Łukasz Niewiński,
Rekrutacja Semantive, zadanie rekrutacyjne 14.02-23.02.2020

Komentarz rozwiązania:

Aplikacja w całości oparta na języku *python*, struktura jest oparta o *docker-compose* oraz komunikujące się ze sobą kontenery. Aplikacja jest dostępna dla użytkownika przeglądarkowego oraz bezpośrednio poprzez API(endpoits opisane w README). Obecna funkcjonalność ogranicza się do pobierania oraz przechowywania tekstu pobranego ze stron internetowych. Do uruchomienia aplikacji wystarczy pojedyncza komenda. Oprócz adresu url strony docelowej, z której mają być pobrane informacje, aplikacja na wejściu przyjmuje 2 informacje typu True/False, które decydują o pobraniu tekstu, obrazów lub obydwu.

Przykładowe requesty do przetestowania API:

POST *http://localhost:5000/api/webpages*

[{"url_path": "https://pl.wikipedia.org/wiki/Lizbona",
"retrieved_text": "true",
"retrieved_img": "true"}]

GET http://localhost:5000/api/webpages

GET http://localhost:5000/api/text?Identifier=1

Architektura:

Utworzone kontenery to:

- ***webpage*** - główny system aplikacji, umożliwia dostęp poprzez przeglądarkę internetową(` *http://localhost:5000/`)* oraz API.
- ***text_retrieve*** – kontener odpowiedzialny za pobieranie tekstu ze stron internetowych. Na wejściu(poprzez wewnętrzną sieć) przyjmuje adres *url_path* oraz zwraca pobrany teskt.
- ***img_retrieve*** – kontener odpowiedzialny za pobieranie obrazów ze stron internetowych. Do pobierania obrazów wykorzystuję Scrapiego, który uploaduje obrazy do chmury AWS oraz zwraca ścieżki, które są przechowywane w bazie danych.
- ***db*** – baza danych posgresql. Pojedyńcza baza danych uruchamiana wraz ze startem aplikacji(dołączyłem entrypoint do automatycznego tworzenia tabel). Przyjąłem założenie, że każda ze stron internetowych może być pobrana więcej niż raz(w różnych odstępach czasu ta sama strona może być inna), wobec czego każdy request o pobranie danych otrzymuje swój liczbowy identyfikator(primary_key) oraz czas zgłoszenia(*created_at*). Kombinacja adresu url strony oraz czas zgłoszenia mogą pełnić rolę unikalnego identyfikatora – przydzielenie liczbowego identyfikatora było prostsze. Przy przygotowywaniu wersję na produkcję zastąpiłbym wewnętrzną bazę danych zewnętrznym serwerem.

Funkcjonalności:

- ***Pobieranie tekstu*** - Tekst pobierany za pomocą biblioteki *html2text*, alternatywne rozwiązania obejmowałyby wykorzystanie *Scrapiego* lub *BeatifulSoup*.
- ***Pobieranie obrazów*** – Zdecydowałem się na przechowywanie obrazów w chmurze, ponieważ jest to rozwiązanie najbardziej skalowalne i wydaje się być najefektywniejsze. Alternatywnym rozwiązaniem byłoby przechowywanie obrazów lokalnie oraz udostępnianie ich poprzez ngnixa jako staticfiles, rozwiązanie te mogłoby się sprawdzić przy małym obciążeniu aplikacji. Przesyłanie zdjęć powinno być strumieniowe. Ewentualnie można by przechowywać obrazy w bazie danych(type BLOB) jednak serwowanie dużych plików z aplikacji może być problematyczne. Początkowo próbowałem napisać własną funkcję do pobierania obrazów z wykorzystaniem BeaufilSoup, jednakże wiązałoby się to z mniejszą efektywnością przy pobieraniu obrazów z różnych stron – wymagane byłoby zaadresowanie problemu różnorodnego tagowania i różnego serwowania zdjęć na poszczególnych stronach. Napotkałem duże trudności przy uruchomieniu pakietu scrapy wewnątrz kontenera. Obecnie pobieranie zdjęć nie funkcjonuje.
- ***Automatyzacja urochomienia*** - `docker-compose up -d --build`
- ***Przechowywanie danych*** - Do przechowywania danych wykorzystuję trzy tabele relacyjne, które można ograniczyć do 1 lub 2. Obecnie istniejące tabele to: *WebpageRetrieved*(przechowuje podstawowe informacje o przyjętych requestach), *TextRetrieved*(przechowuje id zgłoszenia i pobrany tekst) oraz tabela *ImgRetrieved*(przechowuje id zgłoszenia oraz ścieżki do pobranych obrazów przechowywanych w chmurze). Dołączyłem również zewnętrzy volume pod bazę danych, który pozwala na zachowanie danych przy usunięciu obrazów. Obrazy są przechowywane w s3.
- ***Testy*** – Nie zdążyłem przygotować testów. W ramach testowania aplikacji napisałbym dodatkowy scrypt, który wysyłałby requesta, sprawdzał odpowiedź a następnie weryfikował zgodność danych poprzez dodatkowy request o konkretne dane text/img.
- ***Sprawdzanie statusu*** – Niektóre zlecenia mogą zajmować więcej czasu, potrzebne jest wprowadzenie asynchroniczności oraz kolejek(np. job queue). Rozwiązaniem mogłoby być wykorzystanie biblioteki *cellery*. Tutaj również nie zdążyłem zaimplementować tej funkcjonalności.

Rozwijanie aplikacji:

- **Uwierzytelnianie** - należy dodać uwierzytelnianie zarówno przez API jak i stronę internetową.
- **Transakcje –** zapisywanie do bazy danych można wykonywać w charakterze transakcji.
- **Wielowątkowość –** dodanie wielowątkowości zwiększyłoby responsywność aplikacji.
- **Production environments –** przygotowanie środowiska produkcyjnego z wykorzystaniem gunicorna.
- **Errors handling –** usprawniłbym walidację oraz wyłapywanie poszczególnych błędów poprzez try/except.

Podsumowanie:

Czerpałem dużo radości z pracy przy projekcie, sporo też się nauczyłem. Z chęcią przyjmę uwagi co do słabych stron proponowanego przeze mnie rozwiązania oraz o tym jak można usprawnić aplikację.