Raport z projektu: System rozproszonego scrapera stron internetowych

Autorzy: Adrian Witów 21319, Czyżewska Magdalena 21227
Grupa: 3
Data: 11.05.2025
Repozytorium GitHub:
https://github.com/cytruseqq/SCRAPING_PROJECT/tree/main/web-scraper-project

Cel projektu

Celem projektu było stworzenie systemu do scrapowania stron internetowych, działającego w architekturze rozproszonej. Użytkownik za pomocą interfejsu webowego wprowadza adresy URL stron, które mają zostać przetworzone. System następnie asynchronicznie pobiera dane z tych stron i zapisuje je do bazy danych MongoDB.

Projekt miał na celu praktyczne zastosowanie technologii backendowych i konteneryzacji. Istotnym wymaganiem było również rozdzielenie aplikacji na moduły działające jako oddzielne kontenery Docker: frontend/backend, silnik scrapujący oraz baza danych.

Opis działania

Użytkownik uruchamia aplikację i za pomocą formularza dostępnego na stronie webowej wpisuje adresy URL. Po kliknięciu przycisku uruchomienia scrapowania dane są przesyłane do backendu opartego o Flask, który przekazuje je dalej do silnika scrapującego. Silnik działa jako oddzielny proces (uruchamiany z poziomu backendu) i korzysta z asyncio oraz multiprocessingu, aby równolegle przetwarzać wiele adresów.

Po pobraniu danych ze stron (takich jak tytuł, treść i link), wyniki zapisywane są do bazy danych MongoDB. W przypadku serwisów typu aktualności, system potrafi przetworzyć wiele artykułów naraz.

Struktura systemu

Projekt został podzielony na trzy główne moduły:

1. **Interfejs użytkownika (Flask + HTML/CSS)** – odpowiada za przyjmowanie adresów URL oraz komunikację z użytkownikiem.
2. **Silnik scrapujący** – samodzielny proces działający w tle, który obsługuje asynchroniczne scrapowanie treści.
3. **Baza danych (MongoDB)** – przechowuje dane w formacie dokumentowym, co dobrze sprawdza się przy różnorodnych danych tekstowych.

Całość uruchamiana jest jako kontenery za pomocą Docker Compose.

Technologie użyte w projekcie

Projekt wykorzystuje następujące technologie:

- Python 3 (Flask, aiohttp, BeautifulSoup, multiprocessing)
- HTML i CSS (prosty frontend)
- MongoDB (jako baza danych)
- Docker i Docker Compose (uruchamianie aplikacji w kontenerach)

Instalacja i uruchomienie

Aby uruchomić projekt, należy sklonować repozytorium:

```
git clone https://github.com/cytruseqq/SCRAPING_PROJECT.git
cd SCRAPING_PROJECT/web-scraper-project
```

Następnie można uruchomić kontenery za pomocą polecenia:

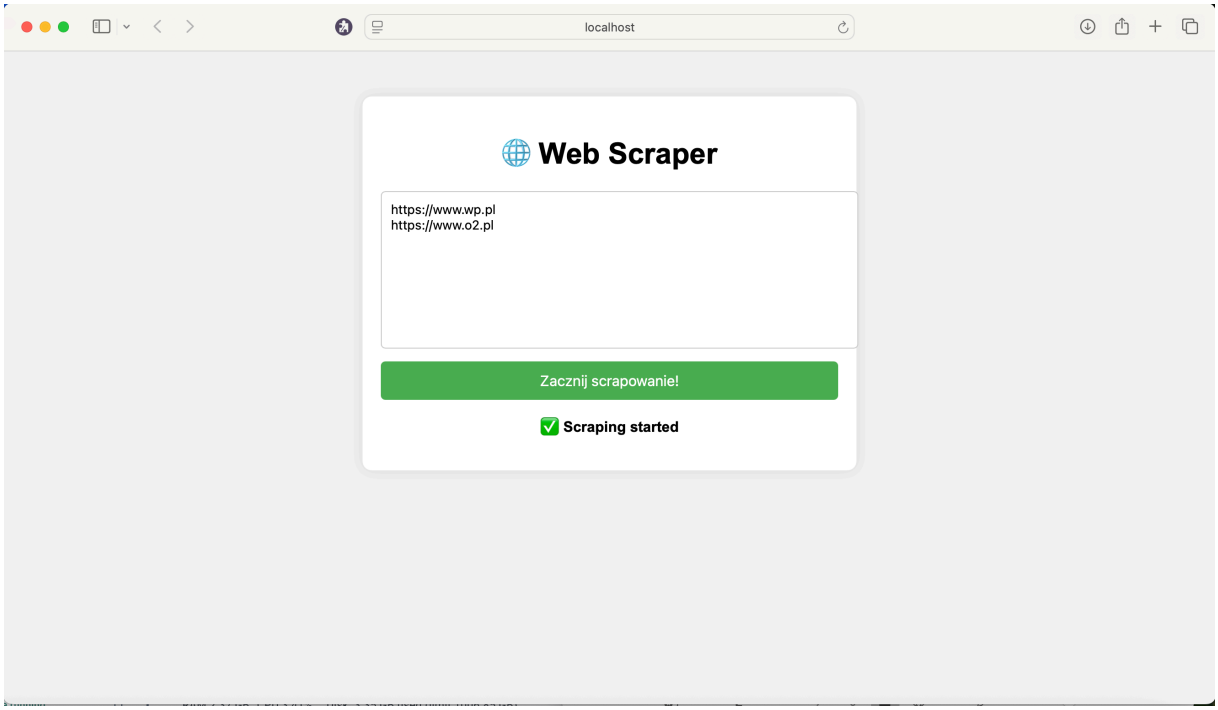
```
docker-compose up --build
```

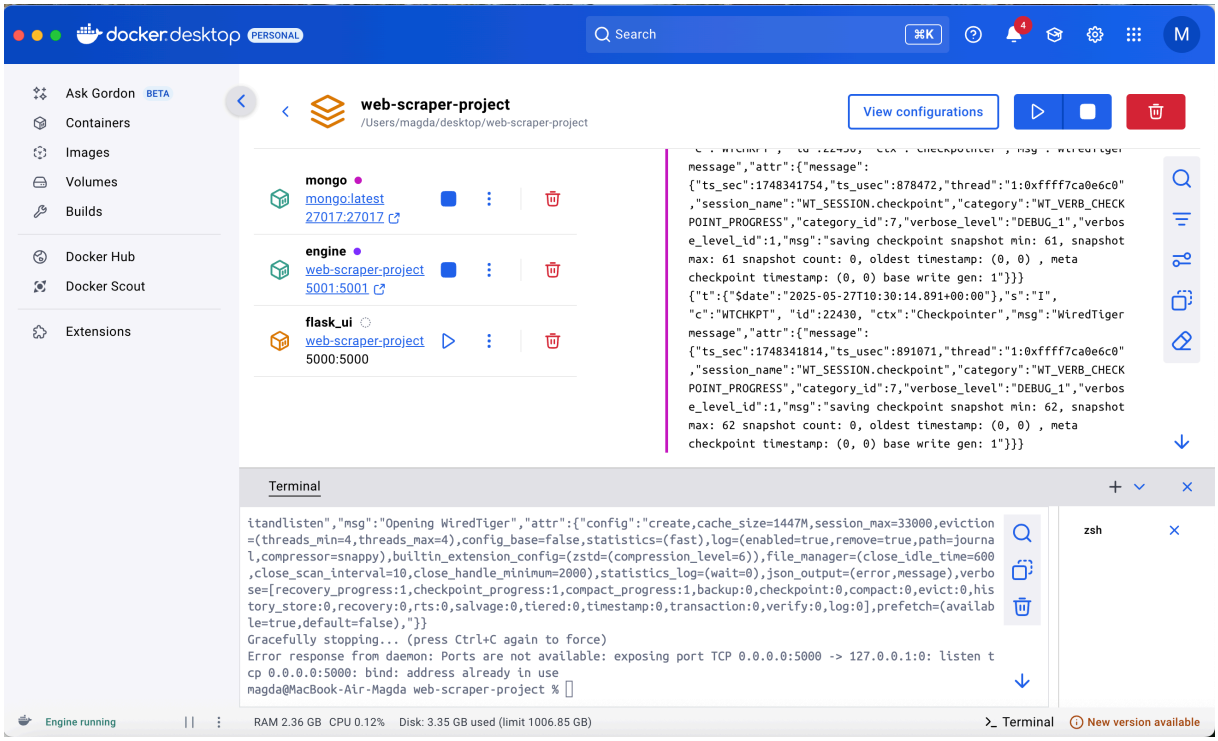
Po uruchomieniu aplikacja jest dostępna pod adresem:

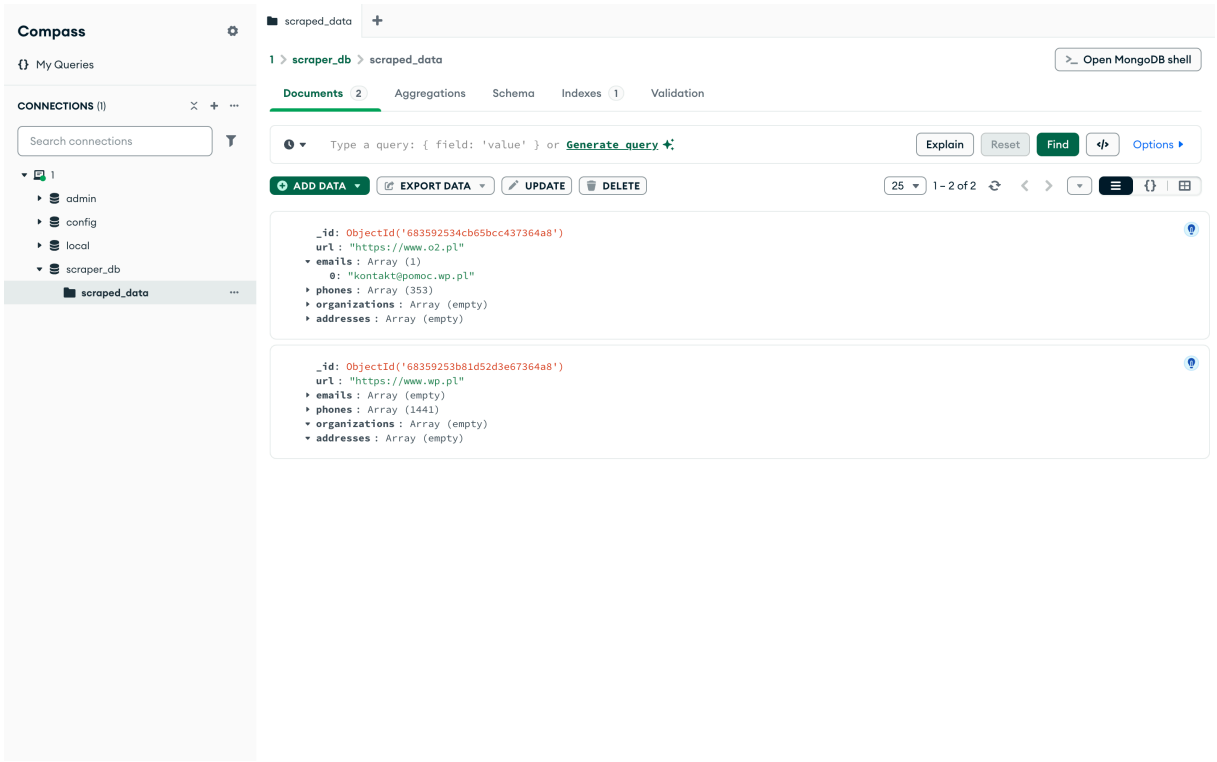
```
http://localhost:5001
```

W przeglądarce pojawi się formularz, do którego można wkleić jeden lub więcej adresów URL. Po kliknięciu przycisku „Start Scraping” rozpocznie się pobieranie danych, a komunikat „OK – scraping started” poinformuje o poprawnym rozpoczęciu działania silnika.

Dane będą sukcesywnie zapisywane do bazy danych MongoDB, w kolekcji `articles`.




docker desktop
PERSONAL
Search
Ask Gordon BETA
Containers
Images
Volumes
Builds
Docker Hub
Docker Scout
Extensions
web-scraper-project
/Users/magda/desktop/web-scraper-project
View configurations
mongo
mongo:latest
27017:27017
engine
web-scraper-project
5001:5001
flask_ui
web-scraper-project
5000:5000
Terminal
zsh
Gracefully stopping... (press Ctrl+C again to force)
Error response from daemon: Ports are not available: exposing port TCP 0.0.0.0:5000 -> 127.0.0.1:0: listen tcp 0.0.0.0:5000: bind: address already in use
magda@MacBook-Air-Magda web-scraper-project %
Engine running
RAM 2.36 GB CPU 0.12% Disk: 3.35 GB used (limit 1006.85 GB)
Terminal
New version available


Compass
My Queries
CONNECTIONS (1)
Search connections
admin
config
local
scraper_db
scraped_data
scraped_data
1 > scraper_db > scraped_data
Open MongoDB shell
Documents 2
Aggregations
Schema
Indexes 1
Validation
Type a query: { field: 'value' } or Generate query
Explain
Reset
Find
Options
ADD DATA
EXPORT DATA
UPDATE
DELETE
25
1 - 2 of 2
_id: ObjectId('683592534cb65bcc437364a8')
url : "https://www.o2.pl"
emails : Array (1)
0: "kontakt@pomoc.wp.pl"
phones : Array (353)
organizations : Array (empty)
addresses : Array (empty)
_id: ObjectId('68359253b81d52d3e67364a8')
url : "https://www.wp.pl"
emails : Array (empty)
phones : Array (1441)
organizations : Array (empty)
addresses : Array (empty)

Architektura aplikacji

Moduły zostały rozdzielone z myślą o łatwej skalowalności oraz logicznym podziale odpowiedzialności. Interfejs użytkownika działa niezależnie od silnika scrapującego, dzięki czemu możliwe jest wdrażanie zmian w UI bez ingerencji w backend. Silnik działa w osobnym procesie i może być z łatwością skalowany w przyszłości, np. przez uruchamianie wielu instancji w osobnych kontenerach.

MongoDB została wybrana jako baza danych ze względu na elastyczność struktury danych oraz dobre wsparcie dla języka Python.

Testowanie

Projekt był testowany lokalnie na systemie Windows i MacOs z użyciem Docker Desktop. Poprawność działania MongoDB została sprawdzona za pomocą narzędzia MongoDB Compass. Dane zapisywały się poprawnie w odpowiednich kolekcjach. Proces scrapowania działał równolegle dla wielu adresów URL, nie powodując przeciążeń ani błędów.

Wnioski i dalszy rozwój

Projekt spełnia założone wymagania. Aplikacja działa stabilnie, zapisuje dane do bazy i jest uruchamiana w formie rozproszonej. Możliwe ścieżki rozwoju to:

- dodanie możliwości przeglądania wyników scrapowania w panelu webowym,
- automatyzacja scrapowania w tle w określonych odstępach czasu (scheduler),
- zastosowanie kolejki zadań (np. Redis + Celery),
- usprawnienie interfejsu użytkownika pod względem estetycznym i UX.

Projekt pokazał praktyczne zastosowanie konteneryzacji oraz asynchronicznego przetwarzania danych w Pythonie i może być solidną bazą pod bardziej zaawansowany system ETL (extract-transform-load) lub crawler.