

<b>AKADEMIA NAUK STOSOWANYCH</b> <b>INSTYTUT INFORMATYKI STOSOWANEJ IM. KRZYSZTOFA BRZESKIEGO</b> <b>Przetwarzanie równoległe i rozproszone</b>	
Sprawozdanie wykonał: Dominik Świerczyński Andrzej Posim	Tytuł ćwiczenia: „Web scraper”
Nr indeksu: 20486 20475	Data wykonania: 02.06.2024

1. Zakres prac nad aplikacją:

- Aplikacja pobiera, selekcjonuje i składa wybrane dane o narzuconym profilu z witryn internetowych.
- Profil danych jest ustalony przez realizującego projekt. Profil danych powinien obejmować min. 4 grupy, np. adresy email, adresy korespondencyjne, schemat organizacyjny itp.
- Program wykorzystuje wielowątkowość/wieloprocusowość. Silnik należy zrealizować we własnym zakresie wykorzystując: multiprocessing i asyncio. Przetwarzanie ma być wieloprocusowe, najlepiej z możliwością skalowania na rdzenie procesora, dalej na komputery, dalej na klastry itp.
- Do parsowania kontentu należy użyć BeautifulSoup.
- Dane mają być zapisywane w BD, np. MongoDB
- Program ma posiadać interfejs graficzny zrealizowany w Python (Flask lub Django)
- Docelowo aplikacja ma być rozproszona na min 3 moduły: interfejs (1 lub więcej kontenerów), silnik (1 kontener), BD (1 kontener). Sposób ulokowania należy opracować we własnym zakresie i potrafić uzasadnić wybory.

2. Aplikacja przeszukuje dwie popularne strony internetowe, CENEO i OLX, aby znaleźć i wyświetlić wybrane przez użytkownika produkty. Automatycznie przegląda te strony i zwraca poszukiwane kryteria takie jak:

- Nazwa produktu
- Cena produktu
- Zdjęcie produktu
- Bezpośredni link do produktu
- Opis pobierany bezpośrednio ze strony produktu

3. Dodatkowo, do zrealizowania interfejsu graficznego został użyty Flask a do parsowania danych używamy BeautifulSoup. Flask zapewnia nam prostą i elastyczną platformę do stworzenia naszej aplikacji. Umożliwiło to na łatwe i przejrzyste prezentowanie wyników

wyszukiwania naszego web scraper'a. BeautifulSoup pozwala na efektywne przetwarzanie i analizowanie zawartości stron internetowych, co pozwala na pobierania informacji z nich.

4. Link do kodu Git Hub

[https://github.com/swierczynski02/projekt\\_prir](https://github.com/swierczynski02/projekt_prir)

5. Aplikacja działa na czterech kontenerach, z których każdy pełni określoną rolę:

- a. Wyszukiwarka\_db\_1:
  - i. Odpowiada za poprawne działanie bazy danych.
  - ii. Obsługiwana przez MongoDB, która zapewnia przechowywanie i zarządzanie danymi produktów.
- b. Wyszukiwarka\_db\_service\_1:
  - i. Jego zadanie polega na zapisywaniu rekordów do bazy danych.
  - ii. Zapewnia integralność i spójność danych przechowywanych w MongoDB.
- c. Wyszukiwarka\_engine\_1:
  - i. Na tym kontenerze uruchomiony jest silnik web scraper'a.
  - ii. Odpowiada za przeszukiwanie stron internetowych CENEO i OLX oraz pobieranie danych o produktach.
- d. Wyszukiwarka\_ui\_1:
  - i. Odpowiada za wygląd i działanie interfejsu użytkownika.
  - ii. Zapewnia użytkownikom dostęp do funkcji aplikacji poprzez przejrzysty i intuicyjny interfejs webowy.

```
debian@Debian:~/wyszukiwarka$ docker compose ps --all
WARN[0000] /home/debian/wyszukiwarka/docker-compose.yml: `version` is obsolete
NAME                                IMAGE                                COMMAND                                SERVICE
wyszukiwarka_db_1                   mongo:4.4                           "docker-entrypoint.s..."           db
wyszukiwarka_db_service_1           wyszukiwarka_db_service              "python app.py"                      db_service
wyszukiwarka_engine_1               wyszukiwarka_engine                  "python engine.py"                   engine
wyszukiwarka_ui_1                   wyszukiwarka_ui                      "python app.py"                      ui
debian@Debian:~/wyszukiwarka$
```

Fot. nr 1

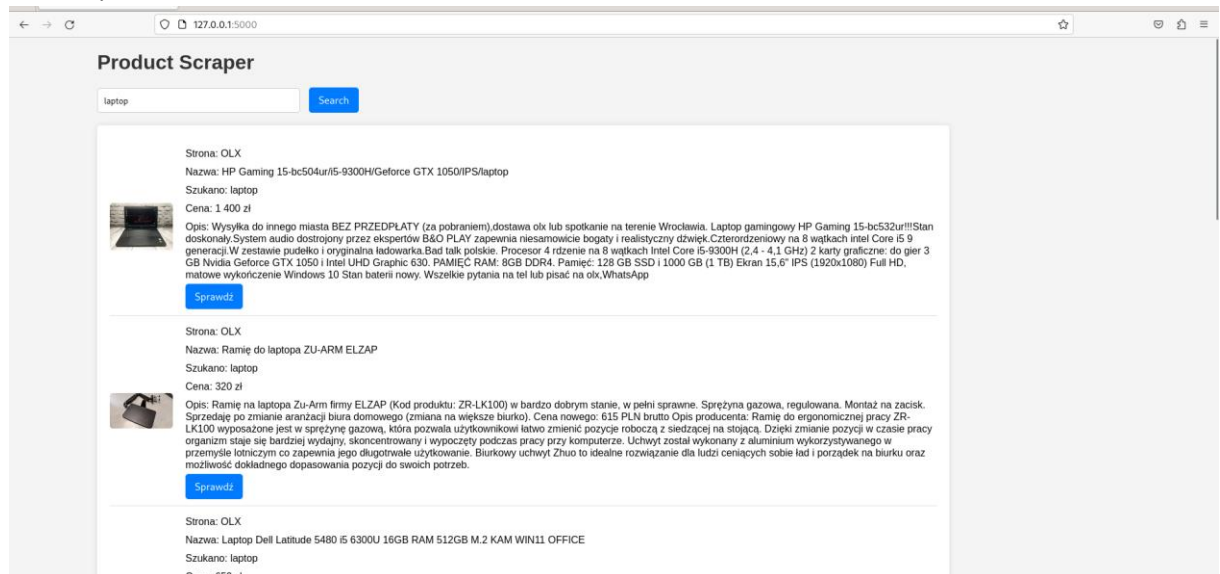
Fotografia nr 1 pokazuje, że kontenery zostały utworzone i działają poprawnie.

6. Funkcjonalność dodatkowa:

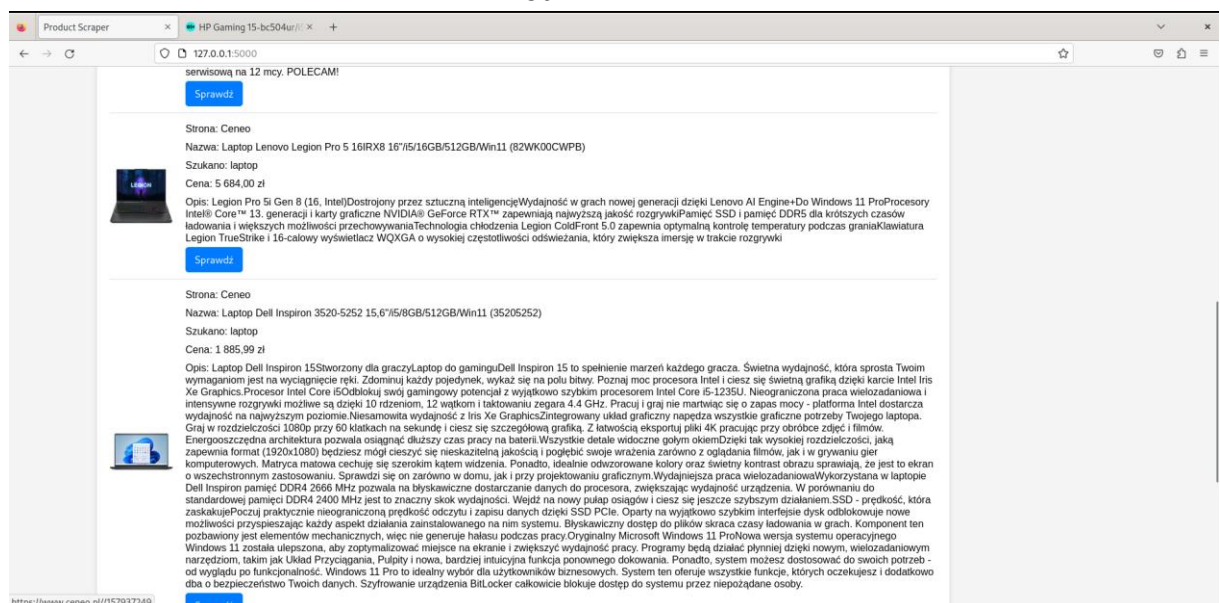
Web scraper po wyszukaniu produktów, przechodzi bezpośrednio do strony internetowej tego produktu. Następnie pobiera opis tam zamieszczony i wyświetla do na naszej stronie. Na fotografii nr 7 i 8 wydać, że web scraper pobiera opis ze strony OLX a następnie wyświetla nam na naszej stronie.

## 7. Zdjęcia z działania programu

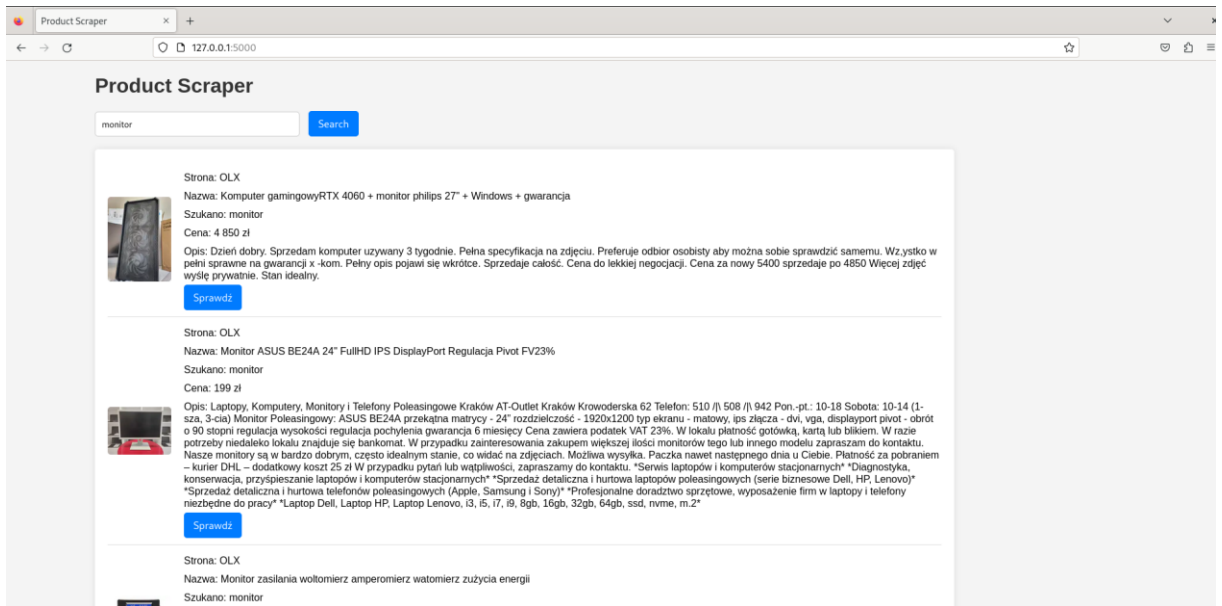
Fotografie od 2 do 6 pokazują przykładowe działanie naszego web scraper'a. Aplikacja pobiera nazwę, cenę, zdjęcie oraz opis dla 3 produktów z OLX i CENEO. Po kliknięciu przycisku „Sprawdź” automatycznie przechodzimy na stronę z danym produktem (fot. nr 7 i 8).



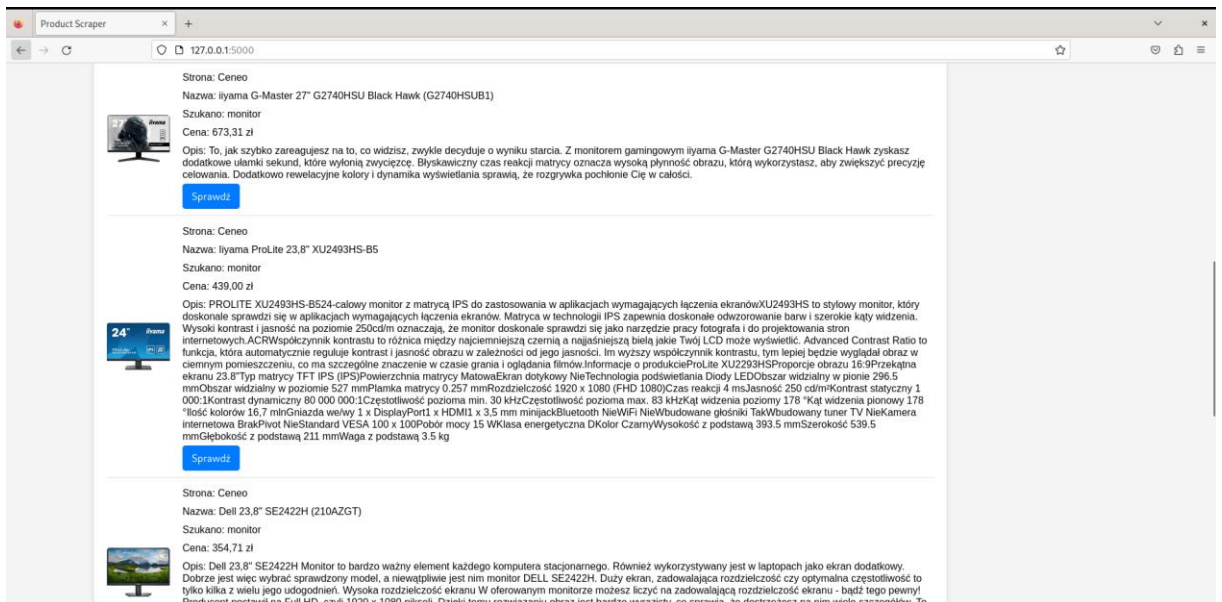
Fot. nr 2



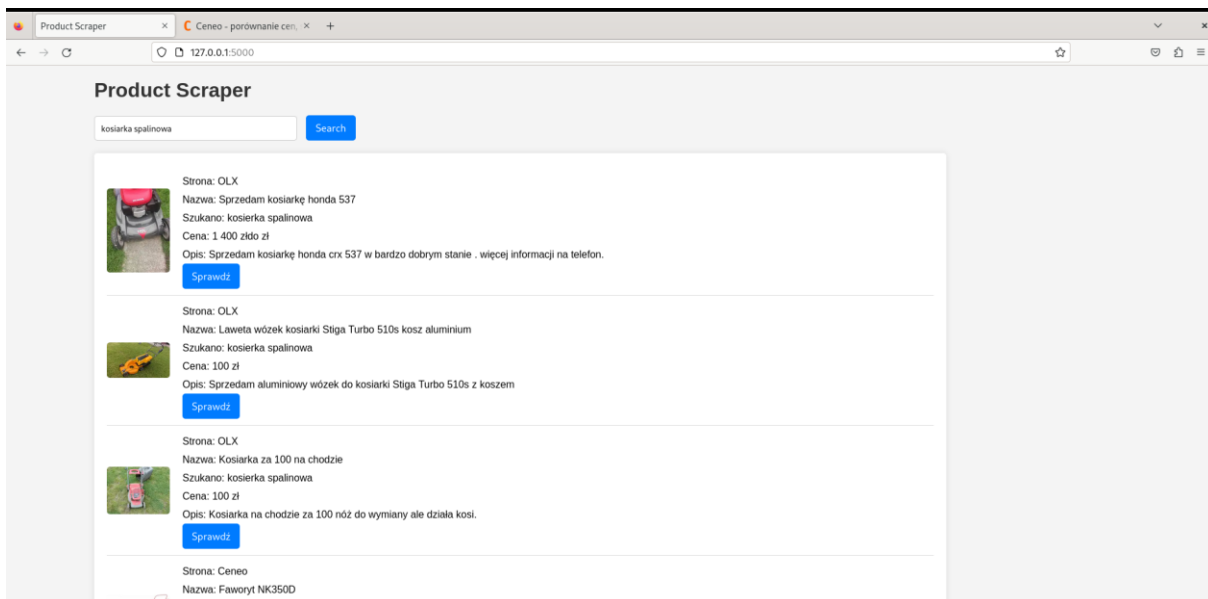
Fot. nr 3



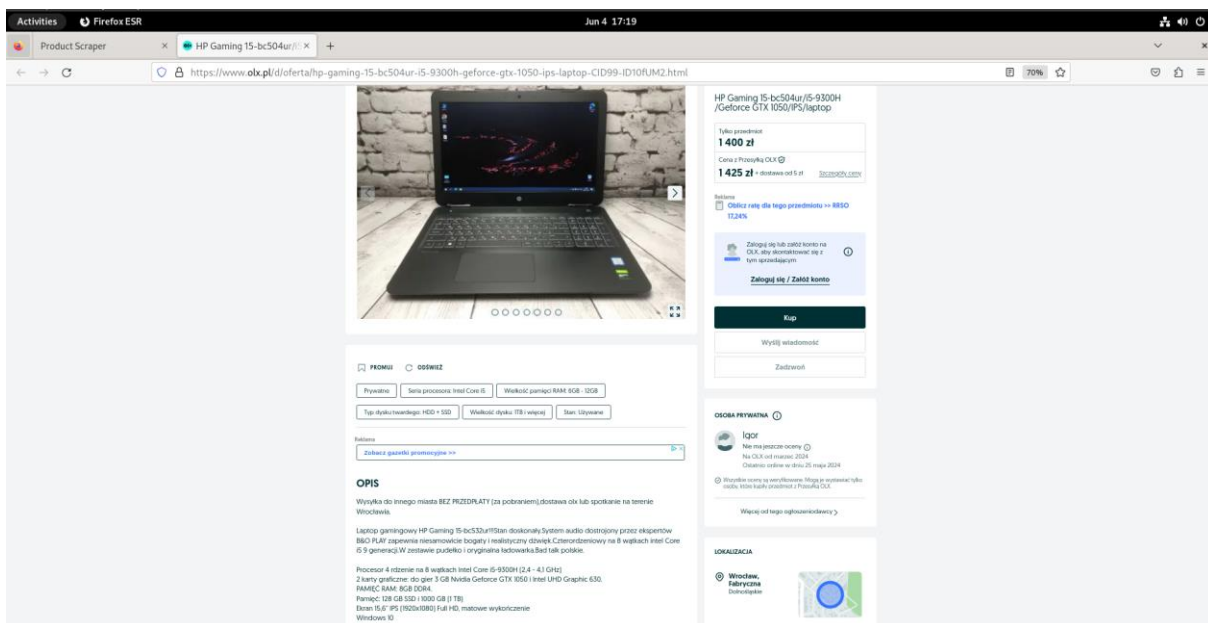
Fot nr 4



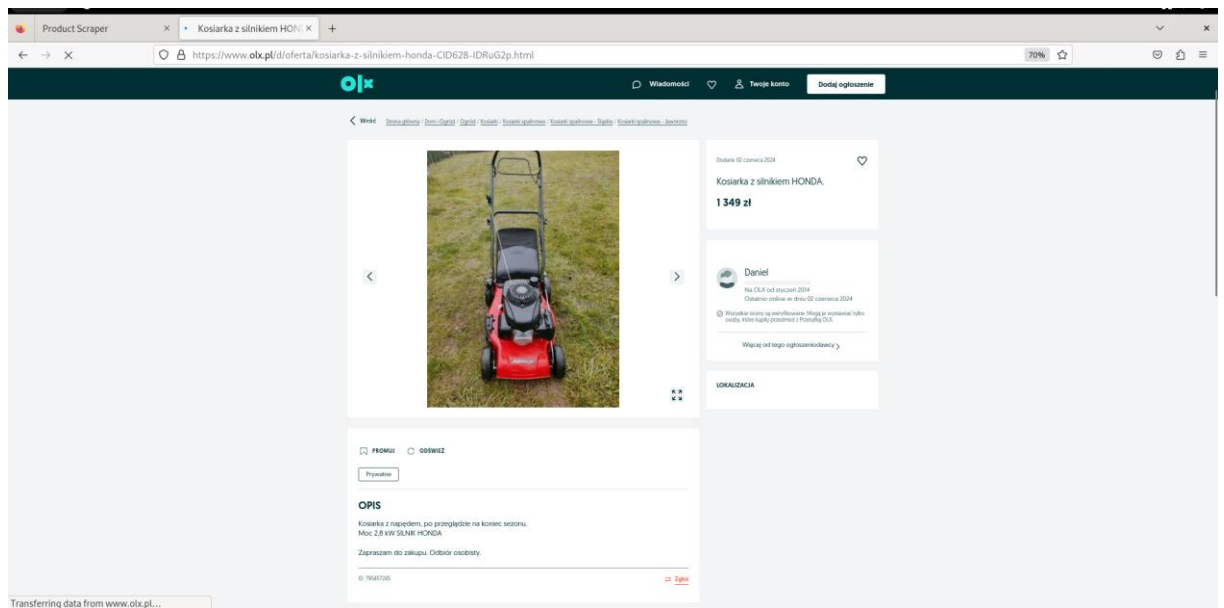
Fot. nr 5



Fot. nr 6



Fot. nr 7



Fot. nr 8

8. Zrzuty ekranu z kodem web scraper'a
  - a. Zawartość folderu „db\_service”
    - i. app.py

Open  app.py  
~/wyszukiwarka/db\_service

```
from flask import Flask, request, jsonify
from pymongo import MongoClient
from bson import ObjectId
import logging

app = Flask(__name__)

logging.basicConfig(level=logging.DEBUG)

# łączenie się z bazą danych
client = MongoClient('mongodb://db:27017/')
db = client.product_db
collection = db.products

#Funkcja do serializacji produktów
def serialize_product(product):
    if isinstance(product, dict):
        for key, value in product.items():
            if isinstance(value, ObjectId):
                product[key] = str(value)
        return product

@app.route('/products', methods=['POST'])

#Dodawanie nowych produktów do bazy danych
def add_product():
    product_data = request.get_json()
    logging.debug(f"Adding product to DB: {product_data}")
    collection.insert_one(product_data)
    return jsonify({'status': 'Product added'}), 201

#Pobieranie produktów z bazy danych
@app.route('/products', methods=['GET'])
def get_products():
    product_name = request.args.get('name')
    logging.debug(f"Fetching products from DB for name: {product_name}")
    products = collection.find({'name': {'$regex': product_name, '$options': 'i'}})
    serialized_products = [serialize_product(product) for product in products]
    return jsonify(serialized_products)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5002)
```

ii. Zawartość „Dockerfile”

```
Open ▾ + Dockerfile
~/wyszukiwarka/db_service

FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "app.py"]
```

iii. Zawartość „requirements.txt”

```
Open ▾ + requirements.txt
~/wyszukiwarka/db_service

Flask
pymongo
```

b. Zawartość folderu „engine”

i. Zawartość „engine.py”



```

import asyncio
import aiohttp
from bs4 import BeautifulSoup
from concurrent.futures import ThreadPoolExecutor
import requests
from flask import Flask, request, jsonify
import logging

app = Flask(__name__)

logging.basicConfig(level=logging.DEBUG)

#Funkcja do pobierania danych z linka (asynchroniczna)
async def fetch(session, url):
    logging.debug(f"Fetching URL: {url}")
    async with session.get(url) as response:
        logging.debug(f"Status code: {response.status}")
        if response.status != 200:
            logging.error(f"Error fetching URL: {url}, status code: {response.status}")
            return ""
        return await response.text()

#Asynchroniczna funkcja do pobierania danych o produkcie
async def fetch_product_data(product_name):
    urls = [f'https://www.olx.pl/oferty/q-{product_name}/',
            f'https://www.ceneo.pl/szukaj-{product_name}']
    # Lista URL-ów do przeszukania
    async with aiohttp.ClientSession() as session:
        tasks = [fetch(session, url) for url in urls]
        responses = await asyncio.gather(*tasks)

    products = []
    for response, url in zip(responses, urls):
        if response:
            logging.debug(f"Response length: {len(response)}")
            soup = BeautifulSoup(response, 'html.parser')
            # Parsowanie HTML do uzyskania danych o produkcie
            if "www.olx.pl" in url:
                try:
                    div_all = soup.find_all('div', {'class': 'css-1sw7q4x'})
                    i = 0
                    while(i<3):
                        div = div_all[i]
                        link = div.find('a', href=True)
                        cena = div.find('p', {'data-testid': 'ad-price'}).text
                        web_name = div.find('h6', {'class': 'css-16v5mdi er34gjf0'}).text
                        szukaj_link = 'https://www.olx.pl/' + link['href']
                        page = requests.get(szukaj_link)
                        soup2 = BeautifulSoup(page.content, 'html.parser')
                        opis_all = soup2.find('div', {'class': 'css-1t507yq er34gjf0'}).text

                        product_info = {
                            'name': product_name,
                            'price': cena.strip("qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM1aAzzżEe0ó5")+ ' zł',
                            'image': div.find('img')['src'],
                            'source': "OLX", # Dodaj URL do informacji o produkcie
                            'link': 'https://www.olx.pl/' + link['href'],
                            'web_name': web_name,

```

```

        'web_name': web_name,
        'opis': opis_all
    }
    logging.debug(f"Parsed product info: {product_info}")
    products.append(product_info)
    i=i+1
except Exception as e:
    logging.error(f"Error parsing HTML: {e}")
    continue
elif "www.ceneo.pl" in url:
    try:
        div_all = soup.find_all('div', {'class': 'cat-prod-row__body'})
        i=0
        while(i<3):
            div = div_all[i]
            cena = div.find('span', {'class': 'price'}).text
            link = div.find('a', href=True)
            szukaj_link = 'https://www.ceneo.pl/'+link['href']+'#tab=spec'
            page = requests.get(szukaj_link)
            soup2 = BeautifulSoup(page.content, 'html.parser')
            opis_all = soup2.find('div', {'class': 'lnd_content'}).text
            #opis = opis_all.find('p').text
            web_name = div.find('strong', {'class': 'cat-prod-row__name'}).text
            product_info = {
                'name': product_name,
                'price': cena.strip("qwertuyiopasdfghiklzxvbnmQWERTYUIOPASDFGHJKLZXCVBNM!@AżżżeFóóś$")+ ' zł',
                'image': div.find('img')['src'],
                'source': "Ceneo", # Dodaj URL do informacji o produkcie
                'link': 'https://www.ceneo.pl/'+link['href'],
                'web_name': web_name,
                'opis': opis_all
            }
            logging.debug(f"Parsed product info: {product_info}")
            products.append(product_info)
            i=i+1
        except Exception as e:
            logging.error(f"Error parsing HTML: {e}")
            continue
    return products

#Funkcja do zapisywania danych produktów do bazy danych
def save_to_db(products):
    db_url = 'http://db_service:5002/products'
    for product in products:
        logging.debug(f"Saving to DB: {product}")
        response = requests.post(db_url, json=product)
        logging.debug(f"DB response: {response.status_code}")

# Funkcja przetwarzająca produkty (pobieranie i zapisywanie)
def process_product(product_name):
    products = asyncio.run(fetch_product_data(product_name))
    save_to_db(products)

# Endpoint do wyszukiwania produktu
@app.route('/search', methods=['POST'])
def search_product():

    # Endpoint do wyszukiwania produktu
    @app.route('/search', methods=['POST'])
    def search_product():
        data = request.get_json()
        product_name = data['product_name']
        logging.debug(f"Starting search for product: {product_name}")


        with ThreadPoolExecutor() as executor:
            executor.submit(process_product, product_name)

        return jsonify({'status': 'Search started'})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5001)

```

## ii. Zawartość „Dockerfile”

Open ▾ 

Dockerfile  
~/wyszukiwarka/engine

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "engine.py"]
```

iii. Zawartość „requirements.txt”

Open ▾ 

requirements.txt  
~/wyszukiwarka/engine

```
aiohttp
beautifulsoup4
requests
flask
```

- c. Zawartość folderu „ui”
  - i. Zawartość „app.py”

```
Open ▾ + app.py
~/wyszukiwarka/ui

from flask import Flask, request, jsonify, render_template
import requests

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

# Endpoint do wyszukiwania produktu
@app.route('/search', methods=['POST'])
def search():
    product_name = request.form['product_name']
    response = requests.post('http://engine:5001/search', json={'product_name': product_name})
    return response.json()

# Endpoint do wyświetlania wyników wyszukiwania
@app.route('/results')
def results():
    product_name = request.args.get('product_name')
    response = requests.get(f'http://db_service:5002/products?name={product_name}')
    return jsonify(response.json())

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

ii. Zawartość „Dockerfile”

```
Open ▾ + Dockerfile
~/wyszukiwarka/ui

FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "app.py"]
```

iii. Zawartość „requirements.txt”

```
Open ▾ + requirements.txt
~/wyszukiwarka/ui

Flask
requests
```

iv. Zawartość „index.html”

```
Open ▾  index.html
~/wyszukiwarka/ui/templates

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Product Scraper</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
  <div id="container">
    <h1>Product Scraper</h1>
    <form id="searchForm">
      <input type="text" name="product_name" placeholder="Enter product name">
      <button type="submit">Search</button>
    </form>
    <div id="results"></div>
  </div>
  <script>
    document.getElementById('searchForm').addEventListener('submit', async function(event) {
      event.preventDefault();
      const form = event.target;
      const data = new FormData(form);
      const response = await fetch('/search', {
        method: 'POST',
        body: data
      });
      const result = await response.json();
      if (result.status === 'Search started') {
        setTimeout(async () => {
          const searchResults = await fetch('/results?product_name=${data.get('product_name')}');
          const searchResultsJson = await searchResults.json();
          const resultsDiv = document.getElementById('results');
          resultsDiv.innerHTML = ''; // Wyczyść wyniki przed wyświetleniem nowych
          searchResultsJson.forEach(product => {
            const productDiv = document.createElement('div');
            productDiv.className = 'product';
            productDiv.innerHTML = `
              
              <div>
                <p>Strona: ${product.source}</p>
                <p>Nazwa: ${product.web_name}</p>
                <p>Szukano: ${product.name}</p>
                <p>Cena: ${product.price}</p>
                <p>Opis: ${product.opis}</p>
                <a href="${product.link}" target="_blank"><button>Sprawdź</button></a>
              </div>
            `;
            resultsDiv.appendChild(productDiv);
          });
        }, 1000); // Czekaj 1 sekund przed pobraniem wyników z bazy
      } else {
        document.getElementById('results').innerText = 'Search failed.';
      }
    });
  </script>
</body>
</html>
```

v. Zawartość pliku „styles.css”

```
body {
  font-family: Arial, sans-serif;
  background-color: #f4f4f4;
  margin: 0;
  padding: 0;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}

#container {
  height: 100%;
}

h1 {
  color: #333;
}

form {
  margin-bottom: 20px;
}

input[type="text"] {
  padding: 10px;
  width: 300px;
  border: 1px solid #ccc;
  border-radius: 5px;
  margin-right: 10px;
}

button {
  padding: 10px 15px;
  border: none;
  background-color: #007bff;
  color: white;
  font-size: 16px;
  border-radius: 5px;
  cursor: pointer;
}

button:hover {
  background-color: #0056b3;
}

#results {
  width: 80%;
  max-width: 1600px;
  margin-top: 20px;
  background-color: white;
  padding: 20px;
  border-radius: 5px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
```

```

#results .product {
  display: flex;
  align-items: center;
  border-bottom: 1px solid #ddd;
  padding: 10px 0;
}

#results .product:last-child {
  border-bottom: none;
}

#results .product img {
  max-width: 100px;
  border-radius: 5px;
  margin-right: 20px;
}

#results .product div {
  display: flex;
  flex-direction: column;
}

#results .product p {
  margin: 5px 0;
}

```

d. Zawartość pliku „docker-compose.yml”

```
version: '3'
services:
  ui:
    build: ./ui
    ports:
      - "5000:5000"

  engine:
    build: ./engine
    command: ["python", "engine.py"]
    depends_on:
      - db_service

  db:
    image: mongo:4.4
    ports:
      - "27017:27017"

  db_service:
    build: ./db_service
    ports:
      - "5002:5002"
    depends_on:
      - db
```