Sanium.scraper

Mikroserwis służący do pobierania ofert pracy ze stron internetowych.

Bartłomiej Olszanowski

Michał Popiel

Błażej Darul

Poznań, 10 września 2020 r.

Spis treści

1. Opis aplikacji

Sanium.scraper to mikroserwis będący komponentem większej aplikacji internetowej, służącej do agregacji ogłoszeń o pracę. Usługa ta sczytuje aktualne oferty o prace z innych serwisów internetowych, a następnie zapisuje je w bazie danych. Cały proces gromadzenia danych jest wykonywany według automatycznie ustalanego harmonogramu.

2. Podział prac

Opis zadania	Osoba realizująca
Przygotowanie boilerplate oraz API Usługa harmonogramu	Bartłomiej Olszanowski
Usługa scrapera	Michał Popiel
Połączenie mikroserwisu z główną aplikacją	Błażej Darul

3. Wykorzystane technologie oraz narzędzia
    a. Python
        i. Python jest językiem o bardzo szerokim zastosowaniu, pozwalającym na napisanie dosłownie wszystkiego przy użyciu odpowiednich bibliotek lub frameworków, przez co bardzo wiele firm używa go w swoich aplikacjach.Jego interpretery są dostępne dla wielu systemów, co czyni go językiem wieloplatformowym. Python obecnie króluje w dziedzinach jak data science czy machine learning. Python może być także używany w Raspberry Pi - platformie komputerowej wielkości karty kredytowe

b. Selenium
    i. Selenium to popularne narzędzie do automatyzowania operacji wykonywanych przez przeglądarkę. Głównym zastosowaniem Selenium są testy aplikacji webowych, w szczególności ich frontentu. Za pomocą tego narzędzia możemy pokryć testami bardziej frontendową część aplikacji, np. kod JavaScript - coś czego zwykłe testy nie są w stanie obsłużyć. Taka funkcjonalność pozwala wykorzystać Selenium również do zaawansowanego sczytywania danych.

c. Flask
    i. Flask jest to mikro framework aplikacji webowych napisany w języku Python. Jest sklasyfikowany jako micro-framework, ponieważ nie wymaga określonych narzędzi ani bibliotek. Nie ma warstwy abstrakcji bazy danych, sprawdzania poprawności formularzy ani żadnych innych komponentów, w których istniejące biblioteki stron trzecich zapewniają wspólne funkcje. Jednak obsługuje rozszerzenia, które mogą dodawać funkcje aplikacji tak, jakby były zaimplementowane w samym Flasku. Istnieją rozszerzenia maperów obiektowo-relacyjnych, sprawdzania poprawności formularzy, obsługi przesyłania, różnych otwartych technologii uwierzytelniania i kilku popularnych narzędzi związanych ze strukturami. Rozszerzenia są aktualizowane znacznie częściej niż sam Flask.

d. APScheduler
    i. Advanced Python Scheduler (APScheduler) to biblioteka Pythona, która pozwala zaplanować późniejsze wykonanie kodu w języku Python, jednorazowo lub okresowo. Możesz dodawać nowe zadania lub usuwać stare w locie według własnego uznania. Jeśli przechowujesz swoje zadania w bazie danych, przetrwają one również ponowne uruchomienie harmonogramu i utrzymają swój stan. Po ponownym uruchomieniu harmonogramu uruchomi wszystkie zadania, które powinien był uruchomić, gdy był w trybie offline.

e. SQLAlchemy
    i. Otwartoźródłowa biblioteka programistyczna napisana w języku programowania Python i służąca do pracy z bazami danych oraz mapowania obiektowo-relacyjnego. Wspiera takie bazy danych jak: Firebird, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, SQLite oraz Sybase.

f. ChromeDriver
    i. WebDriver to narzędzie typu open source do automatycznego testowania aplikacji internetowych w wielu przeglądarkach. Zapewnia możliwości przechodzenia do stron internetowych, wprowadzania danych przez użytkownika, wykonywania JavaScript i nie tylko. ChromeDriver to samodzielny serwer, który implementuje standard W3C WebDriver. ChromeDriver jest dostępny dla Chrome na Androida i Chrome na komputery stacjonarne (Mac, Linux, Windows i ChromeOS).

g. Git
    i. Git to rozproszony system kontroli wersji. Stworzył go Linus Torvalds jako narzędzie wspomagające rozwój jądra Linux. Git stanowi wolne oprogramowanie i został opublikowany na licencji GNU GPL w wersji 2. Pierwsza wersja narzędzia Git została wydana 7 kwietnia 2005 roku, by zastąpić poprzednio używany w rozwoju Linuksa, niebędący wolnym oprogramowaniem, system kontroli wersji BitKeeper.

h. Github
    i. Git to system kontroli wersji. Jego pomysłodawcą i twórcą był Linus Torvalds. Git powstał jako narzędzie wspomagające rozwój jądra Linux. Git to system, który pozwala programistom zapisywać wszystkie zmiany w pisanym kodzie - w taki sposób, aby niczego nie stracili. Programista w każdej chwili może wrócić do poprzedniej wersji, jeżeli zajdzie taka potrzeba. Dzięki podglądowi poprzednich wersji, można prześledzić, jak program się rozwijał, cofnąć się, odzyskać przypadkowo utracone zmiany czy powrócić do wcześniejszych pomysłów. Z systemu kontroli wersji Git korzystają zarówno programiści w korporacyjnych zespołach, jak i ci pracujący samodzielnie. Warto wyrobić sobie ten dobry nawyk korzystania z systemu

kontroli wersji - nie raz okaże się, że uratuje nam to skórę i zapobiegnie marnowaniu czasu!

i. PyCharm
    i. PyCharm to zintegrowane środowisko programistyczne (IDE) dla języka programowania Python firmy JetBrains. Zapewnia m.in.: edycję i analizę kodu źródłowego, graficzny debugger, uruchamianie testów jednostkowych, integrację z systemem kontroli wersji. Wspiera także programowanie i tworzenie aplikacji internetowych w Flask .Jest oprogramowaniem wieloplatformowym pracującym na platformach systemowych: Microsoft Windows, GNU/Linux oraz macOS. Wydawany jest w wersji Professional Edition, która jest oprogramowaniem własnościowym oraz w wolnej wersji Community Edition, która pozbawiona jest jednak części funkcjonalności w porównaniu z wersją własnościową.

4. Architektura

Aplikacja składa się z warstw:

- Kontroler
- Harmonogram
- Scraper
- Model

5. Funkcjonalności

Endpoint	Funkcjonalność
GET /jobs	Zwraca informacje o istniejących zadaniach w systemie.
POST /add	Umożliwia zlecenie zadania scrapowania strony. Określenie jaka strona ma być scrapowana jest dokonywane poprzez przesłanie w ciele zapytania struktury danej strony.
GET /pause	Wstrzymuje harmonogram wykonywanych zadań.
GET /resume	Wznawia harmonogram wykonywanych zadań.
GET /jobs/<idx>	Zwraca informacje o danym zadaniu.

6. Szczegóły implementacyjne

Implementacja scrapera:

1. Opis klas:

Klasa **Scraper** – zarządza zczytywaniem danych z podanego serwisu o określonej strukturze, przechowuje strukturę strony (service_struct), zczytane dane (output), flagę debugowania (debug), opcje webdrivera (options) oraz webdriver (driver).

Klasa **MainPage** – odpowiada za zczytanie danych z listy znajdującej się na stronie, przechowuje informacje niezbędne do poprawnego zczytania danych ze strony (main_page_dict, locators), informacje o typie listy (list_type), adres strony (url), listę ofert (offer_list), webdriver (driver) oraz współrzędne scrollbara (scrollbar_x, scrollbar_y).

Klasa **MainPageLocators** – przechowuje lokalizatory wybranych elementów strony, lokalizator listy ofert (offer_list), lokalizator oferty (offer), lokalizator tytułu oferty (offer_title), lokalizator adresu oferty (offer_url).

Klasa **DetailPage** - odpowiada za zczytanie danych szczegółowych ze strony, przechowuje informacje niezbędne do poprawnego zczytania danych (detail_page_dict, locators), adres strony (url), webdriver (driver) i flagę debugowania (debug).

Klasa **DetailPageLocators** - przechowuje lokalizatory wybranych elementów strony, lokalizator tytułu oferty (offer_title), lokalizator technologii (offer_technology), lokalizator płacy oferty (offer_salary), lokalizator waluty płacy (offer_salary_currency), lokalizator położenia (offer_location), lokalizator doświadczenia (offer_experience), lokalizator typu zatrudnienia (offer_employment), lokalizator opisu oferty (offer_description), lokalizator logo firmy (company_logo), lokalizator nazwy firmy (company_name),

Klasa **Offer** – reprezentuje model oferty. Jest to klasa dziedzicząca po modelu SQLAlchemy, taki zabieg pozwala na translacje obiektów oferty na wpisy w bazie danych. Zawierają się w niej takie pola jak: id, name, description, experience, employment, technology, salary_from,

salary_to, currency, city, street, remote, contact, website, created_at, updated_at, expired_on, employer oraz origin_url.

Metody zebrane w pliku **JobController** – definiuje punkty końcowe API, przez jakie odbywa się komunikacja między usługami. Zawiera endpointy odpowiedzialne za zlecanie zadań scrapowania stron internetowych.

Metody zebrane w pliku **SchedulerService** – odpowiada za obsługę harmonogramu wykonywanych zadań.

2. Opis metod

2.1. Opis metod klasy **Scraper**

- **get_data()** - metoda odpowiadająca za dostęp do danych
  Metoda zwraca słownik (dict) z danymi.

- **set_data(data)** - metoda odpowiadająca za zmianę danych
  Metoda przyjmuje parametr:
  - data – parametr typu dict reprezentujący zebrane dane.

- **run_main_page_scrapping(target_number)** – metoda odpowiadająca za zczytanie danych z głównej strony.

  Metoda przyjmuje parametr:
  - target_number – parametr typu int reprezentujący docelową liczbę zebranych ofert.

  Metoda zwraca słownik (dict) z danymi.

- **run_detail_page_scrapping (target_id)** – metoda odpowiadająca za zczytanie wybranych danych z strony szczegółowej.

  Metoda przyjmuje parametr:
  - target_id – parametr typu string reprezentujący adres url oferty.

- **save_data()** – metoda odpowiadająca za zapis danych do pliku

2.2. Opis metod klasy **MainPage**

- **find_list_scrollbar_pos()** – metoda odpowiadająca za ustalenie parametrów scrollbar_x i scrollbar_y.

- **get_n_offers_from_list(n)** – metoda odpowiadająca za zczytanie n elementów z listy na stronie, działanie metody zależy od typu listy (list_type).
  Metoda przyjmuje parametr:
  - n – parametr typu int reprezentujący liczbę ofert do zczytania.

  Metoda zwraca słownik (dict) z danymi (uproszczony, n-elementowy zbiór ofert).

2.3. Opis metod klasy **MainPageLocators**

- **get_by(by_type)** – metoda statyczna odpowiadająca z przydzielenie typu lokalizatora
  Metoda przyjmuje parametr:
  - by_type – parametr typu string reprezentujący typ lokalizatora.

  Metoda zwraca łańcuch znaków (string)

2.4. Opis metod klasy **DetailPage**

- **format_salary(salary)** – metoda statyczna służąca wyciąganiu parametrów z łańcucha znaków
  Metoda przyjmuje parametr:
  - salary – parametr typu string reprezentujący płace (np. „1000- 3000 PLN”).

  Metoda zwraca słownik (dict) np. {'salary': str, 'salary_from': str, 'salary_to': str, 'currency': str}

- **format_location(location)** – metoda statyczna służąca wyciąganiu parametrów z łańcucha znaków
  Metoda przyjmuje parametr:
  - location – parametr typu string reprezentujący adres firmy.

  Metoda zwraca słownik (dict) np. {'city': str, 'street': str}

- **get_data()** – metoda odpowiadająca za zczytanie wybranych danych ze strony szczegółowej oferty.
  Metoda zwraca słownik (dict) z danymi szczegółowymi

2.5. Opis metod klasy **DetailPageLocators**

- **get_by(by_type)** – metoda statyczna odpowiadająca z przydzielenie typu lokalizatora
  Metoda przyjmuje parametr:
    - by_type – parametr typu string reprezentujący typ lokalizatora.

  Metoda zwraca łańcuch znaków (string)

2.6. Opis metod klasy **Offer**

- **create(**kwargs)** – metoda statyczna opowiadająca za tworzenie obiektu oferty oraz jednoczesne umieszczenie go w bazie danych.
  Metoda przyjmuje nazwane argumenty takie same jak pola klasy.
  Metoda zwraca obiekt oferty

- **find(idx, by)** – metoda statyczna odpowiadająca za wyszukiwanie ofert spełniających kryteria spośród wszystkich dostępnych.
  Metoda przyjmuje argumenty:
    - idx – id oferty
    - by – tablica parametrów, po których można filtrować wyszukiwane oferty

  Metoda zwraca jedną ofertę lub ich listę.

- **find_one(idx, by)** – metoda statyczna odpowiadająca za wyszukanie jednej oferty spełniających podane kryteria.
  Metoda przyjmuje argumenty:
    - idx – id oferty
    - by – tablica parametrów, po których można filtrować wyszukiwane oferty

  Metoda zwraca model oferty.
- **all()** – metoda statyczna odpowiadająca za zwrócenie wszystkich ofert, które znajdują się w bazie danych.

2.7. Opis metod **JobController**

- **print_jobs()** – metoda odpowiadająca za wyświetlenie zleconych już zadań.
- **add_custom_job()** – metoda odpowiadająca za zlecenie nowych zadań scrapowania.
  Metoda zwraca informacje o zleconym zadaniu.
- **pause()** – metoda odpowiadająca za wstrzymanie wykonywania zadań.
- **resume()** – metoda odpowiadająca za wznowienie wykonywania zadań.
- **job(idx)** – metoda odpowiadająca wyświetlenie informacji na temat jednej konkretnego zadania.
  Metoda przyjmuje argument:
    - idx – id zadania

2.8. Opis metod **SchedulerService**

- **create_job(name, func, args, seconds, minutes)** – metoda odpowiadająca za utworzenie oraz dodanie zadania do harmonogramu.
  Metoda przyjmuje argumenty:
  - name – nazwa
  - func – funkcja, która zostanie uruchomiona zgodnie z hamronogranem.
  - args – tablica argumentów, która zostanie przekazana do funkcji w momencie uruchomienia zadania.
  - seconds – liczba sekund po otrzymaniu zlecenia utworzenia zadań używana do obliczenia za ile zadanie ma zostać wykonane. Domyślnie 0.
  - minutes – liczba minut po otrzymaniu zlecenia utworzenia zadań używana do obliczenia za ile zadanie ma zostać wykonane. Domyślnie 0.

  Metoda zwraca zlecone zadanie.

7. Trudności w implementacji

Największą trudnością podczas implementacji było zapewnienie rozszerzalności o nowe źródła danych. Aby to osiągnąć postawiliśmy na użycie pliku w formacie json.

Plik ten oprócz nazwy serwisu, z którego sczytywane będą dane, zawiera podział na obiekty podstron “main_page” oraz “detail_page”. Każda z podstron zawiera lokalizatory wybranych elementów serwisu.

Przykładowy plik:

```
uni_dict = {
    "service_name": "justjoin.it",
    "main_page": {
        "url": "https://justjoin.it/offers",
        "list_type": "inf",
        "offer_list": {
            "by": "class_name",
            "locator": "locator"
        },
        "offer": {
            "by": "tag_name",
            "locator": "a"
        },
        "offer_title": {
```

```
            "by": "tag_name",
            "locator": "a"
        },
        "offer_url": {
            "by": "tag_name",
            "locator": "a"
        }
    },
    "detail_page": {
        "offer_title": {
            "by": "tag_name",
            "locator": "a"
        },
        "offer_technology": {
            "by": "tag_name",
            "locator": "a"
        },
        "offer_salary": {
            "by": "tag_name",
            "locator": "a"
        },
        "offer_salary_currency": {
            "by": "xpath",
            "locator": '//*[@id="job-sidebar"]/div[1]/div[2]'
        },
        "offer_location": {
            "by": "tag_name",
            "locator": "a"
        },
        "offer_experience": {
            "by": "tag_name",
            "locator": "a"
        },
        "offer_employment": {
            "by": "tag_name",
            "locator": "a"
        },
        "offer_description": {
            "by": "tag_name",
            "locator": "a"
        },
        "company_logo": {
            "by": "tag_name",
            "locator": "a"
        },
        "company_name": {
            "by": "tag_name",
            "locator": "a"
        },
    }
}
```

Kolejnym problemem było rozdzielenie stron używających listy dynamicznej i stron wykorzystujących paginacje. Aby to osiągnąć dodaliśmy w pliku json pole “list_type” które zmienia działanie scrapera w zależności od wartości pola.

W przypadku wartości “inf” scraper przewija listę ogłoszeń sczytując unikalne oferty.

W przypadku wartości “pagination” scraper sczytuje oferty z każdej kolejnej strony.

Sczytywanie odbywa się do momentu, gdy liczba sczytanych ofert osiągnie wyznaczoną liczbę lub scraper nie znajdzie kolejnych ofert.

8. Instrukcja użytkowania aplikacji

Użytkowanie aplikacji odbywa się poprzez komunikację sieciową protokołem HTTP. Wydawanie komend serwerowi API odbywa się komunikatami HTTP GET lub HTTP POST z odpowiednimi parametrami.

Aby zlecić zadanie scrapowania przesyłamy komunikat HTTP POST na adres **adres_serwera_ aplikacji/add** w ciele zapytania przesyłamy strukturę strony, która ma być zescrapowna.

W celu zdobycia informacji o zleceniach przesyłamy komunikat HTTP GET na **adres_serwera_ aplikacji/jobs**.

Aby otrzymać informacje na temat konkretnego zlecenia przesyłamy komunikat HTTP GET na **adres_serwera_ aplikacji/jobs/<idx>**.

Ustalony harmonogram możemy wstrzymać przesyłając komunikat HTTP GET na **adres_serwera_aplikacji/pause** możemy również harmonogram wznowić przesyłając komunikat HTTP GET na **adres_serwera_ aplikacji/resume**.