Wyższa Szkoła Ekonomiczna w Białymstoku
Kierunek: Informatyka Stosowana

Web Scraper

Wykonali:
Brunon Aleksiejczuk
Małgorzata Kulik
Dawid Owierkowicz

Cel projektu

Celem aplikacji jest automatyczne pozyskiwanie, przetwarzanie oraz zapisywanie do pliku treści tekstowych ze stron internetowych, zlokalizowanych w obrębie jednej domeny, przy zachowaniu kontroli nad zakresem działania, stabilnością systemu oraz efektywnym wykorzystaniem zasobów obliczeniowych.

Aplikacja została zaprojektowana jako narzędzie typu web crawler z funkcją ekstrakcji treści, umożliwiające użytkownikowi zebranie pełnego zbioru tekstów publikowanych w ramach wskazanego serwisu internetowego, bez konieczności ręcznego przeglądania poszczególnych podstron.

Metodologia i Narzędzia

Język programowania: Python 3.11

- wysoka czytelność składni,
- bogaty ekosystem bibliotek,
- dobre wsparcie dla przetwarzania sieciowego i wielowątkowości,
- szybki czas prototypowania.

Biblioteka requests

Realizacja komunikacji HTTP

- wysyłanie żądań GET,
- obsługa timeoutów,
- walidacja kodów odpowiedzi

Biblioteka requests upraszcza obsługę protokołu HTTP i zapewnia stabilność połączeń sieciowych.

Biblioteka BeautifulSoup

Parsowanie dokumentów HTML

- analiza struktury dokumentu,
- ekstrakcja linków,
- usuwanie elementów nietekstowych

Biblioteka jest odporna na niepoprawny HTML i umożliwia elastyczną analizę dokumentów o zróżnicowanej strukturze.

Metodologia i Narzędzia

Biblioteka tkinter

Budowa graficznego interfejsu użytkownika.

- konfiguracja parametrów pracy,
- wizualizacja statystyk,
- obsługa zdarzeń użytkownika

Tkinter jest biblioteką standardową Pythona, co eliminuje konieczność instalacji dodatkowych zależności i zwiększa przenośność aplikacji.

concurrent.futures

Zarządzanie pulą wątków.

- równoległe wykonywanie zadań,
- uproszczone zarządzanie wątkami

Moduł umożliwia bezpieczne i czytelne zarządzanie wielowątkowością bez konieczności ręcznego sterowania cyklem życia wątków.

threading i queue

Synchronizacja oraz komunikacja między wątkami.

- blokady (Lock),
- sygnały przerwania (Event),
- bezpieczna kolejka zadań

Zapewniają spójność danych i eliminują ryzyko warunków wyścigu w środowisku wielowątkowym.

Architektura kodu

Klient

GUI
- parametry
- start/stop
- statystyki (live update)
- konsola (ConsoleRedirect)
- pobieranie plików

UŻYTKOWNIK URUCHAMIA APLIKACJĘ

Otwiera się interfejs graficzny (GUI)

Użytkownik wpisuje URL startowy (np. https://example.com)

Ustawia parametry:
Maksymalna liczba stron
Liczba wątków (1-50)
Opóźnienie między requestami (min. 0.3s)

Klika przycisk "SCRAPUJ"

Kontroler

Crawler
- Config
- Queue (kolejka)
- Event
- ThreadPoolExecutor

KONTROLER (CRAWLER) INICJALIZUJE SYSTEM

Config normalizuje URL

Queue otrzymuje pierwszy URL do przetworzenia

ThreadPoolExecutor uruchamia pulę wątków roboczych

Komponenty robocze

HTTP Client
- GET
- timeout
- delay

HTMLParser
- links
- text
- errors

Domain Manager
- allowed[]
- is_allowed

WĄTKI ROBOCZE PRZETWARZAJĄ STRONY

Pobranie URL z kolejki
HTTP Client pobiera stronę
HTMLParser analizuje treść
DomainManager określa dozwolone domeny
Znalezione linki trafiają do Queue
Zapis wyników
Aktualizacja stanu

_process() (w wątku)

- HTTP Fetch
- Parse (DomainMgr)
- Storage save
- Stats update
- Queue add links

Stan i zapis

Stats
- visited
- queued
- errors

Storage
teksty.txt
all_links.txt
error_links.txt

Deduplicator (post-process)

Architektura kodu

GUI MONITORUJE POSTĘP

Co 300ms GUI aktualizuje wyświetlane statystyki:

-Przetworzone strony: ile URL już odwiedzono
-Znalezione linki: suma unikalnych URL w kolejce
-Błędy: nieprawidłowe linki, timeouty, błędy HTTP
-Czas działania: live timer od startu

Użytkownik może w każdej chwili kliknąć "PRZERWIJ":

Ustawia Event (stop_event)
Wszystkie wątki kończą pracę po aktualnym URL
Proces zatrzymuje się bezpiecznie

ZAKOŃCZENIE CRAWLINGU

Crawler kończy pracę gdy:

Osiągnięto limit stron
Kolejka jest pusta
Użytkownik przerwał proces

Storage:
Zamyka pliki (teksty.txt, all_links.txt)
Zapisuje błędy do error_links.txt

POST-PROCESSING: DEDUPLICATOR

Automatycznie uruchamia się Deduplicator:
1. Otwiera teksty.txt
2. Dzieli na sekcje (separator _____)
3. Dla każdej sekcji:
    -Usuwa duplikaty linii
    -Zachowuje kolejność pierwszego wystąpienia
4. Zapisuje do teksty_unikalne.txt
5. Wyświetla statystyki (ile linii usunięto)

UŻYTKOWNIK POBIERA WYNIKI

Po zakończeniu aktywują się przyciski:

Pobierz Teksty → zapisuje teksty_unikalne.txt w wybranym miejscu

Pobierz Errory → zapisuje error_links.txt

Wymagania funkcjonalne

1. Pobieranie adresu startowego od użytkownika
System umożliwia użytkownikowi podanie adresu URL strony początkowej za pomocą interfejsu tekstowego (CLI).
W przypadku braku schematu (http:// lub https://) system automatycznie go uzupełnia.

2. Konfiguracja parametrów procesu przeszukiwania
System umożliwia konfigurację podstawowych parametrów procesu przeszukiwania, w tym:
- maksymalnej liczby stron do odwiedzenia,
- liczby wątków roboczych,
- opóźnienia pomiędzy zapytaniami HTTP.

3. Przeszukiwanie stron internetowych w obrębie jednej domeny
System realizuje proces przeszukiwania (crawl) wyłącznie w obrębie domeny strony startowej, uwzględniając zarówno wersję z prefiksem www, jak i bez niego.

4. Pobieranie zasobów HTTP
System pobiera zawartość stron internetowych przy użyciu protokołu HTTP, obsługując błędy sieciowe oraz kody odpowiedzi serwera.

Wymagania funkcjonalne

5. Parsowanie dokumentów HTML
System analizuje pobrane dokumenty HTML i wyszukuje w nich odnośniki prowadzące do innych stron.

6. Normalizacja i walidacja adresów URL
System normalizuje wykryte adresy URL, przekształcając je do postaci absolutnej oraz odrzucając adresy niepoprawne lub niekompletne.

7. Filtrowanie odnośników
System filtruje wykryte odnośniki, pomijając:
-linki prowadzące poza dozwoloną domenę,
-linki do plików binarnych (np. PDF, obrazy, archiwa),
-linki specjalne (mailto, tel, javascript).

8. Zapobieganie duplikacji adresów URL
System zapobiega wielokrotnemu przetwarzaniu tych samych adresów URL poprzez prowadzenie zbioru adresów już odwiedzonych.

Wymagania funkcjonalne

9. Współbieżne przetwarzanie adresów URL
System umożliwia równoległe przetwarzanie wielu adresów URL z wykorzystaniem puli wątków, co zwiększa wydajność procesu crawl.

10. Rejestrowanie błędów
System rejestruje podstawowe błędy występujące podczas pobierania lub przetwarzania stron internetowych, w tym błędy sieciowe oraz nieprawidłowe typy zawartości.

11. Prezentacja postępu działania
System informuje użytkownika o postępie przetwarzania poprzez wyświetlanie komunikatów w GUI (statystyki live).

12. Zapis wyników do plików
System zapisuje:
- listę poprawnie odwiedzonych adresów URL wraz z tekstami ze stron do pliku tekstowego,
- listę błędów do osobnego pliku tekstowego - które linki były błędne oraz powód.

Wymagania niefunkcjonalne

1. Wydajność
System umożliwia efektywne przetwarzanie wielu stron internetowych dzięki zastosowaniu współbieżności oraz ograniczeń liczby wątków.

2. Skalowalność
System pozwala na dostosowanie liczby wątków roboczych oraz limitu przetwarzanych stron, co umożliwia jego wykorzystanie dla stron o różnej skali.

3. Niezawodność
System jest odporny na błędy sieciowe oraz niepoprawne dane wejściowe i nie przerywa działania w przypadku wystąpienia pojedynczych błędów.

4. Bezpieczeństwo przetwarzania
System ogranicza zakres działania do jednej domeny, co zapobiega niekontrolowanemu przeszukiwaniu zewnętrznych zasobów sieciowych.

Wymagania niefunkcjonalne

5. Odpowiedzialne korzystanie z zasobów
System implementuje mechanizm opóźnienia pomiędzy zapytaniami HTTP (rate limiting), ograniczając obciążenie serwerów zewnętrznych.

6. Przenośność
System jest przenośny i może być uruchamiany na różnych systemach operacyjnych wspierających środowisko Python 3.11.

7. Czytelność i utrzymywalność kodu
Kod źródłowy charakteryzuje się czytelną strukturą oraz logicznym podziałem funkcji, co ułatwia jego analizę oraz dalszą rozbudowę.

8. Minimalne zależności zewnętrzne
System wykorzystuje ograniczoną liczbę bibliotek zewnętrznych, co upraszcza proces instalacji oraz zmniejsza ryzyko problemów kompatybilności.

9. Deterministyczność działania
System działa w sposób przewidywalny, a proces crawl kończy się po spełnieniu jasno określonych warunków zakończenia (brak adresów lub osiągnięcie limitu).

Obsługa wyjątków

1. Obsługa wyjątków na poziomie komunikacji HTTP

Obsługiwane przypadki:
- brak odpowiedzi serwera,
- timeout,
- błędy DNS,
- odpowiedzi HTTP 4xx i 5xx,
- zerwane połączenia.

Reakcja systemu:
- wyjątek jest przechwycony,
- adres URL wraz z opisem błędu trafia do zbioru error_links,
- funkcja zwraca pustą listę,
- przetwarzanie innych stron trwa dalej.

```
try:
    r = requests.get(url, timeout=15, headers=self.config.headers)
    r.raise_for_status()

    content_type = r.headers.get('Content-Type', '')
    if 'text/html' not in content_type:
        return False, None, f"Nie-HTML (Content-Type: {content_type})"

    return True, r.text, None
except requests.exceptions.RequestException as e:
    return False, None, f"{type(e).__name__}: {e}"
```

Obsługa wyjątków

2. Obsługa nieprawidłowego typu zawartości

Chroni:
- PDF-y,
- obrazy,
- pliki binarne,
- API JSON.

Reakcja systemu:
- strona jest traktowana jako błąd logiczny,
- zapis do error_links,
- przerwanie dalszego parsowania.

```
try:
    r = requests.get(url, timeout=15, headers=self.config.headers)
    r.raise_for_status()

    content_type = r.headers.get('Content-Type', '')
    if 'text/html' not in content_type:
        return False, None, f"Nie-HTML (Content-Type: {content_type})"

    return True, r.text, None
except requests.exceptions.RequestException as e:
    return False, None, f"{type(e).__name__}: {e}"
```

Obsługa wyjątków

3. Obsługa błędów parsowania HTML

Obsługiwane przypadki:
- niepoprawna struktura dokumentu HTML,
- nieoczekiwane wartości atrybutów,
- błędy w trakcie przetwarzania linków.

Reakcja systemu:
-przechwytuje wyjątki,
-rejestruje je jako błędy parsowania.

```
try:
    links, errors, text = self.parser.parse(url, content)
    self.stats.add_errors(errors)
except Exception as e:
    self.stats.add_error(f"{url} | Błąd parsowania HTML: {type(e).__name__}: {e}")
    print(f"    ❌ Błąd parsowania")
    return []
```

Obsługa wyjątków

4. Obsługa wyjątków podczas przetwarzania linków

Obsługiwane przypadki:
- nieprawidłowe adresy URL,
- błędy składniowe linków,
- wyjątki w trakcie normalizacji adresów.

Reakcja systemu
- wyjątek jest jawnie przechwycony,
- informacja trafia do konsoli,
- błąd jest zapisany w error_links,
- pętla główna działa dalej.

```
    try:
        full = urljoin(source_url, href)
        parsed = urlparse(full)

        is_internal = self.dm.is_allowed(parsed.netloc) or parsed.netloc == ''

        if not parsed.scheme or not parsed.netloc:
            if is_internal or href.startswith(('/', './', '../', 'wp-content', 'uploads')):
                errors.append(f"{href} | Niepoprawny URL (brak schematu/domeny) | Źródło: {source_url}")
            continue

        if not self.dm.is_allowed(parsed.netloc):
            continue

        clean = f"{parsed.scheme}://{parsed.netloc}{parsed.path}"

        if any(clean.lower().endswith(ext) for ext in self.BINARY):
            continue

        links.append(clean)
    except Exception as e:
        errors.append(f"{href} | Błąd parsowania linku: {type(e).__name__}: {e} | Źródło: {source_url}")

return links, errors
```

Obsługa wyjątków

5. Obsługa wyjątków w zapisie danych (I/O)

Obsługiwane przypadki:
- brak uprawnień do zapisu,
- brak miejsca na dysku,
- uszkodzenie pliku.

Reakcja systemu:
- dostęp do plików jest synchronizowany.

```
try:
    self.texts_f.write(f"{url}\n\n{text}\n\n{self.SEP}\n\n")
    self.texts_f.flush()
    self.links_f.write(f"{url}\n")
    self.links_f.flush()
    return True
except Exception as e:
    print(f"   ⚠  Błąd zapisu: {e}")
    return False
```

Obsługa wyjątków

6. Obsługa wyjątków w środowisku wielowątkowym

Obsługiwane przypadki:
- wyjątki zgłoszone w wątkach roboczych,
- błędy przetwarzania pojedynczego zadania.

Reakcja systemu:
-wyjątki zgłaszane w wątkach są przechwytywane podczas odczytu wyniku zadania

```
try:
    new_links = future.result()
    for link in new_links:
        self.queue.put(link)
except Exception as e:
    print(f"❌ Błąd wątku: {e}")
    self.stats.add_error(f"{url} | Błąd wątku: {type(e).__name__}: {e}")
```

Obsługa wyjątków

7. Obsługa wyjątków w interfejsie graficznym

Obsługiwane przypadki:
- błędy konfiguracji,
- nieprawidłowe dane wejściowe,
- wyjątki podczas uruchamiania crawlera.

```
except Exception as e:
    error_msg = f"Błąd podczas crawlingu:\n{e}"
```

Reakcja systemu:
Wyjątki są przechwytywane i prezentowane użytkownikowi w formie komunikatów dialogowych.

Testy jednostkowe

Potrzebne biblioteki: pytest, beautifulsoup4, requests

1. Config.normalize_url – czysta logika

Test dotyczy normalizacji adresu URL wprowadzanego przez użytkownika do aplikacji. Metoda sprawdza, czy adres URL posiada schemat protokołu (http:// lub https://). Jeżeli schemat nie został podany, aplikacja automatycznie dodaje https://.

Celem testu jest weryfikacja, czy aplikacja poprawnie obsługuje adresy URL podane w skróconej formie, adres URL zostaje ujednolicony przed rozpoczęciem procesu crawlingu, uniknięte zostają błędy przy wykonywaniu zapytań HTTP.

Przeprowadzono ten test, ponieważ jest to **czysta logika biznesowa** (brak I/O, brak zależności zewnętrznych), metoda jest krytyczna dla poprawnego działania całej aplikacji – błędny URL uniemożliwia crawling, test zapewnia odporność aplikacji na błędy użytkownika (UX).

Testy jednostkowe

2. DomainManager.is_allowed – logika domen

Test dotyczy kontroli zakresu domen, po których może poruszać się crawler. Aplikacja pozwala na crawling domeny głównej, akceptuje zarówno wersję z www, jak i bez www, blokuje przejścia do domen zewnętrznych.

Celem testu jest sprawdzenie, czy crawler nie wychodzi poza dozwolony obszar, nie dochodzi do niekontrolowanego pobierania treści z innych domen, spełnione są podstawowe zasady etycznego i bezpiecznego web scrapingu.

Przeprowadzono ten test ponieważ ograniczenie domen jest kluczowe dla bezpieczeństwa i przewidywalności aplikacji, chroni przed przypadkowym crawlingiem całego Internetu, zapewnia zgodność z założeniami projektowymi aplikacji.

Testy jednostkowe

3. HTMLParser._extract_text – parsowanie HTML

Test dotyczy ekstrakcji treści tekstowej z dokumentu HTML. Metoda usuwa elementy nienależące do treści merytorycznej (skrypty, style, metadane), zachowuje nagłówki, paragrafy i listy, przygotowuje tekst do dalszego przetwarzania lub analizy.

Celem testu jest potwierdzenie, że z HTML-a usuwane są elementy techniczne, pozostaje czytelna treść przeznaczona dla użytkownika, formatowanie tekstu (np. listy punktowane) jest zachowane.

Przeprowadzono ten test, ponieważ parsowanie HTML jest jednym z najważniejszych elementów aplikacji, błędy w tym miejscu prowadzą do zaśmieconych danych wyjściowych, test zapewnia stabilność przetwarzania różnych struktur HTML.

Testy jednostkowe

Wyniki testów:

```
============================= test session starts ==============================
collecting ... collected 3 items

test_crawler.py::test_config_normalize_url_adds_https PASSED             [ 33%]
test_crawler.py::test_domain_manager_allows_same_domain_and_www PASSED   [ 66%]
test_crawler.py::test_html_parser_extract_text_removes_scripts_and_keeps_content PASSED [100%]

============================== 3 passed in 0.29s ===============================

Process finished with exit code 0
```

Opis problemu i Realizacja

Problemem, który rozwiązuje aplikacja, jest zbieranie wszystkich tekstów ze stron internetowych do przykładowego audytu, edycji tekstów lub sprawdzenia działania linków na stronie internetowej.

Aplikacja ułatwia zbieranie danych potrzebnych do realizacji audytu, eliminuje konieczność ręcznego przeklikiwania oraz zbierania tekstów ze stron internetowych, eliminuje powtórzenia tekstów.

Wyniki i Osiągnięcia

Aplikacja działa poprawnie, jeżeli użytkownik posiada dostęp do strony bez zabezpieczeń przed scrapingiem oraz atakami DDoS. Jest w stanie zebrać wszystkie potrzebne teksty, które mogą przydać się do dalszych działań lub analizy (warunkiem jest strona statyczna a nie dynamiczna, która nie posiada logowania, rejestracji itp.).

Wyniki i Osiągnięcia

Prezentacja aplikacji

Wnioski

Aplikacja działa poprawnie, a jej cel został osiągnięty w zadowalającym stopniu.

Kilka obszarów które mogą zostać ulepszone:
- aplikacja może łączyć się po różnych ip, aby uniknąć blokady połączeń,
- możliwość rozbudowania aplikacji o selenium, które pozwoli nam na pobieranie dynamicznych tekstów generowanych przy interakcjach ze stroną (post, generowanie java script),
- możliwość rozwoju o algorytmy sztucznej inteligencji, która będzie tworzyć automatyczne podsumowanie działań firmy w oddzielnym pliku, który jest gotowy do pobrania przez użytkownika,
- możliwość rozwoju o analizę oraz scraping zdjęć, które znajdują się na stronie internetowej.

Podsumowanie

Aplikacja jest wielowątkowym web crawlerem z graficznym interfejsem użytkownika, którego celem jest automatyczne pobieranie treści tekstowych ze stron internetowych w obrębie jednej domeny. System umożliwia użytkownikowi zdefiniowanie parametrów crawlingu, takich jak adres URL, maksymalna liczba stron, liczba wątków oraz opóźnienie pomiędzy zapytaniami HTTP.

Aplikacja analizuje strukturę HTML pobranych stron, filtruje linki zewnętrzne i binarne oraz ekstrahuje wyłącznie treść merytoryczną. Zebrane dane są zapisywane do plików tekstowych, a następnie poddawane procesowi deduplikacji w celu usunięcia powtarzających się fragmentów. W trakcie działania system monitoruje statystyki pracy w czasie rzeczywistym oraz obsługuje przerwanie procesu przez użytkownika.

Zastosowana architektura oparta na separacji odpowiedzialności, obsługa wielowątkowości oraz testy jednostkowe kluczowych komponentów zapewniają stabilność, skalowalność i możliwość dalszego rozwoju aplikacji.

Dziękujemy za uwagę!