

Przetwarzanie równoległe i rozproszone

STAC [2024/2025 LATO]

(Smyk, Jótkowski) Projekt Semestralny

Oliwier Bogdański 21181

Kacper Szponar 21306

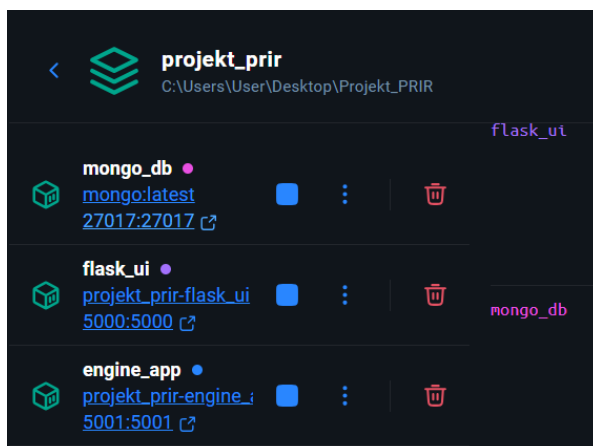
Link do dokumentacji Github: <https://github.com/Thiago1717/PRIR-Projekt>

Opis aplikacji

Aplikacja, której interfejs użytkownika został zrealizowany we frameworku Flask, służy do scrapowania danych o produktach z serwisu eBay. Użytkownik ma możliwość podania nazwy szukanego produktu, przedziału cenowego. Po zainicjowaniu wyszukiwania, aplikacja pozyskuje następujące dane dla każdej znalezionej oferty: tytuł ogłoszenia, cena (w formie tekstowej oryginalnej oraz przekonwertowanej na wartość numeryczną), koszt wysyłki, lokalizacja przedmiotu, link do oryginalnego ogłoszenia na eBay. Wszystkie zebrane dane są następnie składowane w bazie danych MongoDB, gdzie dla każdego zapytania tworzona jest dedykowana kolekcja. Architektura aplikacji została zaprojektowana jako system rozproszony, składający się z trzech odseparowanych modułów, działających w osobnych kontenerach Docker:

1. **Interfejs Użytkownika (flask_ui_container):** Komponent odpowiedzialny za interakcję z użytkownikiem i prezentację danych.
2. **Silnik Scrapujący (engine_container):** Zawiera logikę pobierania (aiohttp, asyncio) i przetwarzania (BeautifulSoup, multiprocessing) danych z eBay.
3. **Baza Danych (mongo_db_container):** Kontener z instancją MongoDB, służący do przechowywania wyników.

Zrzut ekranu przedstawiający kontenery.



Struktura projektu

```
Projekt_PRIR/
├── Dockerfile
├── docker-compose.yml

├── engine/
│   ├── Dockerfile
│   ├── requirements.txt
│   ├── engine_api.py
│   └── scraper.py

├── flask_app/
│   ├── Dockerfile
│   ├── requirements.txt
│   ├── app.py
│   ├── static/
│   │   └── css/
│   │       ├── style_index.css
│   │       └── style_results.css
│   └── templates/
│       ├── index.html
│       └── results.html
```

Struktura projektu opiera się na głównym katalogu Projekt_PRIR, zawierającym plik docker-compose.yml oraz dedykowane podfoldery dla każdego modułu: engine dla logiki silnika scrapującego oraz flask_app dla interfejsu użytkownika, przy czym każdy z tych modułów posiada własny Dockerfile i pliki źródłowe. W folderze flask_app znajdują się również podkatalogi static z plikami CSS oraz templates z szablonami HTML.

Opis Modułu silnika Scrapującego (engine)

scraper.py

Plik zawiera logikę odpowiedzialną za proces scrapowania danych. Implementuje mechanizmy asynchronicznego pobierania stron internetowych oraz wieloprosesowego prasowania treści HTML, w celu uzyskania informacji o produktach.

Konfiguracja globalna

```
MONGO_URI = os.environ.get("MONGO_URI", "mongodb://localhost:27017/")
DB_NAME = "scraper_db"

USER_AGENTS = [
    'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/115.0',
    'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36',
    'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36 Edg/125.0.0.0'
]
```

MONGO_URI: Adres URI połączenia z serwerem MongoDB. Odczytywany jest ze zmiennej środowiskowej MONGO_URI (przekazywanej przez docker-compose.yml), z wartością domyślną mongodb://localhost:27017/.

DB_NAME: Nazwa bazy danych w MongoDB, do której zapisywane są wyniki scrapowania (nazwa „scraper_db”).

USER_AGENTS: Lista przeglądarek internetowych, służy do rotacji nagłówka User-Agent w wysyłanych żądaniach HTTP, ma to na celu zmniejszenie prawdopodobieństwa wykrycia oraz zablokowania scrapera przez serwery eBay.

Funkcja run_ebay_scraper()

Główna funkcja uruchamiająca cały proces scrapowania dla danego zapytania. Inicjalizuje pętlę zdarzeń asyncio, tworzy pulę procesów ProcessPoolExecutor, a następnie wywołuje metodę scrape_and_save w celu pobrania, sprasowania oraz zapisania danych.

```
def run_ebay_scraper(query, min_price, max_price, sort_order):
    loop = asyncio.new_event_loop()
    asyncio.set_event_loop(loop)

    executor = ProcessPoolExecutor()

    scraper = EbayScraper(loop, executor)
    items = []
    message = "Scraping failed (initialization or unexpected error)"

    if scraper.db is None:
        message = "Database not available for scraper (connection failed during init)."
        scraper.close_connection()
        executor.shutdown(wait=False)
        loop.close()
        return items, message

    try:
        items, message = loop.run_until_complete(scraper.scrape_and_save(query, min_price, max_price, sort_order))
    except Exception as e:
        print(f"SCRAPER_FATAL (run_ebay_scraper): Krytyczny błąd wykonania: {e}")
        traceback.print_exc()
        message = f"Critical internal error during scraping: {str(e)}"
    finally:
        scraper.close_connection()
        executor.shutdown(wait=True)
        loop.close()

    return items, message
```

Metoda scrape_and_save

```
async def scrape_and_save(self, query, min_price, max_price, sort_order_str):
    if self.db is None: return [], "Database connection failed"

    request_params = [{"_nkw": query}]
    if min_price: request_params["_udlo"] = min_price
    if max_price: request_params["_udhi"] = max_price
    if sort_order_str == 'price_asc': request_params["_sop"] = "15"
    elif sort_order_str == 'price_desc': request_params["_sop"] = "2"

    pages_to_scrape_params = [request_params]
    all_parsed_items = []

    async with aiohttp.ClientSession() as session:
        fetch_tasks = [fetch_html_content(session, self.base_url, page_params, proxy_url=DEFAULT_PROXY_URL)
                        for page_params in pages_to_scrape_params]

        html_contents = await asyncio.gather(*fetch_tasks, return_exceptions=True)

        parse_tasks = []
        any_successful_fetch = False
        for content_or_exception in html_contents:
            if isinstance(content_or_exception, Exception) or content_or_exception is None:
                continue
            any_successful_fetch = True
            parse_tasks.append(parse_ebay_page_content_multiproc(content_or_exception, query, self.loop, self.executor))

        if not any_successful_fetch:
            return [], "Failed to fetch any content from eBay (all attempts failed)."

        if parse_tasks:
            list_of_item_lists = await asyncio.gather(*parse_tasks)
            for item_list_from_page in list_of_item_lists:
                all_parsed_items.extend(item_list_from_page)

    if parse_tasks:
        list_of_item_lists = await asyncio.gather(*parse_tasks)
        for item_list_from_page in list_of_item_lists:
            all_parsed_items.extend(item_list_from_page)

    if not all_parsed_items:
        if any_successful_fetch:
            return [], "No items found or parsed from successfully fetched eBay pages."
        return [], "Failed to fetch any content from eBay or no items found."

    collection_name = self._sanitize_collection_name(query)
    collection = self.db[collection_name]
    try:
        if all_parsed_items:
            for item_data in all_parsed_items:
                collection.update_one({"_id": item_data["_id"]}, {"$set": item_data}, upsert=True)
    except Exception as e_db:
        print(f"SCRAPER_ERROR: Błąd zapisu do DB (kolekcja: {collection_name}): {e_db}")
        return all_parsed_items, f"Scraping completed but DB write error: {e_db}"

    return all_parsed_items, "Scraping completed successfully"
```

Metoda odpowiada za proces pobierania danych (wywołując `fetch_html_content`) oraz ich prasowania (wywołując `parse_ebay_page_content_multiproc`), oraz odpowiada za ich zapis do bazy MongoDB.

Funkcja fetch_html_content()

```
async def fetch_html_content(session, url, params):
    """Asynchronicznie pobiera zawartość HTML strony, używając losowego User-Agenta."""
    selected_user_agent = random.choice(USER_AGENTS)
    headers = {
        'User-Agent': selected_user_agent,
        'Accept-Language': 'en-US,en;q=0.9,pl;q=0.8',
        'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
    }
    try:
        async with session.get(url, params=params, headers=headers, timeout=aiohttp.ClientTimeout(total=30)) as response:
            response.raise_for_status()
            return await response.text()
```

Funkcja odpowiedzialna jest za pobranie zawartości HTML z podanego URL przy użyciu biblioteki aiohttp. Implementuję rotację nagłówków User-Agent.

Funkcja parse_single_item_html()

```
def parse_single_item_html(item_html_str, query_for_item):
    def _extract_price_static(price_str_val):
        if not price_str_val: return None
        price_str_cleaned = re.sub(r'(\bUSD\b|\bPLN\b|\bEUR\b)', '', price_str_val, flags=re.IGNORECASE).replace(',', '.')
        try:
            cleaned_price_num_str = re.sub(r'[^0-9.]', '', price_str_cleaned)
            if cleaned_price_num_str.count('.') > 1:
                parts = cleaned_price_num_str.split('.')
                cleaned_price_num_str = "".join(parts[:-1]) + "." + parts[-1]
            return float(cleaned_price_num_str) if cleaned_price_num_str else None
        except ValueError: return None

    item_soup = BeautifulSoup(item_html_str, 'html.parser')
    title_tag = item_soup.select_one(".s-item_title span[role='heading'], .s-item_title")
    title_text = title_tag.get_text(strip=True) if title_tag else None
    if title_text and "Shop on eBay" in title_text: return None
    price_tag = item_soup.select_one(".s-item_price")
    price_text = price_tag.get_text(strip=True) if price_tag else None
    link_tag = item_soup.select_one(".s-item_link")
    link_href = link_tag["href"] if link_tag and link_tag.get("href") else None
    shipping_tag = item_soup.select_one(".s-item_shipping, .s-item_logisticsCost")
    shipping_text = shipping_tag.get_text(strip=True) if shipping_tag else "N/A"
    location_tag = item_soup.select_one(".s-item_location")
    location_text = location_tag.get_text(strip=True) if location_tag else "Nieznana"

    if title_text and price_text and link_href:
        price_value = _extract_price_static(price_text)
        item_id = hashlib.md5(f"{title_text}{link_href}".encode('utf-8')).hexdigest()
        return {"_id": item_id, "title": title_text, "price_text": price_text, "price_value": price_value,
                "shipping_info": shipping_text, "location": location_text, "link": link_href, "query_source": query_for_item}
    return None
```

Funkcja uruchamiana jest w osobnych procesach (za pomocą ProcessPoolExecutor) w celu sprasowania fragmentu HTML odpowiadającego pojedynczej ofercie. Wykorzystuję BeautifulSoup do pozyskania szczegółowych danych takich jak tytuł, cena, link, a następnie zwraca je w postaci słownika.

Funkcja `get_mongo_client_and_db()`

```
def get_mongo_client_and_db():  
    try:  
        client = MongoClient(MONGO_URI, server_api=server_api.ServerApi('1'), serverSelectionTimeoutMS=5000)  
        client.admin.command('ping')  
        db = client[DB_NAME]  
        return client, db  
    except Exception as e:  
        print(f"BŁĄD_SCRAPERA: Błąd połączenia/operacji MongoDB: {e}")  
        return None, None
```

Funkcja jest odpowiedzialna za nawiązanie połączenia z serwerem MongoDB, testuje połączenie komendą ping a następnie w przypadku niepowodzenia zwraca komunikat błędu.

Metoda `__init__`

```
class EbayScrapper:  
    def __init__(self, loop, executor):  
        self.base_url = "https://www.ebay.com/sch/i.html"  
        self.client, self.db = get_mongo_client_and_db()  
        self.loop = loop  
        self.executor = executor  
        if self.db is None:  
            pass
```

Funkcja (`__init__`) uruchamia się za każdym razem, gdy tworzony jest nowy obiekt. Jej zadaniem jest przygotowanie scrapera do pracy: ustawia adres strony, łączy się z bazą oraz zapamiętuje narzędzia (loop oraz executor) potrzebne do wydajnego działania programu.

Metoda `sanitize_collection_name()`

```
def _sanitize_collection_name(self, name_str):  
    if not name_str: return "default_ebay_collection"  
    name = name_str.replace(" ", "_").lower()  
    name = re.sub(r'^a-zA-Z0-9_-]', '', name)  
    return name[:100] if name else "default_ebay_collection"
```

Zadaniem funkcji jest to, aby zapisywane wyniki w bazie danych były zawsze jednolite. Zawsze zapisywane są one z małych liter a spacja zastępowana jest podkreśleniem, ponadto usuwa wszystkie inne znaki oprócz liter, cyfr podkreśleń oraz minusów.

engine_api.py

Plik odpowiada za stworzenie serwera API dla silnika scrapującego. Dzięki niemu, interfejs użytkownika może wysyłać zlecenia scrapowania do silnika i otrzymywać wyniki.

Funkcja start_scraping_endpoint()

```
def start_scraping_endpoint():
    params = request.get_json()
    if not params or 'query' not in params:
        print("ENGINE_API: Błąd - Brakujące dane 'query' w żądaniu.")
        return jsonify({"status": "error", "message": "Brakujące dane: query"}), 400

    query = params.get('query')
    min_price = params.get('min_price', '')
    max_price = params.get('max_price', '')
    sort_order = params.get('sort_order', 'price_desc')

    result_items, status_message = run_ebay_scraper(
        query, min_price, max_price, sort_order
    )

    if "pomyślnie" in status_message.lower():
        response_status = "success"
    elif "błąd zapisu do db" in status_message.lower() and result_items:
        response_status = "partial_success"
    else:
        response_status = "error"

    http_status_code = 200
    if response_status == "error":
        http_status_code = 500

    print(f"ENGINE_API: Wiadomość z scrapera: '{status_message}', Ustalony status odpowiedzi: '{response_status}', Znaleziono ogłoszeń: {len(result_items)}")

    return jsonify({
        "status": response_status,
        "message": status_message,
        "ads_found": len(result_items),
    }), http_status_code
```

Funkcja sprawdza, czy otrzymała wszystkie potrzebne dane (przynajmniej nazwa produktu), następnie wywołuje główną funkcję run_ebay_scraper (z pliku scraper.py), przekazując jej parametry wyszukiwania. Na koniec wysyła odpowiedź z powrotem do interfejsu użytkownika, informując czy scrapowanie się powiodło oraz ile ofert zostało znalezionych.

Inicjalizacja i uruchomienie serwera Flask

```
if __name__ == '__main__':
    print("ENGINE_API: Uruchamianie serwera Flask API silnika na porcie 5001")
    debug_mode = os.environ.get('FLASK_DEBUG', '0') == '1'
    app.run(host='0.0.0.0', port=5001, debug=debug_mode)
```

Ta część kodu odpowiedzialna jest za utworzenie aplikacji Flask oraz uruchomienie serwera. Serwer nasłuchuje żądania przychodzące na porcie 5001, co pozwala interfejsowi użytkownika komunikować się z silnikiem. Ustawienie host='0.0.0.0' sprawia, że serwer jest dostępny dla innych kontenerów.

Dockerfile

Plik pełni rolę instrukcji dla systemu Docker, opisującą krok po kroku proces tworzenia obrazu dla silnika scrapującego. Instaluje niezbędne biblioteki z pliku requirements.txt, kopiuje kod aplikacji, eksportuje port 5001 oraz uruchamia aplikację flask (engine_api.py) przy starcie kontenera.

```
FROM python:3.10-slim

WORKDIR /app_engine

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 5001

CMD ["python", "engine_api.py"]
```

Opis modułu interfejsu użytkownika

app.py

Plik steruje stroną internetową, którą widzi użytkownik. Przetwarza akcje użytkownika np. wyszukiwanie i generuje odpowiednie strony HTML.

Inicjalizacja i konfiguracja

```
app = Flask(__name__)

MONGO_URI = os.environ.get("MONGO_URI", "mongodb://localhost:27017/")
DB_NAME = "scraper_db"
ENGINE_API_URL = os.environ.get("ENGINE_URL", "http://engine_app:5001/start-scraping")
```

Inicjalizowane są zmienne środowiskowe używane do konfiguracji połączenia z MongoDB (MONGO_URI, DB_NAME) oraz adresu URL API silnika (ENGINE_API_URL).

Funkcja trigger_scrape()

```
@app.route('/trigger-scrape', methods=['POST'])
def trigger_scrape():
    params = request.get_json()
    if not params or 'query' not in params or not params.get('query').strip():
        return jsonify({"status": "error", "message": "Brakujące lub puste dane: query"}), 400

    query = params.get('query')
    min_price_str = params.get('min_price', '')
    max_price_str = params.get('max_price', '')
    sort_order = params.get('sort_order', 'price_desc')

    if min_price_str and max_price_str:
        try:
            min_price_val = float(min_price_str)
            max_price_val = float(max_price_str)
            if min_price_val < 0 or max_price_val < 0:
                return jsonify({"status": "error", "message": "Ceny nie mogą być ujemne."}), 400
            if min_price_val > max_price_val:
                return jsonify({"status": "error", "message": "Cena minimalna nie może być wyższa niż cena maksymalna."}), 400
        except ValueError:
            return jsonify({"status": "error", "message": "Nieprawidłowy format ceny. Proszę podać liczby."}), 400
    elif min_price_str:
        try:
            if float(min_price_str) < 0:
                return jsonify({"status": "error", "message": "Cena minimalna nie może być ujemna."}), 400
        except ValueError:
            return jsonify({"status": "error", "message": "Nieprawidłowy format ceny minimalnej."}), 400
    elif max_price_str:
        try:
            if float(max_price_str) < 0:
                return jsonify({"status": "error", "message": "Cena maksymalna nie może być ujemna."}), 400
        except ValueError:
            return jsonify({"status": "error", "message": "Nieprawidłowy format ceny maksymalnej."}), 400
```

```
engine_params = {
    'query': query,
    'min_price': str(min_price_str) if min_price_str else '',
    'max_price': str(max_price_str) if max_price_str else '',
    'sort_order': sort_order
}

try:
    response = requests.post(ENGINE_API_URL, json=engine_params, timeout=300)
    response.raise_for_status()
    return jsonify(response.json()), response.status_code
except requests.exceptions.Timeout:
    msg = "FLASK_UI: Błąd: Przekroczono czas oczekiwania na odpowiedź od silnika."
    print(msg)
    return jsonify({"status": "error", "message": msg}), 504
except requests.exceptions.ConnectionError:
    msg = "FLASK_UI: Błąd: Nie można połączyć się z silnikiem scrapującym."
    print(msg)
    return jsonify({"status": "error", "message": msg}), 503
except requests.exceptions.HTTPError as e:
    msg = f"FLASK_UI: Błąd HTTP od silnika: {e.response.status_code}"
    print(f"{msg} - Odpowiedź: {e.response.text}")
    try:
        return jsonify(e.response.json()), e.response.status_code
    except ValueError:
        return jsonify({"status": "error", "message": f"{msg} - {e.response.text}"}), e.response.status_code
except Exception as e:
    msg = f"FLASK_UI: Nieoczekiwany błąd podczas komunikacji z silnikiem: {str(e)}"
    print(msg)
    return jsonify({"status": "error", "message": msg}), 500
```

Funkcja biera parametry w formacie JSON, dokonuje ich walidacji (sprawdza, czy podano słowo kluczowe oraz waliduje poprawność podanej ceny), następnie wysyła żądanie do silnika scrapującego za pomocą biblioteki requests. Obsługuje również błędy komunikacji z silnikiem i zwraca odpowiedź (status, liczbę znalezionych ofert, wiadomość) do skryptu JavaScript na stronie użytkownika.

Funkcja show_results()

```
@app.route('/results')
def show_results():
    query_param_original = request.args.get('query', None)
    sort_param = request.args.get('sort', 'price_asc')

    ads_list = []
    db_error = False
    collection_name_display = query_param_original if query_param_original else "N/A (proszę najpierw wyszukać)"

    mongo_client_ui = get_db_client()
    if mongo_client_ui:
        db_ui = mongo_client_ui[DB_NAME]
        if query_param_original:
            collection_name_to_read = sanitize_collection_name_for_ui(query_param_original)

            if collection_name_to_read in db_ui.list_collection_names():
                collection = db_ui[collection_name_to_read]

                if sort_param == 'price_desc':
                    sort_direction = DESCENDING
                else:
                    sort_direction = ASCENDING
                ads_list = list(collection.find({}).sort([("price_value", sort_direction), ("_id", ASCENDING)]))

        if mongo_client_ui:
            mongo_client_ui.close()
    else:
        db_error = True

    return render_template('results.html', ads=ads_list, db_error=db_error, query=collection_name_display)
```

Funkcja odpowiedzialna jest za wyświetlenie wyników scrapowania. Odczytuje słowo kluczowe(query) oraz sposób sortowania (sort). Następnie łączy się z MongoDB, pobiera kolekcję danych, sortuje zgodnie z preferencjami użytkownika i przekazuje listę do results.html.

index.html

```
<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>eBay Scraper</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='css/style_index.css') }}">
</head>
<body>
  <div class="container">
    <h1>eBay Scraper</h1>

    <form id="scrapeForm">
      <div>
        <label for="query">Słowo kluczowe:</label>
        <input type="text" id="query" name="query" placeholder="np. iPhone 15 Pro" required>
      </div>
      <div>
        <label for="min_price">Cena minimalna ($)</label>
        <input type="number" id="min_price" name="min_price" placeholder="np. 100" min="0" step="0.01">
      </div>
      <div>
        <label for="max_price">Cena maksymalna ($)</label>
        <input type="number" id="max_price" name="max_price" placeholder="np. 1000" min="0" step="0.01">
      </div>
      <div>
        <label for="sort_order">Sortuj według ceny produktu:</label>
        <select id="sort_order" name="sort_order">
          <option value="price_asc">Rosnąco (cena produktu)</option>
          <option value="price_desc" selected>Malejąco (cena produktu)</option>
        </select>
      </div>
      <button type="submit" id="scrapeButton">Uruchom Scrapowanie</button>
      <div class="loader" id="loader"></div>
    </form>
```

```
</form>

<div id="statusMessage" class="message"></div>

<div class="links">
  <a id="resultsLink" href="{{ url_for('show_results') }}">Zobacz Wyniki</a>
</div>
</div>

<script>
  const scrapeForm = document.getElementById('scrapeForm');
  const scrapeButton = document.getElementById('scrapeButton');
  const loader = document.getElementById('loader');
  const statusMessageDiv = document.getElementById('statusMessage');
  const queryInput = document.getElementById('query');
  const minPriceInput = document.getElementById('min_price');
  const maxPriceInput = document.getElementById('max_price');
  const resultsLink = document.getElementById('resultsLink');

  scrapeForm.addEventListener('submit', function(event) {
    event.preventDefault();

    const formData = new FormData(scrapeForm);
    const currentQuery = formData.get('query');
    const minPriceStr = formData.get('min_price');
    const maxPriceStr = formData.get('max_price');
```

```
const maxPriceStr = formData.get('max_price');

if (!currentQuery.trim()) {
  statusMessageDiv.textContent = 'Proszę podać słowo kluczowe.';
  statusMessageDiv.className = 'message error';
  statusMessageDiv.style.display = 'block';
  return;
}

const minPrice = parseFloat(minPriceStr);
const maxPrice = parseFloat(maxPriceStr);

if (minPriceStr && maxPriceStr && !isNaN(minPrice) && !isNaN(maxPrice) && minPrice > maxPrice) {
  statusMessageDiv.textContent = 'Cena minimalna nie może być wyższa niż cena maksymalna.';
  statusMessageDiv.className = 'message error';
  statusMessageDiv.style.display = 'block';
  return;
}

if ((minPriceStr && parseFloat(minPriceStr) < 0) || (maxPriceStr && parseFloat(maxPriceStr) < 0)) {
  statusMessageDiv.textContent = 'Ceny nie mogą być ujemne.';
  statusMessageDiv.className = 'message error';
  statusMessageDiv.style.display = 'block';
  return;
}

const params = {
  query: currentQuery,
  min_price: minPriceStr || '',
  max_price: maxPriceStr || '',
  sort_order: formData.get('sort_order')
};
```

```
sort_order: formData.get('sort_order')
});

scrapeButton.disabled = true;
loader.style.display = 'block';
statusMessageDiv.style.display = 'none';
statusMessageDiv.className = 'message';
statusMessageDiv.textContent = 'Rozpoczynam scrapowanie eBay...';
statusMessageDiv.classList.add('info');
statusMessageDiv.style.display = 'block';

fetch(`${url_for('trigger_scrape')}`, {
  method: 'POST',
  headers: { 'Content-Type': 'application/json', },
  body: JSON.stringify(params)
})
.then(response => {
  if (!response.ok) {
    return response.json().catch(() => {
      throw new Error('Błąd HTTP: ${response.status} ${response.statusText}. Odpowiedź serwera nie była w formacie JSON lub była pusta');
    }).then(errorData => {
      throw new Error(errorData.message || `Błąd HTTP: ${response.status}`);
    });
  }
  return response.json();
})
.then(data => {
  let messageText = data.message || JSON.stringify(data);
  if (data.ads_found !== undefined) {
    messageText += ` Znaleziono ogłoszeń: ${data.ads_found}`;
  }
});
```

```
messageText += ` Znaleziono ogłoszeń: ${data.ads_found}`;
}
statusMessageDiv.textContent = messageText;

if (data.status === 'success') {
  statusMessageDiv.className = 'message success';
  if (resultsLink && currentQuery) {
    const currentSortOrder = document.getElementById('sort_order').value;
    resultsLink.href = `${url_for('show_results')}?query=${encodeURIComponent(currentQuery)}&sort=${encodeURIComponent(currentSortOrder)}`;
  } else {
    statusMessageDiv.className = 'message error';
  }
}

.catch(error => {
  console.error('Fetch error:', error);
  statusMessageDiv.textContent = 'Błąd: ' + error.message;
  statusMessageDiv.className = 'message error';
})

.finally(() => {
  loader.style.display = 'none';
  statusMessageDiv.style.display = 'block';
  scrapeButton.disabled = false;
});
});
</script>
</body>
</html>
```

Strona główna aplikacji zawiera formularz HTML, który pozwala użytkownikowi wprowadzić słowo kluczowe, podać przedział cenowy oraz wybrać metodę sortowania. Ponadto użytkownik po kliknięciu przycisku „Uruchom Scrapowanie” uruchamia skrypt, który wysyła wprowadzone przez użytkownika dane do serwera, następnie czeka na odpowiedź od serwera i na jej podstawie aktualizuje stronę pokazując przy tym odpowiedni komunikat (np. „Scrapowanie zakończone”).

results.html

```
<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Wyniki Scrapowania eBay - {{ query | e if query else 'Wszystkie' }}</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='css/style_results.css') }}">
</head>
<body>
  <div class="container">
    <div class="back-link-container">
      <a href="{{ url_for('index') }}" class="back-link">+ Powrót do Wyszukiwania</a>
    </div>
    <h1>Wyniki dla: {{ query | e if query else 'N/A' }}</h1>

    {% if db_error %}
      <p class="error-message">Wystąpił błąd połączenia z bazą danych. Nie można wyświetlić wyników.</p>
    {% elif ads %}
      <p class="results-info">Znaleziono: <strong>{{ ads | length }}</strong> ofert.</p>
      <table>
        <thead>
          <tr>
            <th>Tytuł</th>
            <th>Cena (tekst)</th>
            <th>Cena (numeryczna)</th>
            <th>Wysyłka</th>
            <th>Lokalizacja</th>
            <th>Link</th>
          </tr>
        </thead>
        <tbody>
          {% for ad in ads %}
```

```
            <tr>
              <td>{{ ad.title | e }}</td>
              <td>{{ ad.price_text | e }}</td>
              <td>{{ ad.price_value if ad.price_value is not none else 'N/A' }}</td>
              <td>{{ ad.shipping_info | e }}</td>
              <td>{{ ad.location | e }}</td>
              <td><a href="{{ ad.link }}" target="_blank" title="{{ ad.link | e }}">Link do eBay</a></td>
            </tr>
          {% endfor %}
        </tbody>
      </table>
    {% else %}
      <p class="no-results">Brak danych do wyświetlenia dla zapytania "{{ query | e if query else 'nieznanego' }}"
    {% endif %}
  </div>
</body>
</html>
```

Ten szablon jest odpowiedzialny za prezentację wyników scrapowania. Otrzymuję listę ofert (ads), następnie za pomocą pętli `{% for ad in ads %}` generuję tabelę HTML, wyświetlając informację o każdym znalezionym produkcie.

Dockerfile

```
FROM python:3.10-slim

WORKDIR /app_ui

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 5000

CMD ["python", "app.py"]
```

Plik jest odpowiedzialny za budowę obrazu docker dla kontenera interfejsu użytkownika, instaluje niezbędne biblioteki z pliku requirements.txt, kopiuje kod aplikacji, eksportuje port 5000 oraz uruchamia aplikację flask (app.py) przy starcie kontenera.

Opis pliku docker-compose.yml

Plik jest kluczowym elementem konfiguracji i zarządzania całą aplikacją. Umożliwia on zdefiniowanie oraz uruchomienie wszystkich modułów systemu (interfejs użytkownika, silnik scrapujący, baza danych). Ustawia dostępność strony internetowej na porcie 5000, która będzie dostępna dla komputera użytkownika, silnik na porcie 5001. Dbą, aby została zachowana odpowiednia kolejność uruchamiania kontenerów co jest kluczowe dla funkcjonowania aplikacji.

```
version: '3.8'

> Run All Services
services:
  > Run Service
  flask_ui:
    build: ./flask_app
    container_name: flask_ui_container
    ports:
      - "5000:5000"
    depends_on:
      - mongo_db
      - engine_app
    environment:
      - MONGO_URI=mongodb://mongo_db:27017/
      - ENGINE_URL=http://engine_app:5001/start-scraping
      - FLASK_DEBUG=1
      - PYTHONUNBUFFERED=1
    volumes:
      - ./flask_app:/app_ui
    networks:
      - app_network

  > Run Service
  engine_app:
    build: ./engine
    container_name: engine_container
    ports:
      - "5001:5001"
    depends_on:
      - mongo_db
    environment:
      - MONGO_URI=mongodb://mongo_db:27017/
      - PYTHONUNBUFFERED=1
    volumes:
      - ./engine:/app_engine
```

```
volumes:
  - ./engine:/app_engine
networks:
  - app_network

> Run Service
mongo_db:
  image: mongo:latest
  container_name: mongo_db_container
  ports:
    - "27017:27017"
  volumes:
    - mongo_data:/data/db
  networks:
    - app_network

volumes:
  mongo_data:

networks:
  app_network:
    driver: bridge
```

Opis modułu Bazy danych

Aplikacja wykorzystuje MongoDB jako system zarządzania bazą danych NoSQL, działający w dedykowanym kontenerze (mongo_db_container). Wszystkie informacje przechowywane są w bazie danych o nazwie scraper_db. Dla każdego zapytania wprowadzonego przez użytkownika, silnik scrapera tworzy osobną kolekcję, której nazwa jest generowana po jej przetworzeniu (zamiana na małe litery, usunięcie znaków specjalnych oraz zastąpienie spacji podkreśleniem, realizowaną przez funkcję sanitize_collection_name). Każdy dokument zapisany w kolekcji zawiera następujące pola klucze:

- **id:** unikalny identyfikator.
- **title:** tytuł ogłoszenia.
- **price_text:** tekst ceny pobrany ze strony.
- **price_value:** Cena przekonwertowana na wartość numeryczną (float).
- **shipping_info:** Informacje o koszcie wysyłki.
- **location:** Lokalizacja przedmiotu.
- **link:** Bezpośredni link do ogłoszenia na eBay.
- **query_source:** Słowo klucz, które doprowadziło do znalezienia tego ogłoszenia.

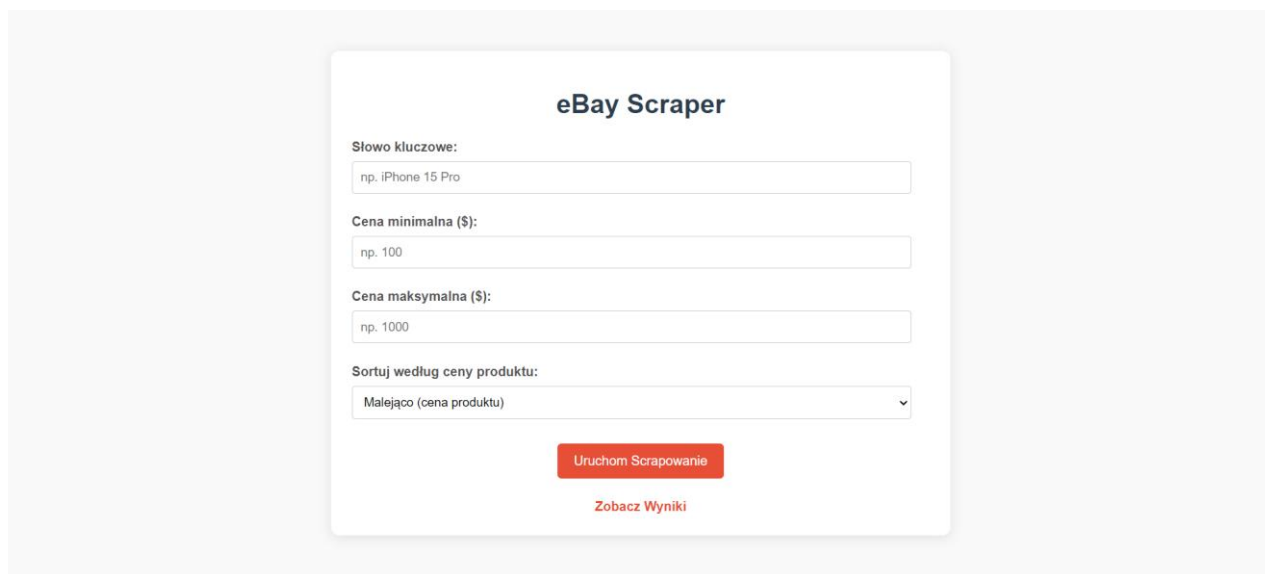
Przepływ danych między modułami

System scrapowania opera się na przepływie danych pomiędzy trzema skonteneryzowanymi modułami: interfejsem użytkownika (flask_ui), silnikiem scrapującym (engine_app) oraz bazą danych (mongo_db). Działanie aplikacji rozpoczyna się od interakcji użytkownika z interfejsem (flask_ui), gdzie wprowadzane są parametry wyszukiwania produktów. Następnie dane przekazywane są do silnika scrapującego (engine_app). Silnik najpierw asynchronicznie pobiera zawartość stron (wykorzystując asyncio i aiohttp) następnie za pomocą BeautifulSoup, parsuje, przetwarza oraz zapisuje wyniki w bazie danych MongoDB. Po zakończeniu operacji, silnik zwraca status do interfejsu użytkownika, który informuje o wyniku operacji.

Działanie aplikacji

Aby uruchomić aplikację oraz kontenery należy wpisać w terminalu polecenie docker-compose up --build, następnie użytkownik będzie w stanie lokalnie połączyć się z aplikacją (<http://localhost:5000>).

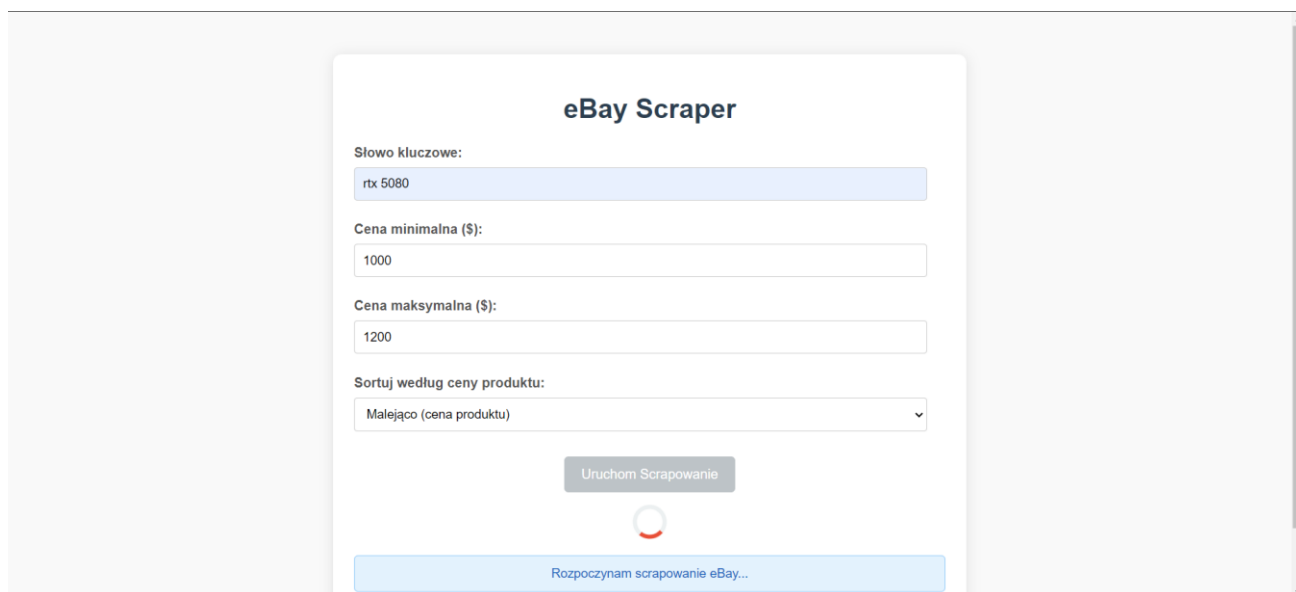
Po połączeniu się z aplikacją wyświetlone zostanie okno aplikacji, w którym użytkownik proszony jest o podanie danych do wyszukania, oraz wybór sortowania wyników (rosnąco lub malejąco).



The screenshot shows the 'eBay Scraper' application window. It features a title bar and a main content area with the following elements:

- Header:** 'eBay Scraper' in bold black text.
- Form Fields:**
 - Słowo kluczowe:** A text input field containing 'np. iPhone 15 Pro'.
 - Cena minimalna (\$):** A text input field containing 'np. 100'.
 - Cena maksymalna (\$):** A text input field containing 'np. 1000'.
 - Sortuj według ceny produktu:** A dropdown menu with 'Malejąco (cena produktu)' selected.
- Buttons:**
 - A red button labeled 'Uruchom Scrapowanie'.
 - A red link labeled 'Zobacz Wyniki'.

Gdy dane zostaną uzupełnione scraper rozpoczyna wyszukiwanie danych.



The screenshot shows the 'eBay Scraper' application window after the user has entered data. The interface is updated as follows:

- Form Fields:**
 - Słowo kluczowe:** The input field now contains 'rtx 5080'.
 - Cena minimalna (\$):** The input field now contains '1000'.
 - Cena maksymalna (\$):** The input field now contains '1200'.
 - Sortuj według ceny produktu:** The dropdown menu remains set to 'Malejąco (cena produktu)'.
- Buttons:**
 - The 'Uruchom Scrapowanie' button is now disabled (grayed out).
 - A loading spinner (a circle with a red segment) is displayed below the button.
 - A blue button labeled 'Rozpaczynam scrapowanie eBay...' is located at the bottom of the form.

Gdy scraper zakończy swoje działanie, zostanie wyświetlony odpowiedni komunikat, informujący o ilości wyszukanych produktów

eBay Scraper

Słowo kluczowe:

rx 5080

Cena minimalna (\$):

1000

Cena maksymalna (\$):

1200

Sortuj według ceny produktu:

Malejąco (cena produktu)

Uruchom Scrapowanie

Scraping completed successfully Znalezione ogłoszeń: 20.

Zobacz Wyniki

Użytkownik po wybraniu opcji „Zobacz Wyniki” będzie w stanie zobaczyć szukany produkt, który został automatycznie dodany do bazy danych.

Wyniki dla: rx 5080

Znaleziono: 20 ofert.

Tytuł	Cena (tekst)	Cena (numeryczna)	Wyszuka	Lokalizacja	Link
DeL Alienware NVIDIA GeForce RTX 5080 16GB GPU / Graphics Card - Triple Fan	\$1,200.00	1200.0	+ \$87.93 delivery	from United States	Link do eBay
PNY NVIDIA GeForce RTX 5080 Triple Fan 16GB OC - Tested No Missing ROPs	\$1,199.00	1199.0	+ \$96.21 delivery	from United States	Link do eBay
ASUS ROG Strix LC GeForce RTX 3080 Ti OC Edition 12GB GDDR6X Gaming Graphics...	\$1,199.00	1199.0	Shipping not specified	from United States	Link do eBay
PNY GeForce RTX 5080 Triple Fan 16GB GDDR7 Reflex 2	\$1,199.00	1199.0	+ \$68.15 delivery	from United States	Link do eBay
PNY NVIDIA GeForce RTX 5080 w/ FREE sata SSD throw in	\$1,174.99	1174.99	Shipping not specified	from United States	Link do eBay
MSI GeForce RTX 5070 Ti 16GB GAMING TRIO OC PLUS [NEW, SEALED]	\$1,149.88	1149.88	+ \$88.28 delivery	from United States	Link do eBay
NVIDIA RTX 4070 Ti 12GB Turbo Deep Learning AI Computing GPU GDDR6X OEM	\$1,129.00	1129.0	Free International Shipping	from China	Link do eBay
NVIDIA RTX 4070 Ti 12GB Turbo Deep Learning AI Computing GPU GDDR6X OEM	\$1,129.00	1129.0	Free International Shipping	from China	Link do eBay
NVIDIA RTX 4070 Ti 12GB Turbo Deep Learning AI Computing GPU GDDR6X OEM	\$1,129.00	1129.0	Free International Shipping	from China	Link do eBay
NVIDIA - GeForce RTX 5080 16GB GDDR7 Graphics Card - Gun Metal Founders Edition	\$1,125.00	1125.0	+ \$62.78 delivery	from United States	Link do eBay
MSI GeForce RTX 5080 16G Inspire 3X OC - No Box (Sealed)	\$1,121.45	1121.45	+ \$18.79 delivery	from Germany	Link do eBay
Colorful iGame GeForce RTX 5070 Ultra W OC 12GB GDDR7 Gaming GPU Graphics Card	\$1,099.00	1099.0	Free International Shipping	from China	Link do eBay
Colorful iGame GeForce RTX 5070 Ultra W OC 12GB GDDR7 Gaming GPU Graphics Card	\$1,099.00	1099.0	Free International Shipping	from China	Link do eBay
Colorful iGame GeForce RTX 5070 Ultra W OC 12GB GDDR7 Gaming GPU	\$1,099.00	1099.0	Free International	from China	Link do

Wyświetlenie tego samego produktu sortując rosnąco.

[← Powrót do Wyszukiwania](#)

Wyniki dla: rtx 5080

Znaleziono: 22 ofert.

Tytuł	Cena (tekst)	Cena (numeryczna)	Wysyłka	Lokalizacja	Link
MSI GeForce RTX 3090 VENTUS 3X OC 24GB GDDR6X w/Box (See Description)	\$1,002.00	1002.0	Free International Shipping	from Japan	Link do eBay
New ListingNVIDIA GeForce RTX 5080 16GB GDDR7 Graphics Card - Founders Edition - Gun Metal	\$1,010.00	1010.0	Shipping not specified	from United States	Link do eBay
COLORFIRE GeForce RTX 4070 SUPER meow 12GB GDDR6 Graphics Card GPU For Gaming	\$1,045.00	1045.0	Free International Shipping	from China	Link do eBay
GeForce RTX 5070 AERO OC 12G Graphics Card	\$1,050.00	1050.0	Shipping not specified	from United States	Link do eBay
PNY NVIDIA GeForce RTX 5080	\$1,050.00	1050.0	N/A	from United States	Link do eBay
PNY NVIDIA GeForce RTX 5080	\$1,050.00	1050.0	+\$77.66 delivery	from United States	Link do eBay
Nvidia GeForce RTX 5080 16GB GDDR7 FE Founders Edition - New Unopened-	\$1,075.00	1075.0	N/A	from United States	Link do eBay
Colorful iGame GeForce RTX 5070 Ultra W OC 12GB GDDR7 Gaming GPU Graphics Card	\$1,099.00	1099.0	Free International Shipping	from China	Link do eBay
Colorful iGame GeForce RTX 5070 Ultra W OC 12GB GDDR7 Gaming GPU Graphics Card	\$1,099.00	1099.0	Free International Shipping	from China	Link do eBay
Colorful iGame GeForce RTX 5070 Ultra W OC 12GB GDDR7 Gaming GPU Graphics Card	\$1,099.00	1099.0	Free International Shipping	from China	Link do eBay
MSI GeForce RTX 5080 16G Inspire 3X OC - No Box (Sealed)	\$1,121.45	1121.45	+\$18.79 delivery	from Germany	Link do eBay
NVIDIA - GeForce RTX 5080 16GB GDDR7 Graphics Card - Gun Metal Founders Edition	\$1,125.00	1125.0	+\$62.78 delivery	from United States	Link do eBay

Logowanie do bazy danych.

```
Microsoft Windows [Version 10.0.26100.4202]
(c) Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\Users\User>docker exec -it mongo_db_container bash
root@44e5b5f06347:/# mongosh
Current Mongosh Log ID: 68445cf24b4ad9d23169e327
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.5.2
Using MongoDB:      8.0.10
Using Mongosh:      2.5.2

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

-----
  The server generated these startup warnings when booting
  2025-06-07T15:20:58.129+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/pr
  odnotes-filesystem
  2025-06-07T15:20:59.136+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
  2025-06-07T15:20:59.136+00:00: For customers running the current memory allocator, we suggest changing the contents of the following sysfsFile
  2025-06-07T15:20:59.136+00:00: We suggest setting the contents of sysfsFile to 0.
  2025-06-07T15:20:59.137+00:00: vm.max_map_count is too low
  2025-06-07T15:20:59.137+00:00: We suggest setting swappiness to 0 or 1, as swapping can cause performance problems.
  -----

test> use scraper_db
switched to db scraper_db
scraper_db> show collections
rtx_5080
```

Połączenie się z kontenerem Docker, w którym działa MongoDB: docker exec -it mongo_db_container bash

Uruchomienie konsoli mongosh wewnątrz kontenera: mongosh

Przełączenie się do odpowiedniej bazy: use scraper_db;

Wyświetlenie wszystkich dokumentów z kolekcji rtx_5080: db.rtx_5080.find();

```
scraper_db> db.rtx_5080.find();
[
  {
    _id: '0e659ce6a65153bbc335425714d26b01',
    link: 'https://www.ebay.com/itm/226791754526?_skw=rtx+5080&itmmeta=81JX5GJ9C8M2A5ADHM6TR104Q06hash=item34cdd78b1e:g:hm8AA0S5wzgZoPaRV&itmprp=enc%3AAQAQAA
AAwFkggFvd1GGDuBw3yxCnilleWd1JPTRoALGB1Vp4N2FSat8Subett1LBJVnpR1V134a5m%2B0Fg8zcmUz1%2F5xR1kgxkTFUXBRcAeZnB2JE5%2B%2FFNznQ4JQUNUNsq6GkUwzo9uWZYEfV15%2FEf5z5J
P8T51QmLWmWjij6q42F7250aw0XJULjw0kr0gk3PX110uau5Q1s72Z63wL%2BPLkA916kdd06uHkAmilyh0mc%2FcPLNkxyd%2FoTllmuIYt3LY0wcI1VtRk0gTh1KvNAd8Q5Rm09syV18CFZg
VoZt%2FGH8R8%7Ctkp%3ABk9SRU6WybDpZQ',
    location: 'from United States',
    price_text: '$1,075.00',
    price_value: 1075,
    query_source: 'rtx 5080',
    shipping_info: 'N/A',
    title: 'Nvidia GeForce RTX 5080 16GB GDDR7 FE Founders Edition - New Unopened-'
  },
  {
    _id: 'b5baf75e1483d3be3556d9dda83ba78',
    link: 'https://www.ebay.com/itm/167558349710?_skw=rtx+5080&itmmeta=81JX5GJ9C8M2A5ADHM6TR104Q06hash=item270341838e:g:xhIAAD5wGpoP1C6&itmprp=enc%3AAQAQAA
AAwFkggFvd1GGDuBw3yxCnilleWd1JPTRoALGB1Vp4N2FSat8Subett1LBJVnpR1V134a5m%2B0Fg8zcmUz1%2F5xR1kgxkTFUXBRcAeZnB2JE5%2B%2FFNznQ4JQUNUNsq6GkUwzo9uWZYEfV15%2FEf5z5J
P8T51QmLWmWjij6q42F7250aw0XJULjw0kr0gk3PX110uau5Q1s72Z63wL%2BPLkA916kdd06uHkAmilyh0mc%2FcPLNkxyd%2FoTllmuIYt3LY0wcI1VtRk0gTh1KvNAd8Q5Rm09syV18CFZg
VoZt%2FGH8R8%7Ctkp%3ABk9SRU6WybDpZQ',
    location: 'from United States',
    price_text: '$1,050.00',
    price_value: 1050,
    query_source: 'rtx 5080',
    shipping_info: '+$77.66 delivery',
    title: 'PNV NVIDIA GeForce RTX 5080'
  },
  {
    _id: 'e917f176f7f1c84ca050335d155f6b8f',
    link: 'https://www.ebay.com/itm/376386664182?_skw=rtx+5080&itmmeta=81JX5GJ9C8M2A5ADHM6TR104Q06hash=item579da002f6:g:nAYAAeSwDSJo0imC&itmprp=enc%3AAQAQAA
AAwFkggFvd1GGDuBw3yxCnilleWd1JPTRoALGB1Vp4N2FSat8Subett1LBJVnpR1V134a5m%2B0Fg8zcmUz1%2F5xR1kgxkTFUXBRcAeZnB2JE5%2B%2FFNznQ4JQUNUNsq6GkUwzo9uWZYEfV15%2FEf5z5J
P8T51QmLWmWjij6q42F7250aw0XJULjw0kr0gk3PX110uau5Q1s72Z63wL%2BPLkA916kdd06uHkAmilyh0mc%2FcPLNkxyd%2FoTllmuIYt3LY0wcI1VtRk0gTh1KvNAd8Q5Rm09syV18CFZg
VoZt%2FGH8R8%7Ctkp%3ABk9SRU6WybDpZQ',
    location: 'from United States',
    price_text: '$1,125.00',
    price_value: 1125,
    query_source: 'rtx 5080',
    shipping_info: '+$62.78 delivery',
    title: 'NVIDIA - GeForce RTX 5080 16GB GDDR7 Graphics Card - Gun Metal Founders Edition'
  },
]
```