 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.

Rozdział 1

Specyfikacja projektu

1.1. Opis biznesowy

Głównym zadaniem aplikacji jest automatyczna analiza rynku pracy. System zajmuje się agregacją oraz przetwarzaniem 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 biorąc pod uwagę jedynie kluczowe elementy, takie jak: tytuł ogłoszenia, opis, datę dodania, itp.
2. Zapisania tak zebranych ogłoszeń do bazy danych
3. Przeprowadzenia analizy na treści ogłoszenia uzyskując listę technologii wymienionych w treści ogłoszenia, których znajomość jest oczekiwana od pracownika.
4. Udostępnienia użytkownikowi interfejsu do wyszukiwania zebranych w systemie ofert oraz wyświetlania statystyk dla wybranych technologii w postaci strony WWW.

Wymienione działania realizują odrębne komponenty systemu.

Użytkownik bezpośrednio korzystać powinien tylko z jednego modułu - wspomnianej wyżej aplikacji WWW.

Przewidzianymi grupami docelowymi użytkowników aplikacji są:

- studenci i osoby szukające pracy - dzięki oferowanym przez aplikację danym, będą mogli ocenić znajomość których technologii staje się pożądana na rynku pracy
- pracodawcy - na podstawie trendów wśród używanych technologii będą mogli podjąć decyzje dotyczące przyszłych projektów
- pozostali użytkownicy zainteresowani zmianami na rynku pracy - możliwość eksportu zgromadzonych przez system danych pozwala na przeprowadzenie samodzielnej analizy

1.2. Wymagania funkcjonalne

Podstawą interakcji użytkownika z systemem jest strona WWW - użytkownik powinien być w stanie otworzyć ją na dowolnym komputerze z dostępem do internetu, wyposażonym w przeglądarkę:

- 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 nie da się zagwarantować że będzie to działanie w pełni poprawne.

Na stronie nie przewidziano kont użytkowników, nawet administracyjnego. Każdy z odwiedzających ma dostęp do tych samych danych oraz takie same możliwości.

Funkcjonalność aplikacji WWW rozbita jest na trzy podstrony:

- statystyki technologii
- wyszukiwanie ofert
- informacje o systemie

Możliwości użytkownika w obrębie każdej z sekcji prezentuje załączony diagram przypadków użycia (Rys. 1.1).

1.2.1. Wyświetlenie statystyk technologii

Jest to pierwsza podstawowa funkcjonalność strony. Pozwala ona użytkownikowi na wybór jednej bądź kilku (przy pomocy auto-uzupełniania) technologii oraz wyświetlenie:

- w przypadku wybrania więcej niż jednej technologii, wykresu porównującego z osobna ich popularność (pod względem ilości ofert)
- wykresu ilości ofert zawierających każdą z wybranych technologii
- wykresu procentowej ilości ofert z systemu zawierających te technologie
- wykresu kołowego przedstawiającego najczęściej poszukujących wybranych technologii pracodawców
- listy dziesięciu najbardziej zbliżonych technologii

1.2.2. Wyszukiwanie ofert wg technologii

Kolejną oferowaną użytkownikom możliwością jest wyszukiwanie ofert zebranych i umieszczonych w bazie. Rolę kryterium wyszukiwania, podobnie jak podczas przeglądania statystyk, pełni jedna lub więcej wybranych technologii. Rezultatem jest lista ogłoszeń oraz przycisk oferujący możliwość ich eksportu w formacie JSON.

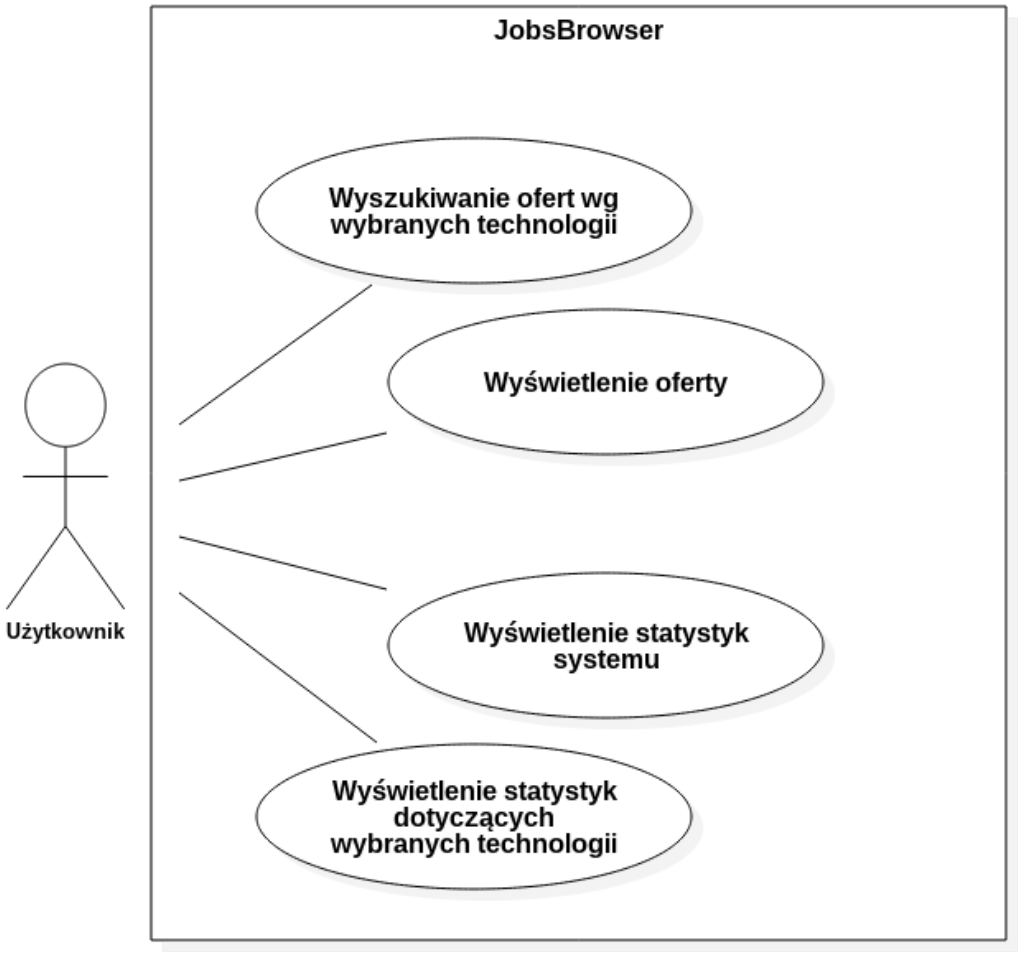
Każdą z ofert widocznych na liście można rozwinąć, uzyskując dostęp do następujących informacji:

- tytuł oferty
- data dodania i wygaśnięcia oferty w macierzystym serwisie
- nazwa pracodawcy i miejsce pracy
- wszystkie wykryte w treści oferty technologie
- odnośnik do oryginalnego ogłoszenia w macierzystym serwisie

1.2.3. Wyświetlenie informacji o systemie

W osobnej sekcji użytkownik ma dostęp do wyświetlenia zbiorczych statystyk dotyczących serwisu i dostępnych w nim danych. Zbiór dostępnych statystyk sprowadza się do:

- liczby wszystkich ofert w bazie systemu
- wykresu powyższej wartości względem czasu
- daty ostatniego zebrania danych z serwisu macierzystego



Rysunek 1.1: Przypadki użycia.

1.3. Wymaganie niefunkcjonalne

Tabela 1.1 przedstawia wymagania niefunkcjonalne postawione stworzonej 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 jest 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.
Wydajność (Performance)	Aplikacja WWW powinna działać płynnie na każdym komputerze z dowolnym systemem operacyjnym wyposażonym w odpowiednią przeglądarkę.
Wsparcie	Dane z których korzysta serwis aktualizowane są na bieżąco. W stopce strony umieszczone są adresy e-mailowe autorów aplikacji, oferujących pomoc w przyadku problemów technicznych.

Rozdział 2

Architektura systemu

2.1. Schemat

Przewidziana architektura ma strukturę modułową. Ogólny schemat komponentów oraz ich połączenie przedstawia załączony diagram (Rys. 2.1), natomiast bardziej szczegółowy opis każdego z nich znajduje się w dalszej części tego rozdziału.

W systemie wyróżnić możemy trzy główne, niezależne od siebie (na tyle że mogą, a nawet powinny, być uruchamiane na różnych maszynach) moduły.

Robot zbierający dane

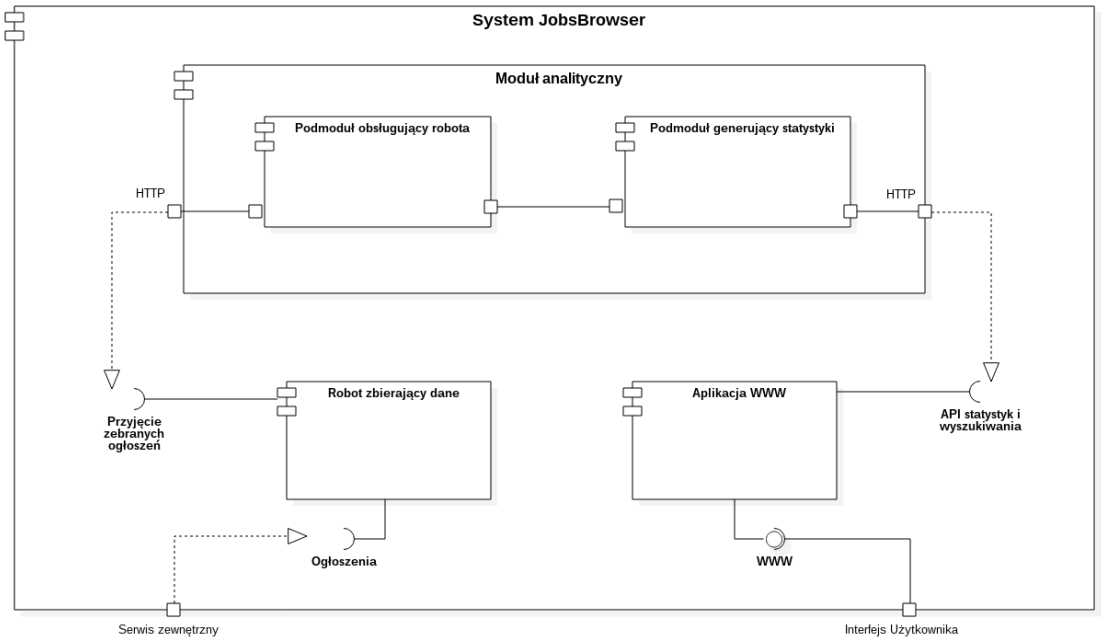
Komponent ten, nazywany też *scraperem*, odpowiada za automatyczne pobieranie ogłoszeń. Jest to program, który cyklicznie łączy się z udostępniającym ogłoszenia serwisem i automatycznie pobiera ich treść. Zebrane oferty przesyła do kolejnego komponentu systemu.

Moduł analityczny

Stosowaną zamiennie nazwą jest *Pipeline* - czyli *łańcuch przetwarzania*. Odbiera on zebrane ogłoszenia, a następnie wykonuje na nich sekwencję operacji obejmujących zapis do bazy danych czy wykrycie zawartych w treści technologii. Komponent ten jest również odpowiedzialny za komunikację z aplikacją WWW, generując wyświetlane przez nią dane.

Aplikacja WWW

Ostatnim elementem systemu, jedynym dostępnym bezpośrednio dla użytkownika jest aplikacja WWW. Nie posiada ona własnej bazy, a co za idzie kont użytkowników. Komunikuje się z modułem analitycznym, pozwalając użytkownikowi na przejrzysty i wygodny dostęp do informacji.



Rysunek 2.1: Diagram komponentów.

2.2. Robot zbierający dane

Kod źródłowy znajduje się pod adresem:

- github.com/jobsbrowser/scraper.

Komponent zajmuje się automatycznym pobieraniem ofert z serwisu zewnętrznego, ekstrakcją podstawowych informacji oraz przesłaniem tak przetworzonych ofert do kolejnego modułu. Wybór padł na serwis Pracuj.pl jako źródło ofert oraz na framework **Scrapy**[1, Scrapy] jako bazę kodu robota.

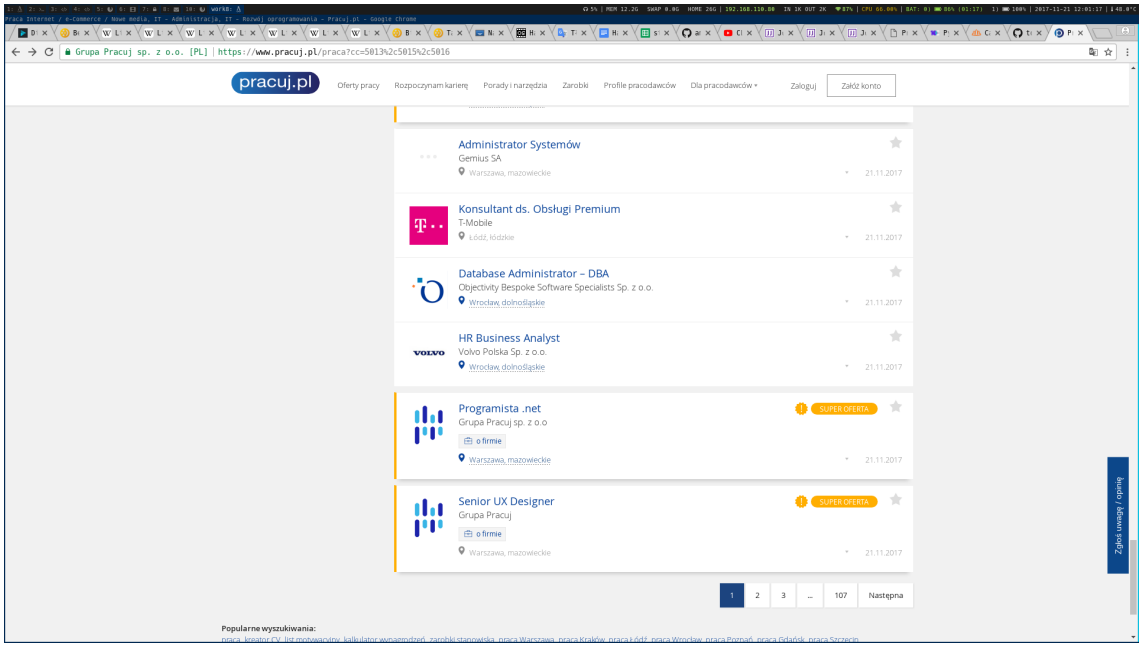
2.2.1. Serwis zewnętrzny

Serwis Pracuj.pl dzieli zamieszczone w nim oferty na kategorie. Z racji tematyki pracy brane pod uwagę będą wyłącznie trzy z nich. Są to (wraz z podziałem na podkategorie):

- Internet / e-Commerce / Nowe media
  - E-marketing / SEM / SEO
  - Media społecznościowe
  - Projektowanie
  - Sprzedaż / e-Commerce
  - Tworzenie stron WWW / Technologie internetowe
- IT - Administracja
  - Administrowanie bazami danych i storage
  - Administrowanie sieciami
  - Administrowanie systemami
  - Bezpieczeństwo / Audyt
  - Wdrożenia ERP
  - Wsparcie techniczne / Helpdesk
  - Zarządzanie usługami
- IT - Rozwój oprogramowania
  - Analiza biznesowa
  - Architektura

- Programowanie
- Testowanie
- Zarządzanie projektem

Widok listy ogłoszeń w serwisie umożliwia wybór kategorii, z których ogłoszenia mają zostać wyświetlone. Możliwość ta pozwola na uzyskanie bazowego linku od którego zacznie się pobieranie ofert.



Rysunek 2.2: Widok paginacji ogłoszeń w witrynie pracuj.pl.

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.

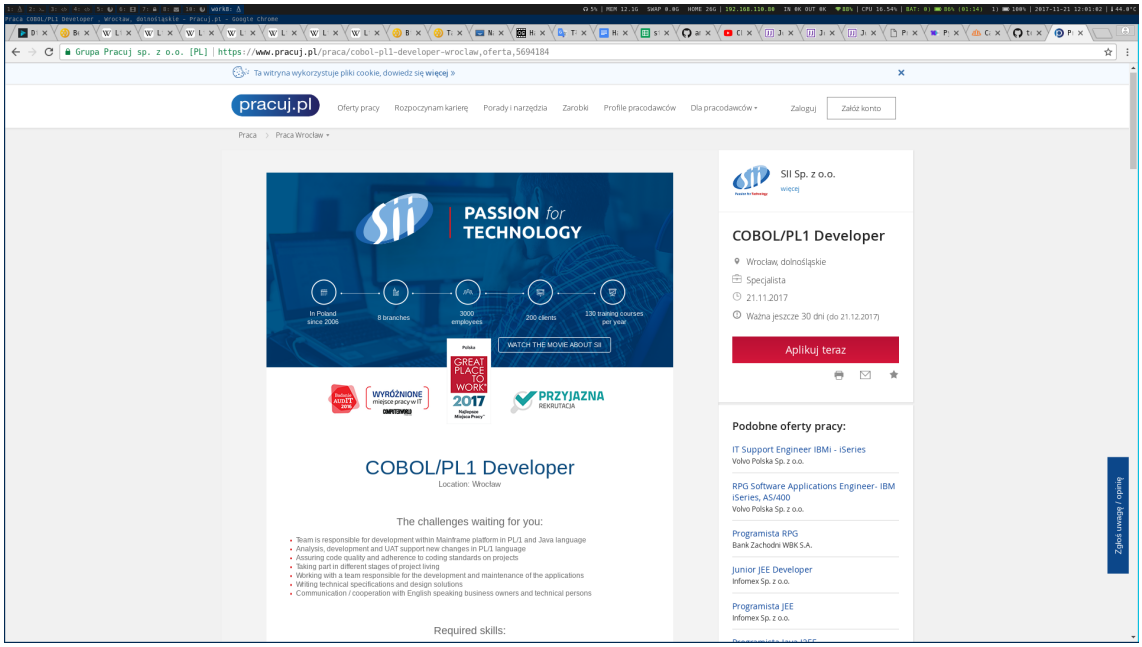
Do każdej z ofert na stronie (których znajduje się ok. 50) prowadzi bezpośredni link, który można wydobyć z kodu HTML listy. Podstrona pojedynczej oferty zawiera wszystkie interesujące informacje, również możliwe do uzyskania z kodu HTML przy użyciu odpowiednich selektorów CSS. Dostępne w przystępny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

- Data dodania oferty
- Data wygaśnięcia
- Nazwa dodającego (pracodawca)
- Lokalizacja (miasto i województwo)
- Tytuł oferty
- Treść oferty
- Kategorie w których znajduje się dana oferta

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 2.3: Widok oferty w witrynie pracuj.pl.

2.2.2. Napotkane problemy

Zadaniem stojącym przed scraperem jest automatyczne przejście po wszystkich dostępnych stronach listy, wydobycie z nich 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 wynikło kilka kwestii które wymagały opracowania konkretnego rozwiązania.

Aktualizacja ofert w serwisie

Ofert na stronie wraz z upływem czasu będzie przybywać - dla działającego 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 są 24 godziny, choć nie jest to wartość na stałe zdefiniowana w programie. To od uruchamiającego system zależy jak ją ustawi.

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. Rozwiązaniem okazało się zaimplementowanie w scraperze możliwości pobrania z modułu do którego trafią 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.

Struktura danych

Początkowo dane pobierane były z wersji portalu Pracuj.pl przeznaczonej dla urządzeń o wyższej rozdzielczości ekranu (Laptopy, komputery PC). Problemem okazała się jednak niejednolita struktura danych, utrudniająca automatyczne wydobywanie informacji z treści ogłoszenia. Rozwiązaniem okazało się korzystanie z wersji portalu przeznaczonej dla urządzeń mobilnych.

2.2.3. Wykorzystane technologie

Scrapy

Licencja: BSD3

Scraper powstał przy użyciu frameworka Scrapy[1, Scrapy]. Jest to framework napisany w języku Python, przy wykorzystaniu biblioteki Twisted. Dzięki temu działa całkowicie asynchronicznie, co czyni go wydajnym przy relatywnej łatwości pisania własnych scraperów.

2.3. Moduł analityczny

Kod źródłowy znajduje się w repozytoriach:

- github.com/jobsbrowser/pipeline
- github.com/jobsbrowser/backend

Modułem systemu do którego trafiają przetwarzane oferty w następnej kolejności jest moduł analityczny. To tutaj odbywa się, kluczowy z punktu widzenia biznesowego zastosowania projektu, proces ekstrakcji ze zbieranych ofert wartościowych informacji. Wejściem modułu są pobierane z serwisu Pracuj.pl oferty, natomiast rolę wyjścia pełni interfejs HTTP z którego korzysta aplikacja WWW.

Moduł ten rozdzielony jest na dwa komponenty - pierwszym z nich jest usługa obsługująca robota zbierającego dane. Przyjmuje ona pobrane oferty, roboczo zapisuje je w nierelacyjnej bazie danych, a następnie dla każdej z nowo dodanych rozpoczyna sekwencyjny proces analizy. Przetworzone ogłoszenia trafiają w ostatnim kroku tego procesu do drugiego komponentu, którym jest usługa obsługująca aplikację WWW. W odrębnej, już relacyjnej, bazie danych zapisuje ona najważniejsze informacje o przetworzonych ofertach oraz, co najważniejsze, listę technologii które udało się w nich wykryć.

2.3.1. Przeznaczenie modułu

Dzięki robotowi zbierającemu dane do systemu trafiają pojawiające się w serwisie Pracuj.pl oferty. Dla przypomnienia, informacje jakie uzyskiwane są bezpośrednio w trakcie ich pozyskiwania to:

- tytuł
- ID
- treść (w postaci kodu HTML)

oraz kilka mniej znaczących z punktu widzenia tego komponentu, a szerzej opisanych w części dotyczącej robota zbierającego dane. Informacje te same w sobie nie są szczególnie istotne dla użytkownika końcowego aplikacji. Są to bowiem te same dane które zobaczyć możemy korzystając bezpośrednio z serwisu pracuj.pl. Nie niosą więc za sobą póki co żadnych dodatkowych wartości.

Sednem aplikacji i systemu samego w sobie, jest przekształcenie dużego zbioru ofert w statystyki dotyczące ich wspólnych elementów - czyli technologii które są w nich wymieniane. Moduł ten pozwala właśnie na to - uzyskanie z pojedynczej oferty listy nazw technologii, które są zawarte w jej treści.

2.3.2. Obsługa robota zbierającego dane

Komunikacja między modułem analitycznym a robotem zbierającym dane odbywa się za pośrednictwem protokołu HTTP. Interfejs udostępniany robotowi pozwala na następujące operacje:

- Dodanie oferty do bazy (nadpisując jeśli oferta z takim adresem URL już istnieje)
- Pobranie listy adresów URL ofert zapisanych już w bazie, tak aby uniknąć przetwarzania ich ponownie
- Uaktualnienie wybranej oferty z bazy danych

Wymagania niefunkcjonalne stawiane tej części systemu sprowadzają się zatem 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 robot zbierający dane nie uzyska odpowiedzi od modułu zajmującego się ich przechowywaniem i analizą, informacje o tej ofercie nie zostaną nigdzie zapisane. Kiedy usługa przechowywania będzie ponownie dostępna, na zapytanie robota 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.

Baza danych

Wykorzystanym silnikiem bazy danych do roboczego zapisu ofert jest MongoDB[2, MongoDB]. Wybór padł na bazę nierelacyjną, ponieważ jedynym typem trzymanych w niej

danych są same oferty. Oznacza to że w bazie relacyjnej znajdowałaby się tylko jedną tabela - bez żadnych relacji czy potrzeby zachowania spójności. Nie ma potrzeby również stosowania mechanizmu transakcji, czy skomplikowanych zapytań. Tym co faktycznie jest oczekiwane od bazy jest wydajność, dostępność oraz ewentualna skalowalność.

Struktura dokumentów przechowywanych w bazie jest identyczna jak struktura zebranego ogłoszenia. Dla przypomnienia, pola wyróżniane 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
- Kategorie oferty
- Kod HTML treści oferty (rozbity na opis, kwalifikacje oraz benefity - wg struktury Pracuj.pl)

Interfejs HTTP

Interfejs HTTP zaimplementowany jest przy użyciu mikro-frameworka Flask[3, Flask]. Przydatne okazało się dostępne rozszerzenie integrujące go z bazą MongoDB, użytą w module.

2.3.3. Ekstrakcja technologii

Uznaliśmy, że aby w miarę poprawnie rozpoznawać technologie, powinniśmy w pierwszym kroku zaopatrzyć się w miarę obszerny i sprawdzony ich zbiór.
Podejście takie, jak później się okazało wydaje się być całkiem słuszne, ponieważ stosowane jest także w projektach o znacznie szerszym zasięgu i złożoności niż nasz. Dla przykładu, powstająca w momencie pisania tej pracy platforma **Google Cloud Job Discovery**[4, Discovery, G.C.J.], zajmująca się automatycznym dopasowywaniem ofert pracy do CV potencjalnych pracowników, do działania wykorzystuje zbudowaną przez zespół Google ontologię zawierającą, jak podają, ok. 50 tys. umiejętności z różnych pól zawodowych.

Z racji tego, że nasz projekt skupia się na ofertach pracy z branży IT, postanowiliśmy skorzystać z faktu że istotne, oczekiwane przez pracodawców umiejętności, pokrywają się z nazwami technologii informatycznych, czy języków programowania. Jako źródło takich danych, postanowiliśmy wykorzystać popularny wśród ludzi zainteresowanych IT portal **Stack Overflow**[5, StackOverflow]. API tego serwisu pozwala na pobranie używanych przez jego użytkowników tagów, posortowanych według popularności. Wszystkich jest blisko 40 tys. W zdecydowanej większości odpowiadają one nazwom technologii.

Na potrzeby naszego projektu, korzystamy ze zbioru dwóch tysięcy najpopularniejszych tagów. Znajdują się wśród nich wszelakie technologie, frameworki, wzorce projektowe i języki programowania. Mając taką listę, możemy każdą przychodzącą ofertę przeszukać pod względem występowania niektórych z nich. Proces ten przeprowadzamy w następując sposób:

1. Po zapisaniu całej oferty do bazy, upraszczamy obiekt przekazując dalej tylko istotne z punktu widzenia komponentu dane, tj.

- ID
- tytuł
- treść

2. Z treści oferty usuwamy tagi HTML
3. Treść oferty rozbijamy na *tokeny*, czyli wyrazy oraz znaki interpunkcyjne
4. Sprawdzamy czy oferta jest w języku polskim, bo tylko takie akceptujemy
5. Usuwamy zbędne z punktu widzenia analizy językowej tokeny, takie jak przyimki, spójniki czy zaimki.
6. Dla listy uzyskanych tokenów sprawdzamy które z nich znajdują się na liście znanych nam technologii, i te zapisujemy.

W ten sposób każdej ofercie z osobna przypisujemy listę technologii które znaleźliśmy w jej treści.

Architektura

Z technicznego punktu widzenia proces ten jest ciągiem funkcji, wykonywanych w ustalonej kolejności, gdzie wyjście każdej z nich przekazywane jest jako argument do następnej.

Stąd powtarzający się często termin *pipeline*. Argumentem pierwszej funkcji jest zapisana do bazy danych chwilę po otrzymaniu od robota oferta, natomiast wynikiem ostatniej uproszczony obiekt oferty zawierający listę znalezionych technologii. Na koniec oferta jest wyszukiwana w bazie, dodawana jest do niej wspomniana lista (dzięki zastosowaniu nierelacyjnej bazy danych nie wiąże się to z potrzebą jej przebudowy), a następnie wysyłana jest do usługi generującej statystyki.

Do obsługi łańcucha, czyli utrzymania poprawnej kolejności wykonywania funkcji, oraz asynchronicznego przetwarzania wielu ofert wykorzystywany jest framework **Celery**[6, Celery].

2.3.4. Próby analizy tekstu i uzasadnienie wyboru algorytmu

Dużo czasu poświęciliśmy na testowanie wielu algorytmów do wyznaczania słów kluczowych z tekstu. Wiele z nich dawało niezadowalające wyniki na zbiorach ogłoszeń napisanych w języku polskim. Poniżej opisujemy krótko algorytmy które przetestowaliśmy wraz z przykładowymi wynikami jakie dawały. Wszystkie przedstawione poniżej wyniki zostały uzyskane z przykładowej oferty (Rys. 2.4).

Wystarczy, że:

- posiadasz doświadczenie w podobnej roli
- znasz na poziomie przynajmniej dobrym język Python (idealnie Python 3) wraz z frameworkiem Django
- znasz również JavaScript (nie tylko jQuery)
- posiadasz dobrą wiedzę na temat relacyjnych baz danych (mile widziana znajomość PostgreSQL)
- sprawnie poruszasz się w HTML5 i CSS3
- cechuje Cię ponadprzeciętna samodzielność oraz umiejętność samoorganizacji
- potrafisz pisać wysokiej jakości kod
- posiadasz umiejętność dekompozycji zadań oraz dostarczania działających rozwiązań
- jesteś skrupulatny (lub przynajmniej pracujesz nad tym, aby takim się stać)

to z dużym prawdopodobieństwem jesteś osobą, której szukamy.

Jeżeli dodatkowo:

- potrafisz pisać testy jednostkowe
- znasz dowolny, inny niż wyżej wymienione, język programowania
- możesz wykazać się znajomością protokołów sieciowych

to z tym większą niecierpliwością oczekujemy na możliwość spotkania z Tobą.

Chcemy powierzyć Ci następujące obowiązki:

- projektowanie oraz implementowanie aplikacji internetowych, w tym wspierających płatności, uwierzytelnianie SMS, jak również aplikacji wyszukujących informacje na wielu stronach
- implementowanie nowych funkcji i poprawek w istniejących aplikacjach internetowych (utrzymanie i rozwój narzędzi stworzonych na potrzeby wewnętrzne)
- ścisłą współpracę z doświadczonymi architektami oprogramowania, wraz z wpływem na kształt oraz sposób działania tworzonych narzędzi

Dodatkowo oferujemy:

- kontakt z najnowszymi technologiami oraz niespotykaną różnorodność tematyczną projektów
- możliwość realizowania własnych, innowacyjnych projektów
- elastyczne godziny pracy
- współpracę z ekspertami z wieloletnim doświadczeniem, którzy chętnie dzielą się swoją wiedzą
- szkolenia wewnętrzne, udział w spotkaniach, seminariach oraz konferencjach
- bardzo dobre warunki pracy, nowoczesny sprzęt, świeże owoce i pyszną kawę
- dostęp do dodatkowych świadczeń (Multisport, Medicover)
- swobodną atmosferę, brak dres code'u, nieformalne relacje
- mnóstwo ciekawych wyzwań i szans rozwoju
- nowoczesne, doskonale skomunikowane biuro w centrum Warszawy

Rysunek 2.4: Oferta wykorzystywana do uzyskania przykładowego wyniku

RAKE(Rapid Automatic Keyword Extraction)[7, Rake-NLTK]

Popularny, stosunkowo prosty algorytm do automatycznej ekstrakcji słów kluczowych. Niestety nie przyniósł zadowalających efektów. Poniżej lista zawiera 10 najważniejszych według algorytmu RAKE fraz/kluczy wykrytych w przykładowej ofercie:

- umiejętność samoorganizacji potrafisz pisać wysokiej jakości kod posiadasz umiejętność dekompozycji zadań
- potrafisz pisać testy jednostkowe znasz dowolny
- dostarczania działających rozwiązań jesteś skrupulatny
- poziomie przynajmniej dobrym język python
- język programowania możesz wykazać
- dużym prawdopodobieństwem jesteś osobą
- temat relacyjnych baz danych
- mile widziana znajomość postgresql
- znajomością protokołów sieciowych to
- posiadasz dobrą wiedzę

Jak widać na powyższej liście algorytm RAKE zwrócił ciekawe rezultaty, jednak ich dokładność i forma pozostawiały wiele do życzenia.

TF-IDF(Term Frequency - Inverse Document Frequency)[8, TF-IDF]

Jeden z bardziej popularnych i szeroko wykorzystywanych algorytmów w świecie przetwarzania języka naturalnego.

Przy opisie algorytmu korzystamy z poniższych oznaczeń:

- D - zbiór wszystkich dokumentów (ogłoszeń w bazie)
- d_i - i-ty dokument (ogłoszenie) z całego korpusu
- t_i - i-te słowo w ogłoszeniu

TF-IDF jest algorytmem przypisującym słowu wagę obliczaną na podstawie istotności tego słowa w dokumencie. Do wyznaczenie jej potrzebne są następujące miary:

- Term Frequency - liczba wystąpień słowa t_i w dokumencie d_j, oznaczamy ją przez $tf(t_i, d_j)$
- Document Frequency - liczba dokumentów w których wystąpiło słowo t_i, oznaczenie: $df(t_i)$

- Inverse Document Frequency - odwrotna częstość dokumentów, obliczana za pomocą wzoru: $idf(t_i) = \log_2 \frac{|D|}{df(t_i)}$

Korzystając z powyżej zdefiniowanych miar, waga słowa wyznaczana przez algorytm TF-IDF jest postaci:

$tfidf(t_i, d_j) = tf(t_i, d_j) \cdot idf(t_i)$

Z powyższego wzoru widać, że waga jest tym większa im częściej w danym dokumencie występuje badane słowo (mnożenie przez $tf(t_i, d_j)$), ale wartość wagi zmniejsza się wraz z wzrostem częstości występowania badanego słowa we wszystkich dokumentach z korpusu ($idf(t_i)$), co przyczynia się do ignorowania słów z tzw. listy stopwords, czyli listy słów nieistotnych, ale często występujących w zdaniach.

Niestety algorytm ten nie dawał zadowalających wyników. Wykryte słowa kluczowe dla przykładowej oferty (Rys. 2.4):

- pisać
- implementowanie
- python
- posiadasz
- wyszukujących
- znasz
- uwierzytelnianie
- dres
- niecierpliwością
- niespotykaną

Z racji bardzo szczególnego podzbioru języka polskiego używanego przy pisaniu ogłoszeń oraz relatywnie niewielkiej ich ilości, powyższe algorytmy dawały niezadowalające wyniki, dlatego też zdecydowaliśmy się na korzystanie z algorytmu opartego na tagach pobranych z serwisu *Stack Overflow*.

2.3.5. Znajdowanie podobnych technologii za pomocą modelu Word2Vec

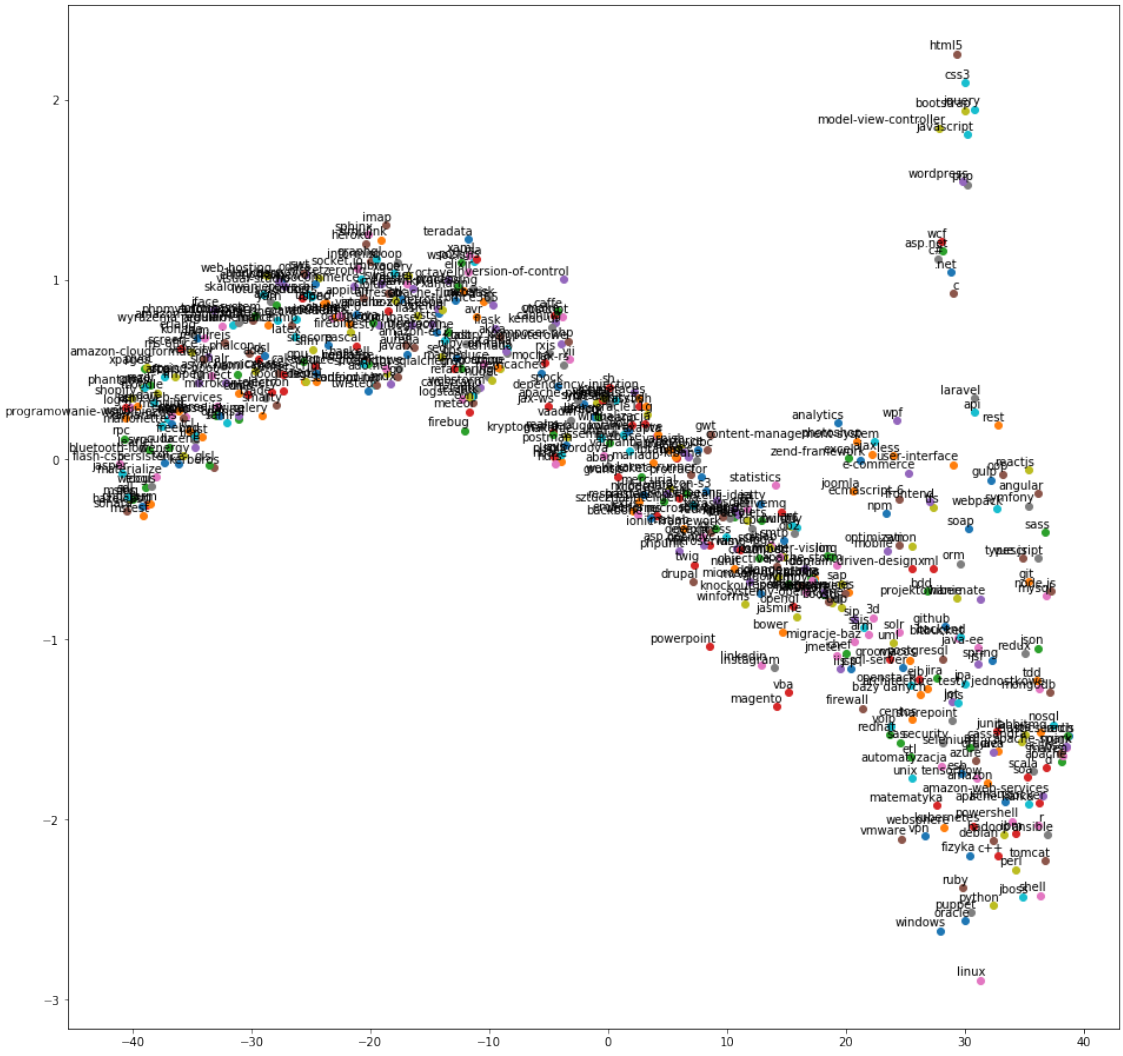
Jedną z funkcjonalności systemu jest generowanie technologii zbliżonych do tych wybranych przez użytkownika. W celu znalezienia takiego zbioru postanowiliśmy skorzystać z

modelu Word2Vec [9, Goldberg, Y. & Levy, O.]. Word2vec to grupa modeli reprezentujących słowa jako tzw. zanurzenia słów [10, Bengio, Y., Réjean, D. & Pascal, V.], czyli słowa reprezentowane jako wektory w wielowymiarowej przestrzeni.

W naszym przypadku zdecydowaliśmy się wytrenować model word2vec nie na całych korpusach (wszystkich ogłoszeniach), ale na wykorzystywanej przez nas liście technologii. Dzięki takiemu podejściu wynikowy zbiór zawierać będzie tylko pozostałe elementy z tej listy. Do wytrenowania modelu skorzystaliśmy z biblioteki gensim[11, Gensim].

Rysunek 2.5 przedstawia wizualizacje wektorów technologii uzyskaną przy pomocy algorytmu TSNE[12, Maaten, L. van der & Hinton, G.], który transformuje zbiór wielowymiarowych wektorów (w przypadku naszego modelu 100-wymiarowych), kompresując je do 2-wymiarowej przestrzeni, zachowując proporcje, tj. wektory które znajdowały się blisko siebie w wielowymiarowej przestrzeni powinny znajdować się również blisko siebie w 2 wymiarowej przestrzeni oraz analogicznie dla wektorów znajdujących się daleko od siebie.

Na rysunku możemy zaobserwować, że technologie podobne do siebie, bądź często łączone ze sobą, występują blisko siebie, tak jak np. `html5`, `css3`, `jquery` i `bootstrap`, czyli technologie front endowe, bardzo często wykorzystywane razem w wielu projektach.



Rysunek 2.5: Wizualizacja wektorów technologii

Aby określić poprawność znajdowanych przez algorytm technologii, wybraliśmy kilka technologii dla których wskazaliśmy dziesięć ich przewidywanych najbliższych sąsiadów (technologie subiektywnie najbardziej podobne do nich). Przykładowo dla technologii `JAVA` jako jej najbliższych sąsiadów wskazaliśmy:

- maven
- hibernate
- junit
- spring
- groovy
- vaadin
- tomcat
- jsp
- jsf
- jenkins

a dla `Vue.js`:

- javascript
- node.js
- mvvm
- frontend
- reactjs
- angular
- api
- webpack
- firebase
- user-interface

Poniżej znajdują się wyniki otrzymane dla technologii `JAVA`:

- java-ee
- maven
- junit
- hibernate
- groovy

- soa
- tomcat
- jsp
- jenkins
- eclipse

Zatem wyżej opisany algorytm poprawnie wskazał 7 z 10 wybranych przez nas technologii oraz jak widać wyniki są zadowalające, wszystkie pozostałe technologie (eclipse, java-ee oraz soa) mają wiele wspólnego z językiem programowania JAVA.

Niestety z powodu stosunkowo małego zbioru treningowego rezultaty nie są tak dobre dla mniej popularnych technologii, mających mało wystąpień. Dla przykładu - `Vue.js`:

- node.js
- elasticsearch
- jasmine
- testy jednostkowe
- bitbucket
- api
- nosql
- webpack
- mongodb
- user-interface

Wynik ten zawiera 4 z 10 wybranych przez nas technologii.

2.3.6. Generowanie statystyk

Przeznaczeniem tej części modułu jest odpowiadanie na żądania wyszukiwania oraz generowania statystyk. Komunikacja taka powinna odbywać się z wykorzystaniem protokołu HTTP, tak aby łatwo można było zintegrować moduł z aplikacją WWW bądź w przypadku takiej potrzeby innym dowolnym konsumentem.

Z powodu złożoności zapytań (jak np. generowanie statystyk sumarycznych dla wielu dni i wielu technologii jednocześnie) część ta potrzebuje własnej bazy danych na której wykonywane będą zapytania. Wymusza to dodatkowe wymaganie w postaci możliwości dodawania

nowych ofert, z którego korzysta proces ekstrakcji technologii (wysyłając na bieżąco w pełni przetworzone oferty).

Wymagania

Wymagania funkcjonalne sprowadzają się więc do obsługi następujących żądań HTTP:

- dodanie nowej oferty do bazy
- wyszukiwanie ofert przy pomocy danego zbioru technologii
- generowanie statystyk, tj. dla każdego dnia z zakresu od 02.12.2018 do daty wykonania żądania obliczenie ilości aktywnych wówczas w serwisie pracuj.pl ofert zawierających podane technologie
- generowanie statystyk całościowych - tj. ilości wszystkich ofert w bazie (również we wspomnianym wyżej zakresie) oraz daty dodania ostatniego ogłoszenia

Natomiast wymagania niefunkcjonalne postawione modułowi to:

- **niezawodność** - usługa powinna być dostępna cały czas, ponieważ dostarcza ona danych niezbędnych do pokazania wszystkich statystyk w aplikacji z której korzysta użytkownik. Bez niej aplikacja WWW staje się bezużyteczna.
- **wydajność** - liczba przychodzących do modułu danych może być momentami bardzo duża, dlatego zapisywanie do bazy nie może być przeprowadzane niewydajnie. Największy wpływ na wydajność mają zapytania generujące odpowiednie statystyki. Niektóre z nich są skomplikowane, dlatego niezbędna jest ich optymalizacja. Wydłużony czas generowania statystyk obniży znacznie komfort używania aplikacji WWW

Interfejs

Usługa z pozostałymi komponentami systemu łączy się przez interfejs HTTP. Dane do statystyk są dostarczane przez proces ekstrakcji technologii. Konsumentem generowanych przez usługę statystyk jest aplikacja WWW z której korzysta końcowy użytkownik.

Interfejs zaimplementowany jest przy użyciu frameworka Django[13, Django]. Implementując ten interfejs wdrażaliśmy dobre praktyki programowania opisane w książce “Two Scoops of Django 1.11: Best Practices for the Django Web Framework”[14, Greenfeld, D.R. & Greenfeld, A.R. 2017].

Baza danych

Wykorzystanym silnikiem bazy danych jest SQLite3[15, SQLite]. Wybór padł na bazę relacyjną ze względu na to, że wykonywane są do niej skomplikowane zapytania, które silnikom nierelacyjnych baz danych zajmują więcej czasu oraz zasobów serwera, o czym przekonaliśmy się, testując te same zapytania na bazie danych MongoDB[2, MongoDB]. Struktura dokumentów przechowywanych w bazie jest relacyjnym odzwierciedleniem struktury ogłoszenia zebranego przez robota zbierający dane. Baza zawiera trzy tabele:

- Oferty
  - 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
  - Kategorie oferty
  - Kod HTML treści oferty (rozbity na opis, kwalifikacje oraz benefity - wg struktury Pracuj.pl)
- Tagi
  - nazwa technologii
- Dodatkowa tabela reprezentująca relację ofert z tagami - jest to bowiem relacja `many-to-many` - czyli wiele do wielu.

W celu optymalizacji często wykonywanych zapytań baza posiada założone indeksy. W tabeli tagów indeksowaną kolumną jest nazwa technologii, zaś w bazie ofert data dodania oferty oraz data wygaśnięcia oferty.

Algorytm generowania statystyk

O ile wyszukiwanie ofert na podstawie listy podanych technologii realizowane jest w wydajny i optymalny sposób przez użyty framework, o tyle generowanie statystyk w określony

przez wymagania modułu sposób jest bardziej skomplikowane i wymagało implementacji własnego algorytmu.

Przypominając wymagania, interfejs statystyk ma działać w następujący sposób:

- Przyjmij listę technologii
- Dla każdego dnia z przedziału od 02.12.2018 (początek zbierania ofert) do dzisiaj policz ile tego dnia było w serwisie pracuj.pl aktywnych ofert zawierających te technologie oraz jaki procent wszystkich aktywnych ofert z branży IT stanowiły.
- Zwróć wynik

Ofertę uważa się za aktywną danego dnia, jeśli zachodzą dwa warunki: `dzień dodania <= wybrany dzień` oraz `dzień wygaśnięcia >= wybrany dzień`. Mając te dwie daty w bazie danych można więc konstruować zapytania wybierające odpowiednie oferty.

Naiwna implementacja algorytmu, sprowadzałaby się do wykonania n zapytań zliczających do bazy danych, gdzie n to ilość dni które upłynęły od daty początkowej. Nie jest to rozwiązanie optymalne, ponieważ zapytania do bazy danych są kosztowne a z każdym kolejnym dniem wygenerowanie statystyk wymagałoby ich więcej.

Postanowiliśmy ograniczyć się do jednego zapytania, a algorytm wygląda następująco:

1. Wybierz oferty aktywne przez choć jeden dzień na zadanym przedziale. Wykorzystane warunki to: `dzień dodania <= koniec przedziału`, `dzień wygaśnięcia >= początek przedziału` oraz warunek zawierania się wszystkich podanych technologii wśród tych wykrytych w ofercie.
2. Utwórz n "kubełków", gdzie każdy kubełek odpowiada jednej dacie z przedziału i na początku ma wartość 0
3. Przejdź po liście ofert i dla każdej z nich:
    - Zwiększ wartość kubełka odpowiadającego dacie dodania oferty o 1. Może się zdarzyć że data dodania oferty poprzedza początek zakresu, i nie ma takiego kubełka. Wtedy zwiększ wartość pierwszego kubełka.
    - Jeśli data wygaśnięcia oferty poprzedza koniec zakresu, to zmniejsz wartość w kubełku z dniem następnym po dniu wygaśnięcia.
4. Wykonaj operację sumy skumulowanej na kubełkach.

Dzięki temu w każdym kubełku odpowiadającym każdemu dniu z zakresu znajdzie się ilość pasujących do zapytania, aktywnych tamtego dnia ofert. Do uzyskania wyniku procentowego operacja jest powtarzana, ale w kroku 1 pomijany jest warunek dotyczący technologii. Wtedy wynikiem jest iloraz rezultatów tych dwóch operacji.

2.3.7. Wykorzystane technologie

Całość kodu modułu analitycznego napisana jest w języku *Python*, implementując go wdrażano wiele wzorców, receptur opisanych w książce "Fluent Python"[16, Ramalho, L. 2015].

Flask

Licencja: BSD3

Flask jest napisanym w języku Python mikro-frameworkiem. Głównym jego zastosowaniem jest tworzenie niewielkich aplikacji sieciowych czy REST API. Posiada wbudowany serwer deweloperski i debugger. Stawia duży nacisk na zwięzłość kodu, co pozwala na szybkie prototypowanie ale i późniejsze rozwijanie aplikacji.

Django

Licencja: BSD3

Jest to jeden z największych i najszybciej rozwijanych projektów Open Source. Zdecydowaliśmy się na niego ze względu na to, że w tym module kluczowa jest wydajność generowania odpowiednich statystyk, ale też duża elastyczność w pisaniu kodu. Ważną zaletą Django jest wbudowany ORM który znacząco ułatwia współpracę z bazą danych.

Celery (i Luigi)

Licencja: BSD3

Celery[6, Celery] jest asynchroniczną kolejką zadań. Zadania są dystrybuowane do kolejki z różnych źródeł, np. z aplikacji sieciowej. Celery jest przeznaczone głównie do operacji zlecanych oraz wykonywanych w czasie rzeczywistym. Jako kolejkę lub bazę na zlecone zadania możliwe jest wykorzystanie wielu backendów np. redis czy rabbitmq.

Początkowe plany zakładały użycie w tym miejscu frameworka **Luigi**[17, Luigi]. Jest to stworzone przez twórców aplikacji *Spotify* narzędzie do łączenia ze sobą kolejnych funkcji / etapów, tworząc łańcuch przetwarzania (*Pipeline*). Okazało się jednak, że przewidziane zastosowania narzędzia obsługują etapy innego typu niż te wymagane przez proces ekstrakcji technologii. Luigi przeznaczony jest do łączenia 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. Możliwe jest uruchamianie ich jedynie za pomocą linii poleceń.

Zdecydowaliśmy się na porzucenie Luigiego na rzecz frameworka *Celery*, którego obsługa zadań jest tym czego potrzebujemy. Minusem takiego wyboru jest utrata odporności na awarie (Luigi zapisuje stan po każdym zadaniu i wraca do niego po awarii), lecz dużym zyskiem jest zwiększona łatwość implementacji.

2.4. Aplikacja WWW

Kod źródłowy znajduje się pod adresem:

- github.com/jobsbrowser/frontend.

Aplikacja będąca ostatnim komponentem systemu oraz jedynym który wchodzi w bezpośrednią interakcję z użytkownikiem.

2.4.1. Wymagania

Głównym wymaganiem funkcjonalnym aplikacji jest prezentowanie danych użytkownikowi w jak najbardziej przystępny oraz przejrzysty sposób. Rozwiązaniem okazała się organizacja strony na trzy zakładki pomiędzy którymi użytkownik może się swobodnie przełączać. Zakładki odpowiadają kolejno następującym możliwościom:

- wyświetlanie statystyk na podstawie wpisanych w polu wyszukiwania technologii
- wyszukiwanie ofert na podstawie wpisanych w polu wyszukiwania technologii
- przeglądanie strony informacyjnej wraz ze statystykami zbiorczymi systemu

Wymagania niefunkcjonalne strony sprowadzają się do:

- **niezawodności** - usługa powinna być dostępna cały czas, ponieważ jest ona usługą końcową z której użytkownicy powinni móc korzystać w każdej chwili.
- **wydajności** - liczba użytkowników korzystających ze strony może być bardzo duża, dlatego też aplikacja powinna działać wydajnie, aby jej niedostępność lub zbyt wolne działanie nie przynosiły negatywnych odczuć użytkownikowi.

2.4.2. Wykonanie (SPA)

Aplikacja wykonana została w technologii *SPA* z użyciem frameworka **Vue.js**[18, VueJS] Podczas prac nad aplikacja przydatne okazały się porady oraz przykłady z książki "Learning Vue.js 2"[19, Filipova, O. 2016]

SPA to skrót od *Single Page Application*. Nazywamy tak stronę internetową, która do zmiany prezentowanej użytkownikowi treści, np. podczas przejścia do innej zakładki w

menu, nie wymaga przeładowania - czyli pobierania raz jeszcze pełnego dokumentu HTML przez przeglądarkę użytkownika. Znaczącą rolę pełni na takiej stronie język JavaScript, To napisany w nim kod odpowiada za obsługę wydarzeń na stronie (np. kliknięcie przycisku w menu), wykonywanie zapytań do serwera w tle (przy użyciu techniki AJAX) oraz podmianę widocznej przez użytkownika struktury HTML tak aby pasowała do nowych danych.

Aplikacje takie często uważane są za przyjemniejsze w obsłudze, ponieważ sprawiają wrażenie bardziej "natywnych" - znacząco skracają czas oczekiwania użytkownika podczas interakcji czy urozmaicają go płynnymi przejściami (paskami postępu, kręcącymi się spinnerami, itp.). Są one bardziej zoptymalizowane, ponieważ podczas interakcji z użytkownikiem nie wymagają od serwera wielokrotnego generowania tych samych danych, np. powtarzającego się kodu HTML.

Do wyboru tej technologii przyczyniły się trzy zasadnicze powody:

1. Aplikacja WWW pełniąca rolę interfejsu użytkownika powinna zapewniać dynamiczną interakcję z użytkownikiem. Zastosowanie odpowiedniego frameworku znacznie usprawnia proces implementacji funkcjonalności takich jak auto-uzupełnianie podczas wprowadzania nazwy technologii czy rozwijanie kafelka zawierającego informacje o ofercie.

2. Frameworki napisane w języku JavaScript, poza samą możliwością stworzenia SPA oferują często rozbudowany system do tworzenia samego interfejsu. Skupiają w sobie takie rzeczy jak obsługa kontrolek, data binding czy gotowe style CSS. Wykorzystanie ich znacznie usprawnia prace nad aplikacją WWW.

3. Czasy generowania statystyk dla nowego zbioru technologii są nieco dłuższe niż przeciętny czas ładowania się nowej strony, do którego przyzwyczajony jest użytkownik. Wiąże się to z obliczeniami które odbywają się w tle po stronie modułu analitycznego. Zastosowanie estetycznej animacji widocznej w trakcie ładowania się danych zwiększy komfort użytkownika, a możliwość pobrania z serwera samych danych, bez zbędnego narzutu w postaci kodu HTML, skróci potrzebny na odpowiedź czas.

2.4.3. Interakcja z użytkownikiem i prezentacja danych

Interakcję użytkownika ze stroną można sprowadzić do możliwości wybrania jednej bądź wielu z sugerowanych technologii na pasku wyszukiwania oraz obserwacji zmian zachodzących w treści strony. Pasek ten widoczny jest po otworzeniu zakładki ze statystykami

(otwarta domyślnie) lub z wyszukiwarką ofert. Podczas wybierania technologii aktywny jest system auto-uzupełniania, który chroni użytkownika przed wprowadzeniem niewłaściwych danych - np. nazwy technologii która nie istnieje, czy zawierającej literówkę. Auto-uzupełnianie daje również swego rodzaju podgląd na to, jakie technologie możliwe są do zbadania przy użyciu systemu.

Statystyki

Na tej zakładce prezentowane są przewidziane w aplikacji wykresy, dotyczące wybranego zbioru technologii. Zawartość zakładki podmienia się automatycznie po zaobserwowanej zmianie na pasku wyszukiwania, a także w tle, np. gdy zmiana taka zostanie wykonana z poziomu innej zakładki. Zakładki statystyk oraz wyszukiwania dzielą bowiem ze sobą pasek wyszukiwania.

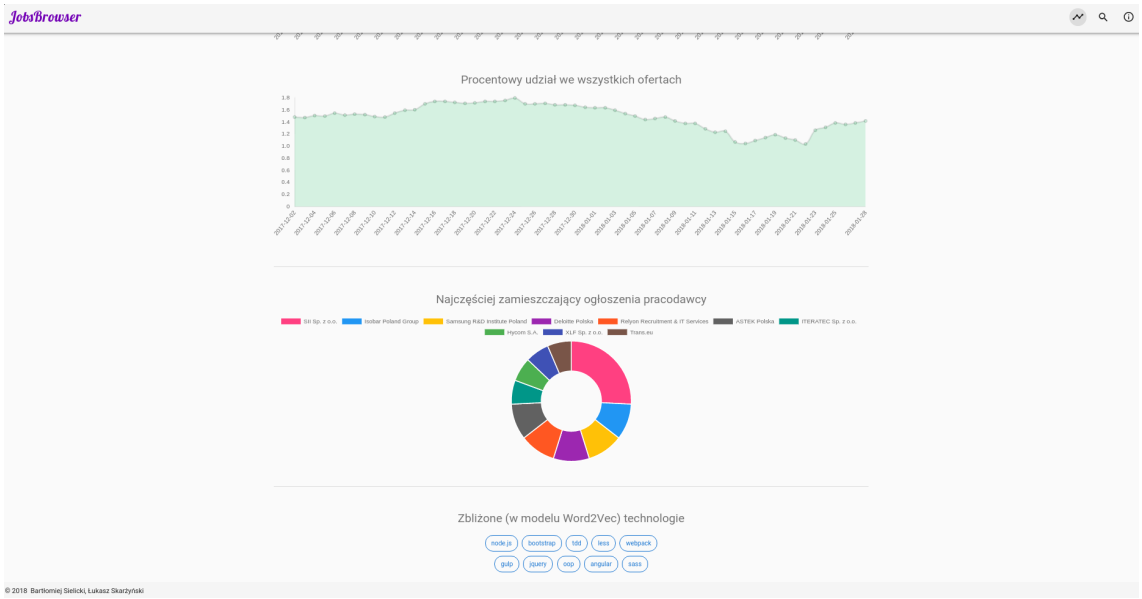
Prezentowane użytkownikowi informacje to:

1. Wykres porównujący dwie lub więcej technologii ze sobą. Porównanie wykonane jest pod względem ilości ofert je zawierających i aktywnych danego dnia. Widoczny jest on na samej górze zawartości zakładki, ale dopiero po wybraniu więcej niż jednej technologii. Informacją płynąca z tego wykresu może być np. jak konkurujące ze sobą technologie mają się do siebie na rynku pracy i jak wyglądało to na przestrzeni czasu.

2. Wykresy ilości ofert zawierających każdą z wybranych technologii. Widoczne są dwa - jeden na którym przedstawiona jest całkowita liczba aktywnych ofert w czasie, oraz drugi mówiący ile procent wszystkich ogłoszeń z branży IT one stanowią. Informacją płynąca z tego wykresu może być np. czy popularność danej technologii rośnie lub maleje na przestrzeni czasu.

3. Wykres kołowy pracodawców. Wyszczególnionych jest na nim 10 najczęściej zamieszczających ogłoszenia wymagające danej technologii pracodawców. Informacją płynącą z tego wykresu może być np. jak wygląda zapotrzebowanie na daną technologię wśród największych firm z branży IT.

4. Lista dziesięciu najbardziej zbliżonych w modelu Word2Vec technologii. Informacją z niej płynącą może być np. jakie technologie występują często wraz z tymi wybranymi przez użytkownika w jednym ogłoszeniu.

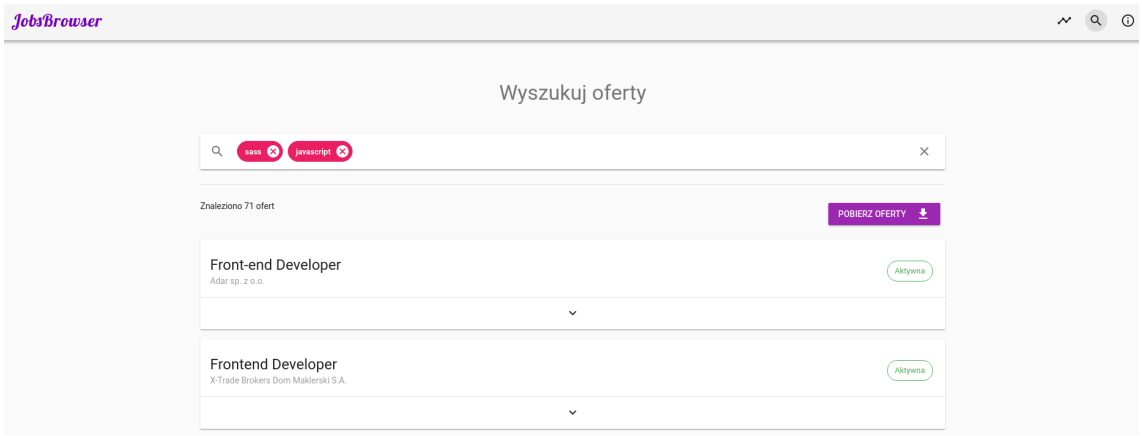


Rysunek 2.6: Widok statystyk z wykresami prezentowanymi na stronie.

Wyszukiwanie

Zakładka wyszukiwarki pozwala na przeglądanie zarówno aktywnych jak i archiwalnych ofert zawierających każdą z wybranych technologii. Dostępne są podstawowe informacje o każdej ofercie oraz odnośnik do oryginalnego ogłoszenia w serwisie Pracuj.pl. Prezentowane są w postaci listy, na której każdy element jest rozwijalny, pokazując więcej szczegółów.

Z poziomu tej zakładki użytkownik może też przeprowadzić operację eksportu danych. Nad listą znalezionych ofert, a pod paskiem wyszukiwania znajduje się przycisk pozwalający na pobranie ofert spełniających dane kryterium wyszukiwania w formacie JSON.

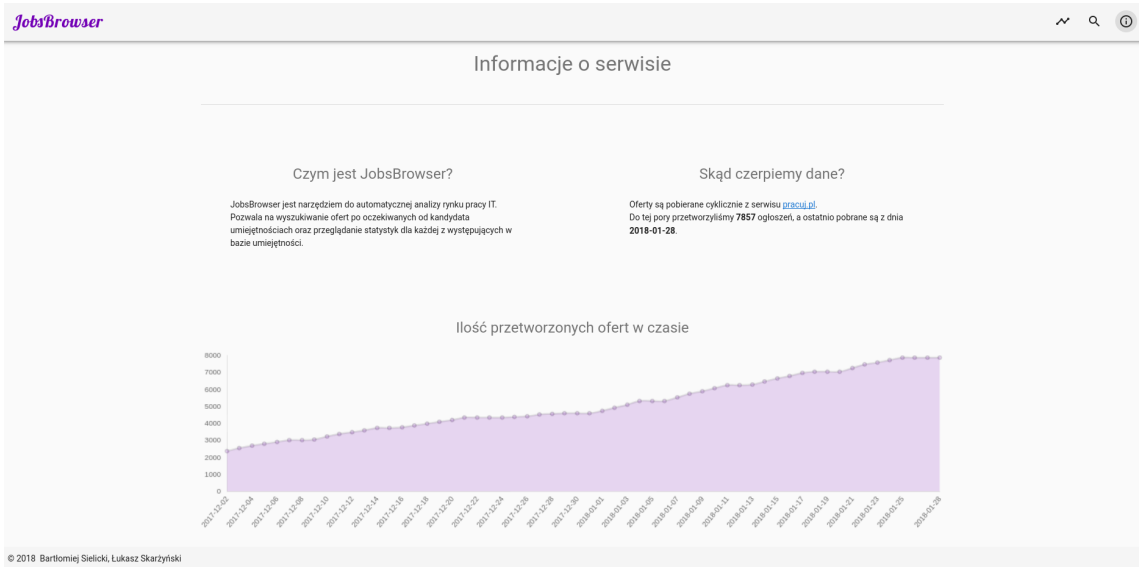


Rysunek 2.7: Widok wyszukiwania z widocznym przyciskiem do eksportu danych.

Informacje

Rolą ostatniej zakładki jest prezentacja użytkownikowi informacji dotyczących serwisu jako całości. Zawiera kilka zdań o projekcie oraz dane takie jak

- datę ostatniego przetwarzania ofert
- liczbę wszystkich przetworzonych pod względem technologii ofert w bazie
- wykres powyższej wartości względem czasu



Rysunek 2.8: Widok informacji o systemie.

2.4.4. Wykorzystane technologie

VueJS

Licencja: MIT

Aplikacja została napisana we frameworku VueJS[18, VueJS]. Jest to framework napisany w języku JavaScript oparty na architekturze MVVVM (Model-View–View-Model). VueJS umożliwia tworzenie lekkich oraz szybkich aplikacji sieciowych. Budowanie aplikacji polega na tworzeniu tzw. komponentów i składaniu z nich całej aplikacji. VueJS wydaje się być dobrym wyborem przy tworzeniu aplikacji, które opierają się na konsumowaniu danych z różnych API oraz prezentowaniu ich użytkownikowi.

Do budowy aplikacji wykorzystano kilka otwartoźródłowych rozszerzeń:

- `vuetify`[20, Vuetify] - framework dostarczający gotowe komponenty stworzone w stylu material design
- `vuex`[21, Vuex] - system (wzorzec) zarządzania stanem aplikacji napisanej w VueJS. Zapewnia miejsce na przechowywanie danych, które będą dostępne przez każdy komponent oraz kontroluje ich nadpisanie czy zmianę
- `chart.js`[22, Chart.js] - biblioteka używana do tworzenia estetycznych wykresów
- `axios`[23, axios] - klient HTTP wykorzystany do komunikacji z modułem analitycznym

Rozdział 3

Instrukcja obsługi aplikacji WWW

3.1. Dostęp do strony

Funkcjonalność serwisu uzależniona jest od ilości przetworzonych ofert znajdujących się w bazie danych (świeżo uruchomiona instancja systemu będzie pusta). Zalecane jest zatem korzystanie z oficjalnej wersji serwisu, dostępnej pod adresem `http://jobsbrowser.pl` (2018-01-29).

3.2. Zakładka statystyk

Jest to pierwsza zakładka którą jako użytkownik widzimy po wprowadzeniu w przeglądarce adresu aplikacji. W górnej części strony znajduje się pasek menu pozwalający na zmianę zakładki, a w środkowej pole do wyboru technologii.

Po wyborze technologii i wciśnięciu klawisza Enter zakoloruje się on, a pod spodem pojawią się dwa wykresy. Wybraną technologię można usunąć lub dodać do niego kolejny pisząc w polu i ponownie wciskając Enter.

Na pierwszym wykresie przedstawiona będzie ilość aktywnych ofert zawierających wybrane technologie konkretnego dnia, a na drugim jaką procentowo część wszystkich aktywnych wówczas ofert z branży IT one stanowiły. Wykresy są w pewnym stopniu interaktywne. Tzn. nie pozwalają na zmianę skali czy zakresu, ale pozwalają na dokładne zbadanie wartości w określonym dniu najeżdżając na wykres kursorem myszy.

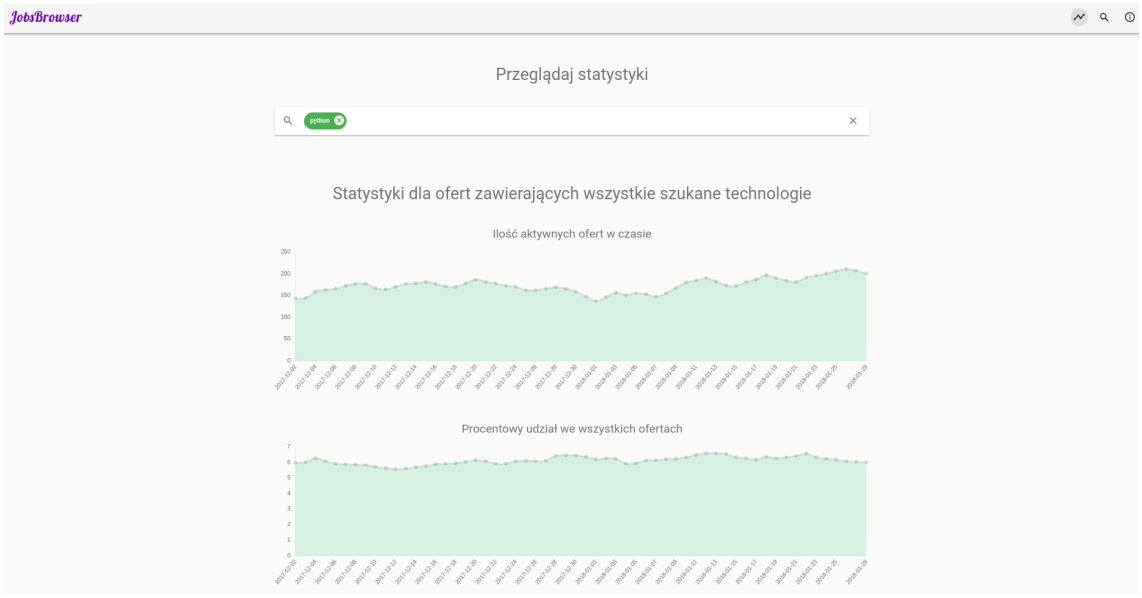
3.3. Zakładka wyszukiwania

Drugą dostępną z menu zakładką jest ta z widokiem wyszukiwania. Tutaj ponownie zobaczymy identyczne pole do wyboru technologii, jednak tym razem pod spodem pojawi nam się lista znalezionych ofert które je zawierają. Każdą z nich możemy rozwinąć żeby zobaczyć więcej informacji, w tym link do oryginalnego ogłoszenia czy pozostałe wykryte w ofercie technologie. Lista jest paginowana, co zwiększa komfort użytkownika i poprawia szybkość ładowania wyników.

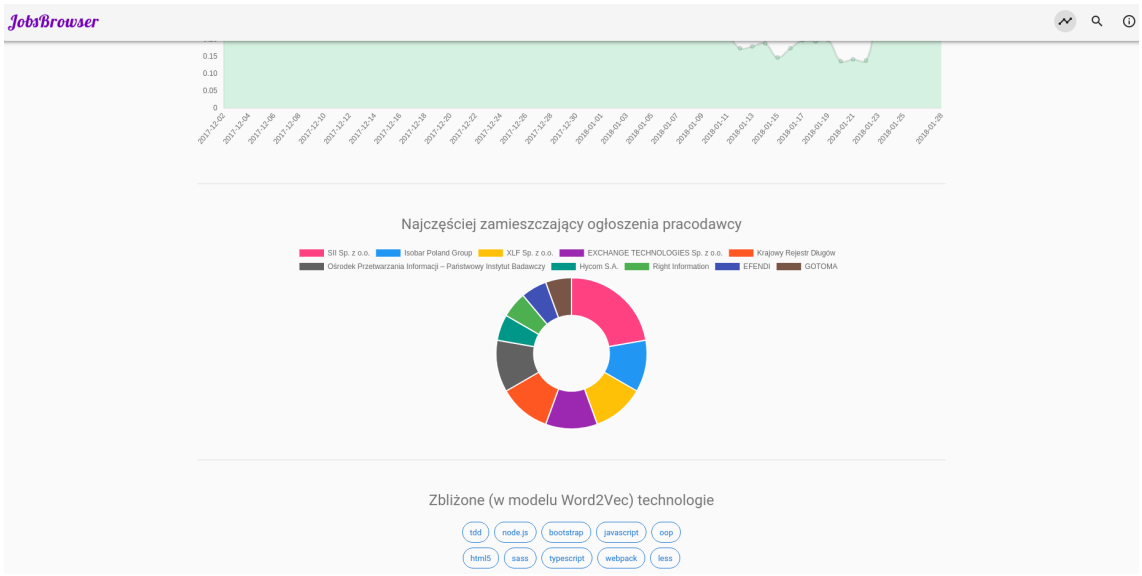
Ogłoszenia na liście posortowane są wg daty ich wygaśnięcia (znaleźć na niej można również ogłoszenia archiwalne). O statusie danej oferty informuje etykieta w prawym górnym rogu.

3.4. Zakładka informacji

Ostatnią zakładką jest ta poświęcona informacjom o serwisie oraz statystykom zbiorczym. Zobaczymy tam krótki opis projektu oraz wykres ilości wszystkich przetworzonych przez niego ofert względem czasu.

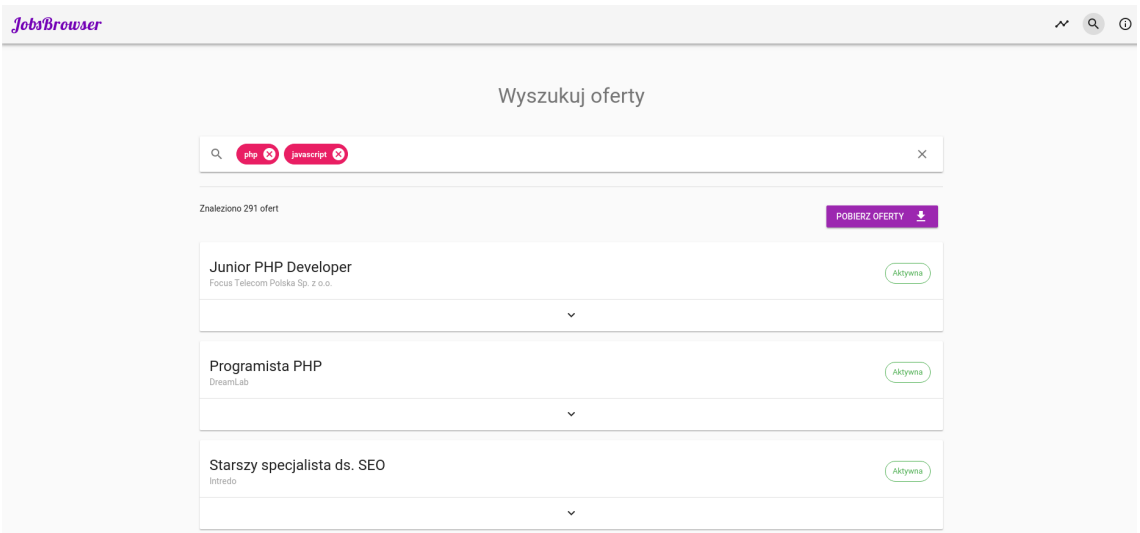


Rysunek 3.1: Zakładka statystyk z wybranym jednym tagiem.

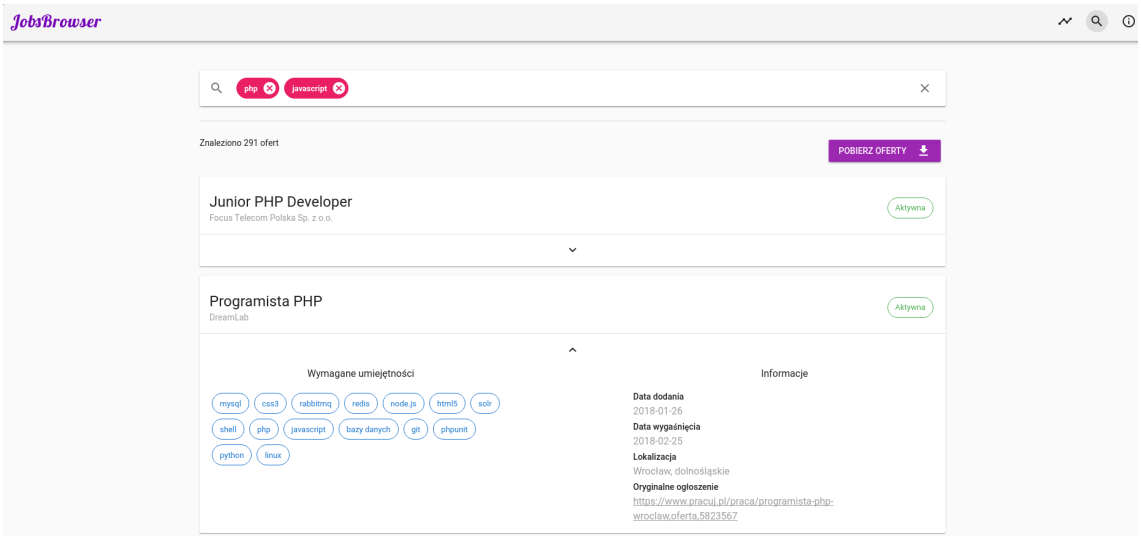


Rysunek 3.2: Wykres pracodawców i lista podobnych technologii.

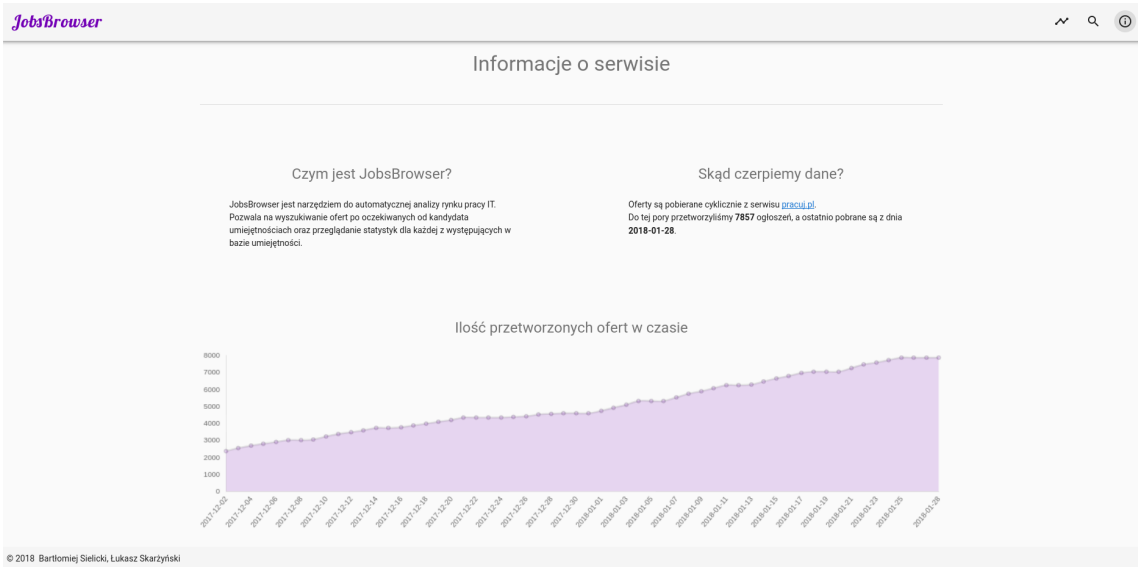
Rysunek 3.3: Wykres porównania tagów z osobna.



Rysunek 3.4: Zakładka wyszukiwania.



Rysunek 3.5: Rozwinięte ogłoszenie.



Rysunek 3.6: Zakładka informacji.

Rozdział 4

Podsumowanie

Głównym celem pracy było stworzenie narzędzia oferującego możliwość opisanej we wstępnych wymaganiach analizy zmian zachodzących na rynku pracy w branży IT. Za kluczowe elementy przyjąć można:

- automatyzację samego systemu - tak, aby bez ingerencji użytkownika obsługiwał pojawiające się w czasie jego działania oferty pracy
- poprawną analizę - polegającą na ekstrakcji nazw odpowiednich technologii z ich treści
- implementację przyjaznego interfejsu użytkownika w postaci aplikacji WWW dającej dostęp do opracowanych przez system statystyk

Założenia te można uznać za spełnione, a ponadto zauważono możliwości rozwoju systemu niosące ze sobą potencjalne korzyści.

Serwisy dostarczające dane

Aplikacja, zgodnie ze wstępnymi założeniami korzysta z tylko jednego serwisu (Pracuj.pl) jako źródła ofert. Jest to jeden z największych tego typu serwisów na polskim rynku, jednak niewątpliwie jakość danych nie ucierpiałaby gdyby system analizował również oferty pochodzące z innych źródeł. Modularna struktura systemu nie wymagałaby procesu jego przerabiania, a jedynie dopisania nowych robotów w przypadku chęci dodania nowego serwisu. Nie byłaby to więc funkcjonalność trudna do wdrożenia

Bardziej zaawansowana analiza tekstu

Zbiór technologii które system wykrywa w treści analizowanych ofert, choć łatwy do powiększenia, jest ograniczony. Być może dzięki odpowiednim zasobom czasu i wiedzy dałoby się dopracować sposób wykorzystania opisanych w rozdz. 2.3.4 algorytmów, co pozwoliłoby na samodzielne rozpoznawanie przez system pewnych słów jako technologie. Jest to temat bez wątpienia ciekawy, i z pewnością wart rozważenia np. jako kontynuacja w ramach pracy magisterskiej.

Dodatek A: Dokumentacja powykonawcza i opis testów

Dodatek ten zawiera opis kodu źródłowego oraz testów poszczególnych komponentów systemu.

4.1. Robot zbierający dane

Kod źródłowy tego modułu znajduje się na załączonej do pracy płycie DVD, w katalogu `src/scraper/`.

4.1.1. Struktura plików kodu źródłowego

Poniżej przedstawione jest drzewo katalogów oraz plików modułu, wraz z krótkim omówieniem:

```
jobsbrowser
    jobsbrowser
        __init__.py
        extractors.py    # definicje klas ekstraktujących linki z ofertami
        items.py         # definicje klas przechowujących zebrane dane
        loaders.py       # definicje klas ładujących zebrane dane
        loaders.py       # definicje klas ładujących zebrane dane
        middlewares.py
        pipelines.py     # definicje kolejnych etapów przetwarzania danych
        settings.py      # ustawienia pająków
        spiders          # katalog z definicjami pająków zbierających dane
            __init__.py
            pracuj.py    # definicja pająka 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
```

4.1.2. Opis klas i metod

- **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 analitycznego.
  - `process_item` - metoda w której za pomocą zapytania HTTP wysyłany jest aktualnie przetwarzany element (oferta).
- **JobsBrowserSpiderMiddleware** - klasa wygenerowana automatycznie podczas tworzenia projektu za pomocą Scrapy'ego.
- **PracujLinkExtractor** - klasa wyciągająca linki do ofert pracy w serwisie pracuj.pl.
- **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_link` - metoda sprawdzająca czy dane ogłoszenie nie jest już w bazie.
  - `parse_item` - zajmuje się wyciągnięciem potrzebnych danych z aktualnie przetwarzanej podstrony
  - `start_requests` - generuje zapytania HTTP do każdej strony z daną kategorią.

4.1.3. Testy jednostkowe

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

Poniżej znajduje się lista plików zawierających testy jednostkowe 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 żadne 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 modułem 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.

4.2. Moduł analityczny - podmoduł obsługi robota i ekstrakcji technologii

Jak wspomniano w rozdz. 2.3, moduł podzielony jest na dwa komponenty, korzystające z różnych technologii, a nawet różnych baz danych. Praca nad nimi przebiegała z użyciem dwóch niezależnych od siebie repozytoriów, tak więc i na załączonej płycie DVD komponenty znaleźć można w odrębnych katalogach.

Omawianym w tej sekcji podmodułem jest ten odpowiedzialny za obsługę robota zbierającego dane oraz ekstrakcję technologii. Jego kod źródłowy znaleźć można w katalogu `src/analytics/pipeline` na załączonej płycie, a jego struktura przedstawia się następująco:

4.2.1. Struktura plików kodu źródłowego

```
jobsbrowser
    api                          # część odpowiedzialna za obsługę scrapera
        __init__.py
        resources.py             # definicja endpointów API
        settings.py              # ustawienia API
        spec.yml                 # specyfikacja API w standardzie OpenAPI(swagger)
    pipeline                     # łańcuch przetwarzania danych - ekstrakcja techn
        _app.py
        chains.py                # konstrukcja łańcucha
        __init__.py
        settings.py              # ustawienia Celery oraz MongoDB
        tasks                    # moduł poszczególnych zadań pipeline
            bases.py             # bazowe klasy zadań
            exceptions.py        # wyjątki modułu zadań pipeline
            __init__.py
            postprocess.py       # zadania wykonywane na koniec przetwarzania
            preprocess.py        # zadania przygotowujące ofertę to ekstrakcji cech
            process.py           # zadania ekstrakcji technologii
            utils.py             # pożyteczne, mniejsze funkcje
    __init__.py
tests                            # katalog z testami podmodułu
    api                          # katalog z testami części obsługującej robota
        __init__.py
        test_api_resources.py    # testy endpointów API
    pipeline                     # katalog z testami modułu łańcucha
        __init__.py
        tasks
            __init__.py
            test_process.py      # testy zadań ekstrakcji pipeline
            test_postprocess.py  # testy zadań końcowych pipeline
            test_preprocess.py   # testy zadań początkowych pipeline
```

4.2.2. Opis klas i metod

Obsługa scrapera

- `add_offer` - funkcjas 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.
- `update_offer` - funkcja implementująca uaktualnianie oferty z bazy.
- `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.

Łańcuch przetwarzania

- `MongoDBTask` - klasa bazowa dla etapów łańcucha przetwarzania danych korzystających z danych znajdujących się w bazie MongoDB.
- `TagsFindingTask` - klasa bazowa dla etapu pipeline'a, który znajduje technologie(tagi) w ofercie.
- `LanguageNotSupported` - klasa błędu, rzucanego w przypadku próby przetwarzania oferty napisanej w języku innym niż polski.
- `prepare` - etap przygotowujący dane do dalszego przetwarzania.
- `strip_html_tags` - etap usuwający tagi HTML z ofert.
- `tokenize` - etap rozbijający opis oferty na tokeny.
- `detect_language` - etap przeprowadzający detekcję języka oraz kończący przetwarzanie w przypadku detekcji języka innego niż polski.
- `remove_stopwords` - etap usuwający tokeny składające się ze słów nieistotnych.
- `find_tags` - etap znajdujący tagi z technologiami spośród tokenów oferty.
- `save_to_mongodb` - etap zapisujący ofertę do bazy MongoDB.
- `pracuj_pipeline` - pipeline wykorzystywany do ekstrakcji z oferty niezbędnych cech, takich jak tagi z technologiami wymienionymi w opisie oferty.

4.2.3. Testy jednostkowe

Aby uruchomić testy jednostkowe, w konsoli należy wpisać polecenie `tox`. Poniżej znajduje się opis testów z pliku `jobsbrowser/tests/api/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.
- `test_update_offer_resource_query_mongo_db` - test sprawdza czy po wykonaniu zapytania HTTP PUT na endpoint /offer wykonywana jest próba pobrania oferty oraz uaktualnienia jej w bazie.

Oraz tych dotyczących łańcucha przetwarzania:

- `pipeline.tasks.test_preprocess.py` - plik z testami etapów przygotowujących oferty do ekstrakcji technologii.
  - `TestDetectLanguage` - klasa zawierająca metody testujące etap detekcji języka w którym napisane jest ogłoszenie.
  - `test_prepare_extract_proper_fields_from_offer` - test sprawdzający czy etap przygotowujący ofertę, tworzy oraz wybiera odpowiednie pola z całej oferty.
  - `test_remove_stopwords_return_tokens_without_stopwords` - test weryfikujący poprawne usunięcie tokenów będących słowami nieistotnymi.
  - `test_strip_html_tags_return_tokens_without_html_tags` - test sprawdzający czy etap poprawnie usuwa tagi HTML.
  - `test_tokenize_return_list_of_lowercase_tokens` - test sprawdzający czy etap zwraca odpowiednią listę tokenów składających się z małych liter.
- `pipeline.tasks.test_process.py` - plik z testami etapów wykonujących ekstrakcje technologii z oferty.
- `pipeline.tasks.test_postprocess.py` - plik z testami etapów wykonujących zadania po wykonaniu ekstrakcji technologii.

4.3. Moduł analityczny - podmoduł statystyk i wyszukiwarki

Kod źródłowy podmodułu znaleźć można w katalogu `src/analytics/backend`.

4.3.1. Struktura plików kodu źródłowego

Zaprezentowana poniżej struktura plikóœ i katalogów odpowiada szablonowemu projektowi działającemu z użyciem Django. Zdecydowaną większość funkcjonalności (filtrowanie, paginację, serializowanie modeli, wyszukiwanie po atrybucie) zapewniły gotowe, oferowane przez framework narzędzia. Główna część logiki znajduje się w pliku `offers/views.py` i jest to zoptymalizowane generowanie statystyk.

```
backend
    config                    # moduł ustawień (Django)
        __init__.py
        settings.py           # plik z ustawieniami
        urls.py               # konfiguracja adresów URL serwera
        wsgi.py               # aplikacja WSGI (np. do integracji z Apache)
    offers                    # moduł aplikacji
        migrations            # migracje (kod generujący schemat bazy danych na podstawie m
        __init__.py
        apps.py
        filters.py            # filtrowanie po tagach
        models.py             # modele oferty oraz technologii
        pagination.py         # paginacja (używana przy wyszukiwaniu ofert)
        serializers.py        # serializacja modeli
        tests.py              # testy jednostkowe
        views.py              # główna część logiki - implementacja interfejsu
manage.py                     # skrypt do zarządzania serwerem i jego uruchamiania
requirements.txt              # lista koniecznych do zainstalowania zależności
```

4.3.2. Opis klas i metod

Najbardziej istotnymi klasami są:

- **Offer** - Model oferty. Tutaj zadeklarowane są pola stanowiące strukturę tabeli w bazie danych.
- **Tag** - Model pojedynczej technologii. Zawiera jedynie pole na nazwę.
- **OffersListView** - klasa widoku obsługującego wyszukiwanie ofert.
- **OffersStatsView** - klasa widoku obsługującego generowanie statystyk
- **SystemInfoView** - klasa widoku obsługującego statystyki całościowe

każda z tych klas jest rozszerzeniem funkcjonalności oferowanych przez framework, dziedzicząc po odpowiedniej klasie bazowej. Niezależnie zaimplementowana logika znajduje się tylko w klasach **OffersStatsView** oraz **SystemInfoView** w postaci odpowiednich metod. W pozostałych dostosowanie klasy oferowanej przez framework sprowadza się do ustawienia odpowiednich atrybutów.

4.3.3. Testy jednostkowe

Aby uruchomić testy jednostkowe w konsoli należy wpisać polecenie `python manage.py test`.

Testy jednostkowe znajdują się w pliku `offers/offers.py` i obejemują klasy oraz metody implementujące logikę widoków znajdujących się w pliku `offers/views.py`.

4.4. Aplikacja WWW

Kod źródłowy aplikacji WWW znaleźć można w katalogu `src/www`.

4.4.1. Struktura plików kodu źródłowego

```
index.html            # główny plik HTML
package.json          # plik z zależnościami projektu
package-lock.json
public                # katalog na zasoby niezmieniające się np. zdjęcia, ikony
README.md
src                   # katalog z kodem źródłowym aplikacji
    App.vue           # główny komponent aplikacji
    components        # katalog z komponentami UI
```

```
        LineChart.js  # komponent obsługujący rysowanie wykresów liniowych
        Menu.vue      # komponent reprezentujący menu aplikacji
        Offer.vue     # komponent reprezentujący widok oferty
        TagInput.vue  # komponent obsługujący wybieranie technologii przez użytkownika
    main.js           # plik wejściowy dla aplikacji
    pages             # katalog z komponentami reprezentującymi podstrony aplikacji
        Info.vue      # komponent prezentujący informacje o projekcie
        Search.vue    # komponent umożliwiający interaktywne wyszukiwanie ofert
        Stats.vue     # komponent pokazujący statystyki
    router
        index.js      # konfiguracje routingu (ścieżek URL) aplikacji
    store
         index.js     # konfiguracja oraz inicjalizacja vuex-store
webpack.config.js
```

4.4.2. Opis klas i metod

- komponenty UI
  - `Menu` - komponent generujący widok górnego menu.
  - `TagInput` - komponent obsługujący pobieranie technologii (tagów) od użytkownika w interaktywny sposób.
  - `Offer` - komponent generujący widok listy ofert z odpowiednimi tagami pobranymi od użytkownika poprzez komponent `TagInput`.
  - `LineChart` - komponent renderujący wykresy liniowe dla technologii pobranych dzięki komponentowi `TagInput`.
- komponenty stron
  - `Search` - komponent korzystający z komponentów: `Menu`, `TagInput` oraz `Offer`. Pobiera informacje o ofertach z pobranymi tagami i prezentuje je w formie listy.
  - `Info` - komponent korzystający z komponentów: `Menu`. Pobiera informacje na temat projektu **JobsBrowser** oraz je prezentuje.
  - `Stats` - komponent korzystający z komponentów: `Menu`, `TagInput` oraz `LineChart`. Pobiera statystyki dotyczące wybranych technologii oraz prezentuje je użytkownikowi w formie wykresów.

Dodatek B: Zawartość załączonej płyty DVD

1. `thesis.pdf` - plik PDF zawierający całą treść pracy
2. `title.pdf` - strona tytułowa
3. `abstract_pl.pdf` - streszczenie w języku polskim
4. `abstract_en.pdf` - streszczenie w języku angielskim
5. `src/` - katalog z kodem źródłowym systemu. Jego zawartość jest szerzej opisana w dokumentacji powykonawczej (dod. A)
6. `vm/` - obraz maszyny wirtualnej z zainstalowaną i skonfigurowaną instancją systemu. Instrukcja jej uruchomienia znajduje się poniżej.

Uruchomienie maszyny wirtualnej

Obraz maszyny wirtualnej zawiera system operacyjny Debian 9 i przeznaczony jest do użytku z narzędziem Virtualbox. Po uruchomieniu obrazu należy zalogować się przy pomocy danych:

- użytkownik: `vagrant`
- hasło: `vagrant`

a następnie wykonać komendę:

```
jobsbrowser_start
```

Uruchomiona zostanie sesja programu *tmux* z trzema oknami. W nich uruchomione są poszczególne moduły systemu. Aplikacja WWW jest teraz dostępna na porcie `8080` uruchomionej maszyny.

Podział pracy

Na każdym etapie podczas pisania pracy oraz budowania systemu obowiązki były równo i sprawiedliwie podzielone pomiędzy uczestników zespołu. Podział ten prezentuje się następująco

Tabela 4.1: Podział pracy

Autor	Wykonana część pracy
Bartłomiej Sielicki	Implementacja komunikacji pomiędzy robotem zbierającym dane a modułem analitycznym
	Implementacja procesu ekstrakcji technologii
	Implementacja podmodułu generującego statystyki
	Praca nad warstwą wizualną aplikacji WWW
	Praca nad warstwą logiczną aplikacji WWW
	Przygotowanie instrukcji obsługi aplikacji WWW
	Przygotowanie rozdz. 1 pracy
	Przygotowanie wstępu, podsumowania oraz słownika pojęć
	Przygotowanie pacy do druku
Łukasz Skarżyński	Implementacja robota zbierającego dane
	Przygotowanie środowiska pracy (repozytoria)
	Obsługa bazy MongoDB z poziomu języka Python
	Próby analizy tekstu przy pomocy algorytmów ML
	Implementacja możliwości wyszukiwania ofert
	Wyszukiwanie najbliższych technologii w modelu W2V
	Przygotowanie załączonej do pracy dokumentacji powykonawczej
	Przygotowanie instrukcji obsługi
	Przygotowanie rozdz. 2 pracy

Spis rysunków

Spis tabel

Bibliografia

[1] **Scrapy**. Zestaw narzędzi umożliwiający szybką, wydajną ekstrakcję informacji ze stron internetowych.
Dostępne pod adresem: *https://scrapy.org* [2017-12-16].

[2] **MongoDB**. Nierelacyjna baza danych, przechowująca rekordy w formacie bson.
Dostępne pod adresem: *https://www.mongodb.com/* [2017-12-16].

[3] **Flask**. Framework wykorzystywany do tworzenia stron internetowych oraz api.
Dostępne pod adresem: *http://flask.pocoo.org/* [2017-12-16].

[4] **Discovery, G.C.J.** Platforma firmy google, pozwalająca na automatyczne dopasowywanie ofert pracy z podaniami.
Dostępne pod adresem: *https://cloud.google.com/job-discovery/* [2017-12-16].

[5] **StackOverflow**. Portal dla programistów i osób zainteresowanych it.
Dostępne pod adresem: *https://stackoverflow.com/* [2018-01-13].

[6] **Celery**. Framework pozwalający na asynchroniczne wykonywanie zadań i łańcuchów przetwarzania (pipeline).
Dostępne pod adresem: *http://www.celeryproject.org/* [2017-12-16].

[7] **Rake-NLTK**. Implementacja w języku python (z użyciem nltk) algorytmu rake (en. rapid automatic keyword extraction). wykorzystywany do automatycznej ekstrakcji słów kluczowych.
Dostępne pod adresem: *https://github.com/csurfer/rake-nltk* [2017-12-16].

[8] **TF-IDF**. Algorytm wyznaczania wagi słów na podstawie ich wystąpień.
Dostępne pod adresem: *https://en.wikipedia.org/wiki/Tf-idf* [2018-01-21].

[9] **Goldberg, Y. & Levy, O.** Word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method.
Dostępne pod adresem: *https://arxiv.org/abs/1402.3722* [2018-01-27].

[10] **Bengio, Y., Réjean, D. & Pascal, V.** Advances in neural information processing systems 13 (nips'00).
Dostępne pod adresem: *http://www.iro.umontreal.ca/~lisa/publications2/index.php/publications/show/64* [2018-01-27].

[11] **Gensim**. Biblioteka przeznaczona do przetwarzania języka naturalnego oraz pozyskiwania informacji w tekscie.

Dostępne pod adresem: *https://github.com/RaRe-Technologies/gensim* [2018-01-27].

[12] **Maaten, L. van der & Hinton, G.** Visualizing high-dimensional data using t-sne.
Dostępne pod adresem: *https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf* [2018-01-27].

[13] **Django**. Framework wykorzystywany do tworzenia aplikacji webowych oraz rest api.
Dostępne pod adresem: *https://www.djangoproject.com/* [2018-01-13].

[14] **Greenfeld, D.R. & Greenfeld, A.R.** 2017. *Two scoops of django 1.11: Best practices for the django web framework* 4th ed., Two Scoops Press.

[15] **SQLite**. System zarządzania relacyjną bazą danych.
Dostępne pod adresem: *https://www.sqlite.org/* [2018-01-13].

[16] **Ramalho, L.** 2015. *Fluent python*, O'Reilly.

[17] **Luigi**. Moduł pozwalający budować skomplikowane łańcuchy(en. pipelines) zadań, operacji czy transformacji na danych.
Dostępne pod adresem: *https://luigi.readthedocs.io/en/stable* [2017-12-16].

[18] **VueJS**. Framework wykorzystywany do tworzenia progresywnych aplikacji internetowych.
Dostępne pod adresem: *https://vuejs.org* [2017-12-16].

[19] **Filipova, O.** 2016. *Learning vue.js 2* 1st ed., Packt Publishing.

[20] **Vuetify**. Zestaw komponentów ui do vuejs zbudowany przy użyciu wyglądu material design.
Dostępne pod adresem: *https://vuetifyjs.com/* [2018-01-13].

[21] **Vuex**. System zarządzania stanem komponentów vuejs.
Dostępne pod adresem: *https://vuex.vuejs.org* [2018-01-13].

[22] **Chart.js**. Biblioteka ułatwiająca rysowanie wykresów.
Dostępne pod adresem: *http://www.chartjs.org/* [2018-01-13].

[23] **axios**. Klient http napisany w node.js.
Dostępne pod adresem: *https://github.com/axios/axios* [2018-01-13].