МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 80.00.00.000 ПЗ

Група ІПм-24-3

Федунишин Владислав

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Федунишин Владислав Едуардович
(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі, методи та алгоритми вебскрейпінгу для збору динамічних даних

(назва роботи)

Інженерія програмного забезпечення
(назва освітньої програми)

121 - Інженерія програмного забезпечення
(шифр і назва спеціальності)

Федунишин В.Е.
(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Крихівський Михайло Васильович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту
Завідувач кафедри
доц. Бандура В. В.
(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Вовк Р. Б.
(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітньо-кваліфікаційний рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2025 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Федунишину Владиславу Едуардовичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “Моделі, методи та алгоритми вебскрейпінгу для збору динамічних даних**”**
керівник проекту (роботи) Крихівський Михайло Васильович, к.т.н., доцент
затверджені наказом закладу вищої освіти від **“** 05 ” листопада 2025 р. № 695/7

2. Строк подання студентом проекту (роботи) 15 грудня 2025 р.

3. Вихідні дані до проекту (роботи) Архітектура, формальний опис та алгоритми функціонування системи вебскрейпінгу для збору, нормалізації та зіставлення динамічних вебданих з онлайн-магазинів

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Теоретичні основи вебскрейпінгу та динамічних вебданих
2. Аналіз сучасних технологій вебскрейпінгу та інструментів збору динамічних даних
3. Математичні моделі та алгоритми вебскрейпінгу
4. Розробка та реалізація системи збору динамічних даних на основі вебскрейпінгу

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Функція для виводу даних категорії АТБ (рис. 4.5, ст. 87)
2. Вивід даних з словника в html шаблон (рис. 4.6, ст. 87)
3. Сторінка з товарами, які співпадають з обох магазинів (рис. 4.8, ст. 89)
4. Результат роботи матчера (рис. 4.9, ст. 90)
5. Результат роботи скрепера для АТБ (рис. 4.10, ст. 91)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц. к.т.н. Вовк Р. Б.	

7. Дата видачі завдання 04 вересня 2025 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____ _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури	20.09.2025	виконано
2	Аналіз предметної області та сучасних технологій вебскрейпінгу	01.10.2025	виконано
3	Дослідження методів роботи з динамічним вебконтентом, автоматизацією, механізмами захисту	12.10.2025	виконано
4	Розробка моделей та алгоритмів збору, оновлення, нормалізації та зіставлення	25.10.2025	виконано
5	Проєктування системи та програмна реалізація модулів	05.11.2025	виконано
6	Тестування розробленої системи та оформлення магістерської роботи	22.11.2025	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2025	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 96 с., 13 рис., 54 джерел.

Тема: Моделі, методи та алгоритми вебскрейпінгу для збору динамічних даних.

Об’єкт дослідження: моделі, алгоритми та технології збору й обробки динамічних вебданих.

Мета роботи: розроблення моделі, алгоритмів та програмної системи для збору, обробки та зіставлення динамічних вебданих з різних джерел з використанням сучасних інструментів вебскрейпінгу..

Предмет дослідження: інформаційні технології вебскрейпінгу, алгоритми нормалізації, зіставлення товарів та методи обробки динамічних змін у вебданих.

Результати дослідження:

Виконано комплексний аналіз сучасних інструментів і технологій вебскрейпінгу. На основі дослідження розроблено архітектуру системи збору даних. Реалізовано два скрейпери, алгоритми нормалізації тексту та порівняння товарів.

Висновок:

У результаті дослідження та програмної реалізації отримано повноцінну систему вебскрейпінгу, здатну стабільно збирати динамічні вебдані, нормалізувати їх, зіставляти між собою та зберігати історію змін.

ВЕБСКРЕЙПІНГ, ДИНАМІЧНІ ДАНІ, НОРМАЛІЗАЦІЯ, FUZZY MATCHING, PLAYWRIGHT, DJANGO, MATCHING ТОВАРІВ, РОЗПОДІЛЕНІ СИСТЕМИ, АНАЛІЗ ЦІН, СКРЕЙПЕРИ.

ANNOTATION

Master's work: 96 p., 13 fig., 54 sources..

Topic: Models, methods and algorithms of web scraping for collecting dynamic data.

Object of research: models, algorithms, and technologies for collecting and processing dynamic web data.

Purpose: development of a model, algorithms, and a software system for collecting, processing, and matching dynamic web data from various sources using modern web scraping tools.

Subject of research: information technologies of web scraping, algorithms for normalization and product matching, and methods for handling dynamic changes in web data.

Research results:

A comprehensive analysis of modern web scraping tools and technologies was performed. Based on the study, the architecture of a data collection system was developed. Two web scrapers were implemented, along with algorithms for text normalization and product comparison.

Conclusion:

As a result of the research and software implementation, a full-fledged web scraping system was obtained that is capable of stably collecting dynamic web data, normalizing it, matching it across different sources, and storing the history of changes.

WEB SCRAPING, DYNAMIC DATA, NORMALIZATION, FUZZY MATCHING, PLAYWRIGHT, DJANGO, PRODUCT MATCHING, DISTRIBUTED SYSTEMS, PRICE ANALYSIS, SCRAPERS.

ЗМІСТ

Стр.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API - прикладний програмний інтерфейс

REST API - прикладний інтерфейс взаємодії між клієнтом і сервером через HTTP

JSON - текстовий формат обміну структурованими даними

HTML - мова розмітки гіпертексту

CSS - каскадні таблиці стилів

JS (JavaScript) - мова програмування для динамічних вебсторінок

HTTP - протокол передачі гіпертексту

HTTPS - захищений протокол передачі гіпертексту

HTTP/2 - модернізована версія протоколу HTTP з мультиплексуванням

TLS - Transport Layer Security, протокол шифрування трафіку

DOM - Document Object Model, об’єктна модель вебсторінки

AJAX - асинхронні запити між браузером і сервером

SPA - односторінковий вебзастосунок (Single Page Application)

OCR - оптичне розпізнавання символів

Regex - регулярні вирази

Tokenization - процес поділу тексту на окремі токени

Stem / Stemming - приведення слова до базової форми

TF-IDF - статистичний показник важливості терміну в документі

NLP - Natural Language Processing, обробка природної мови

Fuzzy Matching - нечітке порівняння рядків

Levenshtein Distance - метрика відстані між двома рядками

Damerau–Levenshtein - метрика з урахуванням транспозицій символів

DTO - Data Transfer Object, об’єкт для передачі даних

ORM - Object-Relational Mapping, відображення об’єктів на таблиці БД

SQL - структурована мова запитів

CRUD - операції створення, читання, оновлення та видалення даних

Django - Python-фреймворк для веброзробки

Django ORM - ORM-система Django

DRF (Django REST Framework) - фреймворк для створення REST API на Django

Playwright - фреймворк для браузерної автоматизації

Selenium - система автоматизації веббраузерів

Puppeteer - JS-інструмент для керування браузером Chrome

Headless mode - режим роботи браузера без інтерфейсу

cURL / curl_cffi - HTTP-клієнти для виконання запитів

User-Agent - заголовок, що ідентифікує клієнтський застосунок

Fingerprinting - метод визначення клієнта за технічними ознаками

Proxy - сервер-посередник у мережі

IP Rotation - ротація IP-адрес задля уникнення блокування

ETL - процес Extract–Transform–Load, вилучення, перетворення, завантаження

Data Normalization - нормалізація даних

Clustering - метод групування об'єктів за подібністю

Product - канонічний товар у системі

StoreProduct - товар у конкретному магазині

PriceHistory - історичні дані про зміну цін

F1-score - метрика точності моделей класифікації

Precision - частка правильних позитивних відповідей

Recall - частка знайдених релевантних об’єктів

ПЗ - програмне забезпечення

БД - база даних

ОЗП (RAM) - оперативна пам’ять

ВСТУП

Актуальність роботи

У сучасних умовах стрімкого розвитку електронної комерції, цифрової аналітики та автоматизації бізнес-процесів вебскрейпінг став одним із ключових інструментів отримання актуальних даних із мережі Інтернет. Онлайн-магазини, торгові платформи, інформаційні портали та динамічні вебзастосунки щоденно генерують величезні обсяги структурованої й неструктурованої інформації, що має значну практичну цінність для аналітичних систем, сервісів порівняння цін, моніторингових платформ та алгоритмів прогнозування.

Особливої важливості набуває проблема збору динамічних даних, де контент постійно оновлюється, змінюється структура DOM, застосовується JavaScript-рендеринг та механізми захисту від ботів. У таких умовах традиційні методи статичного скрейпінгу є недостатніми, що вимагає застосування інноваційних моделей, алгоритмів та інструментів, здатних адаптуватися до зміни середовища та забезпечувати стабільність роботи.

Паралельно з цим, актуальним стає завдання нормалізації, уніфікації та матчингу зібраних даних, оскільки різні сайти представляють однакові сутності у різному форматі. Наприклад, різні онлайн-магазини можуть подавати одну й ту саму товарну позицію з різними назвами, різною структурою, різними одиницями вимірювання або акційними позначеннями. Це створює потребу у розробці ефективних алгоритмів обробки та зіставлення інформації.

Вебскрейпінг також стикається з юридичними, етичними та технічними аспектами: обмеженнями robots.txt, CAPTCHA, антибот-фільтрами, блокуваннями підозрілих запитів, а також питаннями безпеки передачі даних. Тому розробка стійкої системи збору інформації з динамічних вебсайтів, що враховує всі ці нюанси, є надзвичайно важливим і актуальним завданням.

Порівняння роботи з відомими розв’язаннями проблеми

Проблематика вебскрейпінгу широко досліджується в наукових роботах, технічній документації та аналітичних оглядах. Більшість існуючих систем та бібліотек орієнтовані на:

- статичний аналіз HTML (Requests, BeautifulSoup),
- побудову високорівневих краулерів (Scrapy),
- рендеринг JavaScript-контенту (Selenium, Puppeteer, Playwright),
- обробку великих обсягів даних (Pandas, Data Lakes),
- подолання технічних обмежень, пов'язаних зі складністю сучасних SPA-технологій.

Разом із тим, на практиці існують значні труднощі в уніфікації та порівнянні даних з різних джерел. Публічні моделі машинного навчання й інструменти нечіткої подібності (FuzzyWuzzy, RapidFuzz, TF-IDF, кластеризація) дозволяють частково автоматизувати процес матчингу, проте потребують адаптації до конкретної галузі та типів даних.

Оглянуті технології демонструють широкий спектр можливостей, проте кожне окреме рішення має свої недоліки - від низької продуктивності до невідповідності складним динамічним структурам вебсайтів. Саме тому постає необхідність створення комплексної системи, яка:

- поєднує динамічний і статичний скрейпінг,
- забезпечує стабільність при масштабуванні,
- підтримує нормалізацію даних,
- здійснює автоматичний матчинг однакових товарів,
- надає інтерфейс для інтеграції з аналітичними платформами.

Мета і задачі дослідження

Метою магістерської роботи є розроблення моделі, алгоритмів та програмної системи для збору, обробки та зіставлення динамічних вебданих з різних джерел з використанням сучасних інструментів вебскрейпінгу.

Для досягнення поставленої мети необхідно розв’язати такі **задачі**:

1) Провести аналіз існуючих методів і технологій вебскрейпінгу.
2) Дослідити особливості роботи з динамічними вебсайтами та SPA-застосунками.
3) Розробити архітектуру системи збору динамічних вебданих.
4) Реалізувати модулі скрейпінгу для різних типів вебресурсів.
5) Розробити алгоритми нормалізації та уніфікації отриманих даних.
6) Реалізувати алгоритми матчингу товарів між різними джерелами.
7) Створити базу даних для зберігання актуальних та історичних значень.
8) Реалізувати вебінтерфейс та REST API для взаємодії із системою.
9) Провести тестування системи та оцінити її продуктивність.

Об’єктом дослідження є моделі, алгоритми та технології збору й обробки динамічних вебданих.

Предметом дослідження є інформаційні технології вебскрейпінгу, алгоритми нормалізації, зіставлення товарів та методи обробки динамічних змін у вебданих.

Методи дослідження У роботі застосовано: аналітичний метод для систематизації інформації про технології вебскрейпінгу; системний підхід до проєктування програмної архітектури; методи ETL для обробки та нормалізації даних; статистичні та лінгвістичні алгоритми для матчингу; експериментальне тестування та вимірювання продуктивності; емпіричний аналіз для оцінки стійкості та точності роботи системи.

Наукова новизна одержаних результатів

У роботі:

- запропоновано вдосконалену модель збору динамічних вебданих з об’єднанням статичного та динамічного скрейпінгу;

- розроблено алгоритм зваженого матчингу товарів, що враховує назву, вагу, бренд та структурні ознаки;
- реалізовано універсальну процедуру нормалізації текстових даних для різних онлайн-магазинів;
- сформовано архітектуру системи, здатну масштабуватися та обробляти великі обсяги постійно оновлюваних даних.

Практичне значення одержаних результатів.

На основі проведеного дослідження розроблено програмну систему, яка дозволяє: збирати дані з динамічних вебресурсів, автоматично нормалізувати товари, зіставляти однакові позиції між різними онлайн-магазинами, аналізувати зміни цін у часі, інтегрувати дані в аналітичні або комерційні сервіси.

Система може бути використана для побудови сервісів порівняння цін, моніторингових платформ, маркетингових аналітичних інструментів та рішень у сфері електронної комерції.

Особистий внесок

1. Аналіз сучасних методів вебскрейпінгу та обробки даних.
2. Проєктування архітектури системи.
3. Реалізація скрейперів для динамічних і статичних сайтів.
4. Розробка алгоритмів нормалізації та матчингу.
5. Створення Django-системи з API та інтерфейсом.
6. Проведення тестування та оцінювання продуктивності.

Структура магістерської роботи.

Магістерська робота викладена на 96 сторінці друкованого тексту і складається зі вступу, чотирьох розділів, висновків, списку використаних джерел (54 найменування). Робота містить 13 рисунків.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ВЕБСКРЕЙПІНГУ ТА ДИНАМІЧНИХ ВЕБДАНИХ

1.1 Поняття вебскрейпінгу та класифікація вебданих

1.1.1 Типи контенту: статичний, динамічний

Сучасний Інтернет є одним з основних джерел даних для аналітики, машинного навчання, моніторингу ринку, досліджень та побудови інтелектуальних систем. Водночас більша частина інформації представлена у вигляді вебсторінок, що орієнтовані на відображення для людини, а не на машинну обробку. Саме тому набуває актуальності вебскрейпінг (web scraping) - процес автоматизованого отримання структурованих даних з вебресурсів за допомогою програмних засобів. У загальному вигляді вебскрейпінг можна визначити як технологію, що дозволяє автоматично завантажувати HTML-сторінки, виділяти з них релевантну інформацію та перетворювати її у зручний для подальшої обробки формат (таблиці, бази даних, набори даних для ML тощо) [1].

Вебдані, з якими працюють системи вебскрейпінгу, можна класифікувати за різними ознаками. Одним із ключових критеріїв є спосіб формування вмісту сторінки:

- статичний контент – HTML-сторінка формується на сервері один раз і надсилається клієнту без подальшої зміни структури документу;
- динамічний контент – значна частина вмісту підвантажується або оновлюється без повного перезавантаження сторінки за допомогою JavaScript, AJAX, fetch-запитів, WebSockets тощо [2].

1.1.2 Структура HTML, DOM, CSS-селектори

Статичні вебсторінки традиційно генеруються на сервері (наприклад, за допомогою PHP, Django, Node.js та інших серверних технологій) і надсилаються у

вигляді повністю готового HTML. Контент змінюється рідко (наприклад, сторінки з документацією, прості інформаційні сайти), а структура DOM є передбачуваною, що робить скрейпінг таких ресурсів порівняно простим: достатньо виконати HTTP-запит, отримати HTML-код і розібрати його за допомогою бібліотек на кшталт BeautifulSoup, lxml тощо [3].

1.1.3 AJAX, SPA, рендеринг на клієнті

Динамічні вебсторінки, навпаки, активно використовують JavaScript для побудови або оновлення контенту на стороні клієнта. Після первинного завантаження базової HTML-структури сторінка виконує додаткові запити до сервера (через AJAX, fetch API, WebSockets) для отримання даних у форматі JSON або XML і вже на клієнті формує відображення елементів.

Такий підхід характерний для односторінкових застосунків (SPA) на базі фреймворків React, Vue, Angular, де основна логіка рендерингу DOM перенесена у браузер користувача [3].

З точки зору вебскрейпінгу, статичний і динамічний контент вимагають різних підходів та моделей обробки, що обґрунтовує необхідність побудови окремих моделей, методів та алгоритмів збору даних для кожного типу вебресурсів.

1.2 Архітектура сучасного вебу та особливості збору динамічних даних

1.2.1 React/Vue/Angular, як проблеми для скрейперів

За останні роки архітектура вебзастосунків істотно змінилася: від простих багатосторінкових сайтів із повністю серверним рендерингом до складних SPA з інтенсивним використанням JavaScript, клієнтського рендерингу і численних фонових запитів. У класичній моделі серверного рендерингу (SSR) сервер генерує готовий HTML-документ для кожного запиту, а браузер переважно відображає уже сформований контент [4].

На противагу цьому, клієнтський рендеринг (CSR) передбачає, що сервер

повертає мінімальний HTML-каркас і JavaScript-бандл, який, будучи виконаним у браузері, самостійно формує вміст сторінки на основі даних, отриманих з API. Цей підхід є типовим для SPA на базі React, Vue, Angular [5].

Для таких застосунків характерні:

- клієнтська маршрутизація (без повного перезавантаження сторінки);
- активне використання AJAX / fetch / WebSockets для підвантаження даних;
- динамічне оновлення DOM без зміни URL або з його мінімальною зміною [6].

1.2.2 Виклики такі, як Cloudflare, Akamai, антибот захист

HTML, отриманий простим HTTP-запитом, може не містити потрібних даних, оскільки вони завантажуються окремими JavaScript-запитами після рендерингу сторінки в браузері.

Часто необхідно емулювати поведінку браузера: виконання JavaScript, прокрутка сторінки, обробка подій (клік «Показати ще», бескінечний скрол) [6].

Сучасні сайти використовують механізми захисту від ботів - аналіз поведінки, обмеження частоти запитів, захисні сервіси (Cloudflare, Akamai). Наприклад, платформи захисту дедалі частіше за замовчуванням блокують підозрілих ботів і спеціалізовані AI-краулери, щоб перешкодити масовому збору контенту без дозволу правовласників [7].

З огляду на це для роботи з динамічними вебресурсами широко застосовуються:

- браузерна автоматизація (Playwright, Selenium, Puppeteer);
- гібридні підходи, коли спочатку аналізуються мережеві запити в браузері, виявляються внутрішні API, а потім для збору даних використовуються безпосередні HTTP-запити до цих API без повного рендерингу сторінки [6].

Таким чином, архітектура та спосіб рендерингу контенту безпосередньо визначають моделі та алгоритми вебскрейпінгу, змушуючи переходити від простого «парсингу HTML» до комплексних рішень, що враховують поведінку SPA, мережеві

патерни та захисні механізми.

1.3 Моделі та підходи до вебскрейпінгу

У літературі та практиці можна виділити кілька типових моделей вебскрейпінгу, які відрізняються як технічним підходом, так і рівнем абстракції над вебресурсами.

1.3.1 Імперативні моделі (HTTP-клієнти)

Найбільш базовий підхід - використання HTTP-клієнтів (requests, curl, curl_cffi тощо) для надсилання запитів до вебсерверів і обробки отриманого HTML або JSON-відповідей. У цій моделі скрейпер явно задає URL, параметри запитів, заголовки (headers) та інші параметри, а потім вручну розбирає DOM чи JSON-структуру. Такий підхід добре підходить для статичних сайтів або відкритих API, де немає складної логіки рендерингу на клієнті [1].

1.3.2 Браузерна автоматизація

Для роботи з динамічними сайтами, що активно використовують JavaScript, застосовується модель, у якій скрейпер керує повноцінним браузером (звичайним або headless): відкриває сторінки, виконує скрипти, чекає завантаження елементів, скролить сторінку тощо. Цю модель реалізують фреймворки Selenium, Playwright, Puppeteer. Вона дозволяє отримати фінальний DOM після виконання всіх скриптів, що робить можливим скрейпінг складних SPA. Недоліком є зростання витрат ресурсів і часу виконання, але натомість підвищується стійкість до різних форматів динамічного контенту [8].

1.3.3 Гібридні моделі

Поширеним практичним компромісом є гібридний підхід. На початковому етапі використовується браузерна автоматизація для аналізу поведінки сторінки та

мережевих запитів. Після виявлення внутрішніх REST/GraphQL API, які повертають структуровані дані (JSON), подальший скрейпінг виконується вже напряму через HTTP-клієнти без повного рендерингу сторінки.

Такий підхід поєднує точність роботи з динамічними сайтами та ефективність «легких» HTTP-запитів, знижуючи навантаження як на інфраструктуру скрейпера, так і на цільові сайти [6].

1.3.4 Моделі циклічного збору даних

Окремим аспектом є часова модель роботи скрейпера. Для задач моніторингу цін, відстеження наявності товарів, змін у контенті тощо використовуються періодичні (cron-подібні) моделі, коли скрейпер із певною періодичністю обходитиме цільові ресурси, виявляючи зміни у даних. У таких моделях важливим стає не лише разовий збір, а організація історії змін, логування та збереження попередніх станів даних для подальшого аналізу.

1.3.5 Моделі синхронного / асинхронного скрейпінгу

За способом організації виконання запитів доцільно виділити синхронну та асинхронну моделі вебскрейпінгу. У синхронній моделі кожен запит виконується послідовно, що спрощує реалізацію, але обмежує продуктивність і масштабованість. Асинхронна модель (наприклад, на базі asyncio в Python) дозволяє виконувати десятки й сотні запитів паралельно, що особливо важливо при зборі динамічних даних, які часто розподілені по багатьох сторінках чи API-ендпойнтах [1].

1.4 Етичні та правові аспекти вебскрейпінгу

1.4.1 Robots.txt

Хоча вебскрейпінг є потужним інструментом для збору даних, він пов’язаний із низкою правових та етичних обмежень. У загальному випадку сам по собі факт автоматизованого доступу до публічно доступних сторінок не є однозначно

незаконним, однак зміст даних, спосіб доступу та юрисдикція можуть робити окремі сценарії скрейпінгу правово ризикованими [9].

Більшість сайтів використовують файл robots.txt для комунікації з роботами і краулерами, вказуючи дозволені та заборонені розділи для автоматичного обходу. Хоча формально robots.txt не завжди має пряму юридичну силу, ігнорування його інструкцій може сприйматися як недобросовісна практика і посилювати ризики у випадку спорів [10]. Останні дослідження також аналізують можливість трактування порушення robots.txt як фактору відповідальності в контексті контрактного, авторського та деліктного права [11].

1.4.2 Законність збору відкритих даних

Вебконтент часто охороняється авторським правом, а умови використання сайту (Terms of Service) можуть прямо забороняти масовий збір даних або їх комерційне використання без дозволу власника. На практиці це призводить до судових спорів між компаніями, які збирають великі обсяги даних, і правовласниками контенту (медіакомпанії, соціальні платформи тощо).

Особливу увагу слід приділяти збору персональних даних (PII). Багато юрисдикцій (GDPR в ЄС та інші закони) суворо регулюють обробку таких даних, включно з автоматизованим збором. Рекомендованою практикою є уникнення скрейпінгу персональних даних без явної згоди або законної підстави [12].

1.4.3 Обмеження та ризики

Важливим етичним і юридичним питанням є обходження технічних бар’єрів – захисних механізмів, CAPTCHA, систем виявлення ботів, спеціалізованих сервісів (Cloudflare тощо). У деяких випадках це може розглядатися як несанкціонований доступ або порушення заборонених технічних засобів. Це особливо актуально на тлі нещодавніх конфліктів між платформами та AI-сервісами, яких звинувачують у масовому скрейпінгу контенту всупереч заборонам [13].

Ряд рекомендацій виділяє набір принципів «етичного вебскрейпінгу»: повага

до robots.txt, обмеження частоти запитів, ідентифікація як бота, мінімізація навантаження на сайт, уникнення збору чутливих даних, дотримання умов використання, прозорість цілей збору [9].

У контексті магістерської роботи важливо не тільки реалізувати ефективні засоби збору динамічних даних, а й показати усвідомлення правових та етичних рамок, у яких такі системи повинні функціонувати.

1.5. Висновки до розділу

У даному розділі розглянуто теоретичні основи вебскрейпінгу та особливості роботи з динамічними вебданими. Було дано визначення вебскрейпінгу як процесу автоматизованого вилучення інформації з вебресурсів, проведено класифікацію вебданих за характером формування вмісту (статичний та динамічний контент) і показано, що зростання частки динамічних, JavaScript-орієнтованих вебзастосунків суттєво ускладнює процес збору даних.

Проаналізовано архітектурні особливості сучасного вебу, зокрема відмінності між серверним і клієнтським рендерингом, роль SPA-фреймворків, а також пов’язані з цим виклики для вебскрейпінгу, включно з необхідністю емулювати поведінку браузера і долати механізми динамічного завантаження контенту.

Окрему увагу приділено моделям та підходам до вебскрейпінгу:

- імперативний підхід на основі HTTP-клієнтів;
- модель браузерної автоматизації;
- гібридні рішення з використанням внутрішніх API;
- синхронні та асинхронні моделі виконання;
- періодичний (циклічний) збір даних.

Нарешті, було розглянуто етичні та правові аспекти вебскрейпінгу, серед яких дотримання robots.txt, повага до авторських прав і умов використання контенту, регуляції щодо персональних даних, а також питання обходу технічних захистів та відповідального використання зібраної інформації.

Отримані теоретичні результати створюють підґрунтя для подальшого дослідження конкретних технологій, інструментів і алгоритмів вебскрейпінгу, що буде виконано в наступному розділі, а також для розробки програмної системи збору динамічних даних у рамках магістерської роботи.

РОЗДІЛ 2
АНАЛІЗ СУЧАСНИХ ТЕХНОЛОГІЙ ВЕБСКРЕЙПІНГУ ТА ІНСТРУМЕНТІВ ЗБОРУ ДИНАМІЧНИХ ДАНИХ

2.1 Огляд бібліотек вебскрейпінгу (requests, curl_cffi, BeautifulSoup, Scrapy)

У сучасному Python-стеку для вебскрейпінгу сформувався набір інструментів, які розв'язують різні задачі: від виконання HTTP-запитів до парсингу DOM, обробки HTML та організації масштабного краулінгу. Найчастіше для побудови систем збору даних використовуються такі бібліотеки: requests, curl_cffi, BeautifulSoup, Scrapy. Кожна з них має власну архітектуру, сильні сторони та обмеження, що визначають область застосування.

2.1.1 Опис бібліотек

requests це найпопулярніша Python-бібліотека для надсилання HTTP-запитів. Її головна перевага простота використання та стабільність. Requests дозволяє легко виконувати GET/POST-запити, передавати параметри, заголовки, cookies, керувати сесіями та автентифікацією.

Можливості: підтримка всіх основних HTTP-методів; робота із сесіями (Session()), що зберігає куки та заголовки між запитами; просте встановлення custom headers (наприклад, User-Agent); підтримка SSL, редіректів, проксі; зручна обробка відповіді (r.text, r.json()).

Недоліки: не виконує JavaScript непридатний для динамічних сайтів; стандартна реалізація штатного TLS-стека інколи блокується захисними системами; менша продуктивність, ніж у curl-базованих рішень, при великій кількості паралельних запитів [13].

curl_cffi - сучасна і дуже потужна альтернатива requests, побудована поверх libcurl, але з Python-інтерфейсом. У реальних системах вебскрейпінгу curl_cffi дедалі

частіше замінює requests завдяки своїй продуктивності та можливості імперсонації браузера, що дуже важливо при роботі з антибот-захистами.

Можливості: використовує libcurl → величезний приріст продуктивності та стабільності; підтримує імперсонацію браузера (Chrome, Safari, Firefox), повторюючи TLS-handshake реальних браузерів; стійкіший до Cloudflare та Akamai, ніж requests; підтримка HTTP/2, HTTP/3; краща робота з проксі та паралельними запитами.

Сильні сторони: найвища швидкість серед Python-HTTP-клієнтів; один з найефективніших методів обходу антибот-систем без headless-браузера; ідеально підходить для великих проєктів зі збирання масивів даних.

Недоліки: менш дружній API, ніж у requests; потрібно додатково розбиратися з параметрами curl; не виконує JavaScript (це лише транспорт) [14].

BeautifulSoup - найпоширеніша бібліотека для парсингу HTML у Python. Вона перетворює HTML у дерево, з яким можна працювати через методи пошуку елементів.

Можливості: підтримка різних парсерів (html.parser, lxml, html5lib); пошук елементів за тегами, класами, CSS-селекторами; обробка "брудного" HTML, автоматичне виправлення структури; простота інтеграції з requests/curl_cffi.

Сильні сторони: чудово працює зі структурованим HTML; дуже проста і зрозуміла навіть для початківців; гнучка система пошуку елементів.

Недоліки: повільніша, ніж lxml; не підходить для дуже великих HTML-документів; не працює з JavaScript - потрібно спершу отримати фінальний DOM (наприклад, через Playwright) [15].

Scrapy - фреймворк для побудови великих, високопродуктивних систем вебскрейпінгу та краулінгу. Це не бібліотека, а повноцінна екосистема.

Можливості: асинхронний механізм запитів (Twisted); вбудовані черги, middleware, pipelines, throttling; паралельна обробка десятків тисяч сторінок; інтеграція з проксі, cookie-middleware, кешуванням; автоматичний збір даних за правилами або селекторами.

Переваги: найкраще рішення для масштабних проектів (десятки/сотні тисяч сторінок); економне використання ресурсів; можливість розподіленої архітектури; швидкість у 5–20 разів вища, ніж послідовний requests-скрейпінг.

Недоліки: крута крива навчання; важче інтегрувати динамічний (JavaScript) контент зазвичай вимагає Playwright/Selenium middleware; Надмірний для маленьких проєктів [16].

2.1.2 Порівняння ефективності

Висновки з порівняння:

- Requests це найпростіший інструмент для статичних сторінок.
- curl_cffi це найкращий вибір для високої продуктивності та обходу захистів без браузера.
- BeautifulSoup це ідеальний парсер HTML, але лише для статичного HTML.
- Scrapy це оптимальний вибір для масштабного збору даних, але без JavaScript.

Кожен із них оптимізований під конкретні задачі: від базових HTTP-запитів до великих фреймворків для промислового збору даних. З огляду на зростання частки динамічних вебзастосунків, важливо розуміти, що класичні бібліотеки (requests, BeautifulSoup) стають лише першою частиною повного стеку, тоді як ефективні сучасні системи повинні поєднувати їх з інструментами браузерної автоматизації або низькорівневими клієнтами (curl_cffi) для обходу обмежень.

2.2 Технології браузерної автоматизації та рендерингу контенту

Зростання популярності динамічних вебзастосунків, побудованих на основі React, Vue, Angular та інших SPA-фреймворків, призвело до того, що класичні інструменти вебскрейпінгу (requests, BeautifulSoup) перестали бути універсальними. Багато сайтів сьогодні: завантажують дані через AJAX / fetch, рендерять DOM у браузері, використовують динамічне завантаження контенту, застосовують

антибот-системи (Cloudflare, Akamai, PerimeterX), змінюють HTML-структуру під час роботи JavaScript.

У таких умовах виникає потреба в технологіях браузерної автоматизації, здатних: виконувати JavaScript, відтворювати поведінку реального користувача, повертати фінальний DOM, обходити блокування та антибот-системи.

Найпоширеніші інструменти сьогодні - Selenium, Playwright, Puppeteer.

2.2.1 Playwright

Playwright - сучасна бібліотека для автоматизації Chromium, Firefox та WebKit, створена командою Microsoft. Вона вважається найнадійнішим та найшвидшим вибором для вебскрейпінгу, особливо динамічного.

Можливості Playwright:

- Підтримка трьох основних браузерних рушіїв: Chromium, Firefox, WebKit.
- Автоматична синхронізація з мережею та DOM (не потребує “sleep”).
- Дуже швидкий рендеринг у headless-режимі.
- Можливість перехоплення мережевих запитів (network interception).
- Вбудоване очікування елементів та подій.
- Можливість емулювати мобільні пристрої, геолокацію, timezone.

Переваги:

- Найвища стабільність серед браузерних автоматизаторів.
- Краща продуктивність, ніж Selenium і Puppeteer.
- Здатність обходити деякі антибот-механізми завдяки натуральній поведінці браузера.
- Чіткий і сучасний API.

Недоліки:

- Висока ресурсомісткість у порівнянні з HTTP-скрейпінгом.
- Обмежений рівень антибот-обходу без додаткових проксі або fingerprint-технік [17].

2.2.2 Selenium

Selenium - найстаріший та найпоширеніший інструмент для браузерної автоматизації. Спочатку створювався для тестування, проте активно використовується і для web-scraping.

Можливості Selenium

- Підтримка Chrome, Firefox, Edge, Safari.
- Підтримує WebDriver-протокол.
- Гнучке налаштування проксі, cookies, user-agent.
- Велика екосистема бібліотек і драйверів.

Переваги:

- Підтримує найбільшу кількість браузерів.
- Має стабільну, багаторічну екосистему.
- Підійде для складної взаємодії з UI (формами, кнопками, input-елементами).

Недоліки:

- Повільніший за Playwright (через WebDriver-посередник).
- Схильний до падінь, помилок синхронізації та race-condition.
- Не оптимізований для масштабного скрейпінгу [18].

2.2.3 Puppeteer

Puppeteer був створений командою Google для автоматизації Chromium/Chrome.

Можливості Puppeteer:

- Повний контроль над Chrome/Chromium через DevTools Protocol.
- Дуже швидкий headless-режим.
- Перехоплення мережі, посторінкова навігація, емуляція пристроїв.

Переваги:

- Найкраща підтримка Chrome та DevTools.
- Висока швидкість роботи у headless.
- Ідеальний для JS-орієнтованих сайтів.

Недоліки:

- Підтримує лише Chromium менша різноманітність, ніж Playwright.
- Слабша екосистема порівняно з Selenium чи Playwright [19].

2.2.4 Порівняння технологій браузерної автоматизації за: швидкістю, стійкістю, обходом захистів

Playwright - найкращий сучасний інструмент для вебскрейпінгу динамічних сайтів. Висока швидкість, стабільність, підтримка різних браузерів, можливість перехоплювати мережу, краща сумісність із антибот-системами.

Selenium підходить для автоматизації тестування, але вже не є оптимальним інструментом для скрейпінгу на 2024–2025 роки.

Puppeteer - відмінний варіант для Chromium-орієнтованих задач, але програє Playwright через меншу гнучкість.

2.3 Алгоритми обходу динамічних елементів сторінки

У сучасних вебзастосунках значна частина контенту не завантажується разом із початковим HTML, а формується на стороні клієнта під час виконання JavaScript. Це включає: відкладене завантаження списків товарів, динамічні фільтри, infinite scroll, AJAX/fetch-запити, SPA-структури (React/Vue/Angular), приховані або lazy-loaded блоки DOM. Традиційні HTTP-запити не здатні виконати JavaScript і, відповідно, не можуть отримати весь кінцевий DOM. Для таких випадків використовуються алгоритми взаємодії з динамічними елементами, реалізовані у браузерних автоматизаторах - Playwright, Selenium, Puppeteer.

Нижче розглянемо основні техніки, які дозволяють правильно обробляти динамічний контент.

2.3.1 Очікування selector'ів

На відміну від статичних HTML-сторінок, у динамічних сайтах потрібні

елементи можуть з’являтися:

- після виконання асинхронного JS-коду,
- після звернення до серверного API,
- після взаємодії з інтерфейсом (клік, вибір фільтра),
- після "лінивого" завантаження (lazy load).

Замість небезпечного й нестабільного методу sleep(), інструменти автоматизації використовують розумні очікування.

Основні види очікувань:

- Очікування появи елемента в DOM page.waitForSelector("#product-list")
- Очікування видимості елемента (важливо для елементів, що завантажуються, але ще приховані)
- Очікування завершення мережевих запитів Playwright має автоматичні network idle очікування.
- Очікування конкретної подіїю. Наприклад, кінець анімації чи completion певного fetch-запиту.

Це потрібно оскільки завдяки алгоритмам очікування:

- зменшується кількість помилок "Element not found",
- підвищується стабільність скрейпера,
- зникає потреба в «магічних таймерах»,
- зростає швидкість, бо очікування завершується одразу після появи елемента [20].

2.3.2 Прокрутка

На багатьох сайтах контент завантажується тільки після прокрутки до певної області. Це можуть бути: каталоги товарів, стрічки новин, коментарі, фото-галереї, соціальні мережі.

Основні техніки:

- Плавна прокрутка сторінки вниз - step scroll Скрейпер по декілька сотень пікселів прокручує сторінку та чекає завантаження нових елементів.

- Прокрутка до конкретного елемента element.scrollIntoViewIfNeeded()
- Прокрутка до кінця сторінки з перевіркою змін DOM. Алгоритм: прокрутити - чекати - перевірити, чи збільшилась кількість елементів - повторити.
- Інтерсепція мережевих запитів infinite-scroll. Часто сайти просто роблять запити до /api/products?page=2.

Без прокрутки велика частина даних не потрапляє в DOM взагалі, тому скрейпер бачить лише перші елементи [21].

2.3.3 Емуляція дій користувача

Антиботи та динамічні сайти часто очікують реальних взаємодій. До них належать:

- клік кнопок: Load more, Next page, Accept cookies,
- введення тексту в пошуковий рядок,
- вибір фільтрів чи категорій,
- відкриття вкладок або акордеонів.

Приклади дій:

page.click("#show-more"),

page.hover(".product-card"),

page.fill("#search-input", "молоко").

Такі дії необхідні оскільки деякі елементи з’являються лише після дії користувача. Деякі сайти перевіряють поведінкові фактори, як рух миші, фокус input-поля тощо. Частина контенту завантажується тільки після події, наприклад "scroll" чи "click".

Емуляція дій дозволяє роботі поводитися як реальний користувач, зменшуючи ймовірність блокування [22].

2.3.4 Headless vs headed режими

Режими роботи браузера визначають поведінку автоматизатора.

Headless Mode (без графічного інтерфейсу). Переваги:

- найвища швидкість,
- мінімальне споживання ресурсів,
- ідеально для серверів та хмар,
- немає необхідності рендерити UI.

Проблеми:

- деякі сайти детектують headless-браузери,
- відсутні visual debug-можливості.

Headed Mode (з графічним інтерфейсом). Переваги:

- точна емуляція користувача,
- менша ймовірність бану,
- можливість візуально перевірити, що відбувається.

Недоліки:

- споживає більше ресурсів,
- повільніший.

Як сайти можуть виявити headless режим:

- navigator.webdriver = true.
- Відсутні GPU-процеси.
- Нестандартний user-agent.
- Відсутність рухів миші.
- Неповний стек WebGL.

Playwright частково виправляє ці сигнатури із Playwright headless-режим дає кращі результати, ніж у Selenium [23].

Алгоритми роботи з динамічним контентом є критично важливими для вебскрейпінгу сучасних сайтів. Правильне використання механізмів очікування, прокрутки, емуляції дій користувача та коректний вибір режиму роботи браузера дозволяють: стабільно та точно отримувати дані, мінімізувати ризик блокувань, працювати з будь-якими SPA-застосунками, зменшувати кількість помилок, пов’язаних із динамічністю DOM. На їх основі будується практично весь сучасний динамічний вебскрейпінг.

2.4 Технології обходу антибот-систем

Сучасні вебресурси активно використовують системи захисту від ботів з метою:

- запобігання масовому збору інформації (web scraping),
- захисту персональних даних,
- обмеження трафіку, який не приносить цінність,
- запобігання парсингу конкурентами.

До найпопулярніших антибот-платформ належать Cloudflare, Akamai Bot Manager, PerimeterX (Human Security), Datadome, Imperva, які аналізують як технічні характеристики запитів, так і поведінкові сигнатури користувача.

Цей підрозділ розглядає ключові технології обходу таких систем, включно з fingerprinting, token-challenge, проксі-технологіями, а також імперсонацією браузера (browser impersonation).

2.4.1 Fingerprinting

Fingerprinting - це технологія ідентифікації пристрою/користувача за набором параметрів середовища, включно з: User-Agent, Accept-Language, Canvas fingerprint, WebGL fingerprint, шрифти системи, роздільна здатність екрану, navigator.webdriver, Plugins, часова зона, Cookies enabled, Hardware Concurrency, Touch support.

Антибот-платформи на основі цих сигналів визначають: чи є користувач реальним, чи запит надходить із headless-браузера, чи використовується автоматизація Selenium, чи запит надходить із підозрілої інфраструктури (VPS, hosting provider).

Чому Selenium легко детектується: navigator.webdriver = true, недійсний Canvas fingerprint, відсутність браузерних плагінів, нестандартна поведінка миші, відсутність GPU-компонентів, нестандартний порядок шрифтів.

Методи обходу fingerprinting:

- Stealth-модулі для Playwright/Puppeteer.
- Імперсонація браузера (Chrome, Safari, Firefox).

Бібліотеки на кшталт curl_cffi дозволяють відтворювати реальні TLS-патерни браузера.

- Підміна Canvas / WebGL fingerprint. Плагіни "stealth" автоматично генерують валідні значення замість headless-сигнатур.
- Реалістичні поведінкові патерни: рух миші, скрол, затримки між діями, hover події [24].

2.4.2 Token-challenge

Багато сайтів застосовують challenge-технології, що перевіряють користувача перед доступом до контенту:

- Cloudflare JS Challenge,
- Cloudflare Turnstile,
- Akamai Bot Manager Challenge,
- reCAPTCHA v2/v3,
- hCaptcha.

Сервер надсилає браузеру JavaScript-код, який необхідно виконати: обчислення певної математичної функції, перевірка середовища виконання, генерація токена (cf_clearance), зберігання токена у cookie.

Без виконання JS-коду - доступ забороняється.

Requests і curl НЕ можуть пройти challenge, але: Playwright / Puppeteer можуть виконати JS отримують токен як реальний браузер. Curl_cffi може пройти challenge при коректній імперсонації Chrome.

Методи обходу:

- автоматичне проходження challenge через headless-браузер,
- збереження cookie-токена і подальше використання HTTP-запитів,
- використання проксі, які вже містять валідний clearance-token [25].

2.4.3 Проксі

Проксі найважливіший елемент у роботі з антиботами. Бот може бути заблокований одразу, якщо:

- він надсилає забагато запитів із одного IP,
- IP належить хостингу (DO, AWS, Hetzner),
- IP фігурує в чорних списках,
- IP має підозрілу історію (DC proxies).

Основні типи проксі: Datacenter proxies (дешеві, легко детектуються), Residential proxies (IP домашніх провайдерів важче заблокувати), Mobile proxies (найкращі для обходу захистів, але дуже дорогі).

Алгоритми ротації IP:

- циклічна ротація (кожен N-й запит),
- ротація при статус-кодах 403/429,
- ротація на рівні session-based cookies,
- ротація для кожного домену (domain-based routing) [26].

2.4.4 Імперсонація браузера (Chrome, Firefox)

Один із найефективніших способів обходу антиботів - імітація справжнього браузера, включно з: TLS-handshake патернами, JA3 fingerprint, ALPN negotiation, Cipher Suites order, HTTP/2 priorities, специфічними заголовками Chrome/Safari[27].

Антиботи аналізують не лише DOM, а й нижчі мережеві рівні, наприклад:

- TLS fingerprint (JA3),
- SEQ порядкові номери пакетів,
- час між пакетами,
- сигнатури HTTP/2 stream'ів[28].

Інструменти, які підтримують імперсонацію:

- curl_cffi: імперсонує Chrome 109–123, повторює TLS-поведінку браузера
- Playwright: природно використовує реальні браузери (Chromium, Firefox), підтримує реальні DevTools.

- Puppeteer: працює через Chrome DevTools Protocol, що дає автентичну поведінку [29].

Врезультаті сервер сприймає скрейпер як повністю валідний браузер, а не як бот.

2.5 Методи обробки великих масивів скрейпінгових даних

Процес вебскрейпінгу часто передбачає отримання значних обсягів даних, які відрізняються за структурою, форматом, повнотою та якістю. Через неоднорідність HTML-коду, різні стандарти оформлення інформації та можливу наявність шуму необхідно застосовувати комплексні методи обробки, що включають парсинг, нормалізацію, збереження, а також алгоритми виявлення дублікатів і кластеризації даних.

2.5.1 Парсинг HTML

Парсинг HTML - це процес перетворення HTML-документа на структуроване дерево, з якого можна вилучити необхідні дані.

У вебскрейпінгу використовуються різні підходи до парсингу, залежно від складності HTML та вимог до продуктивності.

Основні інструменти парсингу:

1. BeautifulSoup: дружній до “брудного” HTML (пошкоджені теги, неправильні вкладення); підтримує різні парсери (html.parser, lxml, html5lib); простий API для пошуку елементів.
2. lxml: надзвичайно швидкий XML/HTML-парсер; підтримка XPath-запитів; ідеальний для роботи з великими HTML-файлами.
3. CSS-селектори та XPath. Великі системи (Scrapy, Playwright) працюють з DOM через: page.querySelector() / querySelectorAll(); xpath().

Ці методи дають високу гнучкість та точність вилучення даних.

Методи стійкого парсингу: оскільки HTML часто змінюється, слід

використовувати:

- гнучкі селектори, що не прив’язані до конкретних класів;
- парсинг за структурою, а не за стилями;
- перевірки на наявність елементів, щоб уникати помилок;
- комбіновані селектори (наприклад: div.product > h3).

Проблеми парсингу:

- часті зміни класів та структур DOM - особливо в SPA;
- динамічно створювані елементи;
- неповний HTML у випадку AJAX-рендерингу (вирішується Playwright/Selenium).

2.5.2 Нормалізація

Нормалізація - це перетворення первинних даних у уніфікований формат, придатний для подальшої аналітики або зберігання в БД.

Для даних, отриманих зі скрейпінгу, нормалізація особливо важлива, тому що: різні сайти подають інформацію по-різному; назви товарів можуть містити зайві символи, маркетингові описи; одиниці вимірювання відрізняються (г, кг, мл); формати цін та чисел залежать від локалі.

Типові етапи нормалізації:

1. Очистка тексту: видалення HTML-тегів; заміна нерозривних пробілів; видалення спецсимволів, emoji, рекламних фраз.
2. Приведення регістру (наприклад: МОЛОКО 2.5% «ПРЕМІЯ» = молоко 2.5% премія).
3. Нормалізація одиниць вимірювання (наприклад: "900 г", "0.9 кг" = 900 г; "1 л", "1000 мл" = 1000 мл).
4. Виділення ключових атрибутів (Наприклад із назви товару: “Молоко «Галичина» пастеризоване 2.5% 900 г”, можна виділити: бренд: Галичина; тип: молоко; жирність: 2.5%; вага: 900 г) такі алгоритми можуть використовувати регулярні вирази або NLP-методи.

5. Видалення шуму (наприклад: «вигідна ціна», «акція», «новинка», «краща пропозиція») [31].

Практичні бібліотеки для нормалізації:

- regex (регулярні вирази),
- rapidfuzz (підвищена точність строкових операцій),
- text-unidecode,
- nltk / spaCy (для NLP-нормалізації).

2.5.3 Зберігання

Після збору та нормалізації даних необхідно зберегти їх у структурованій формі. Вибір методу зберігання залежить від: обсягу даних, потреб у пошуку та фільтруванні, необхідності зберігати історію змін, частоти оновлення.

Підходи до зберігання:

1. Реляційні бази даних (PostgreSQL, MySQL). Переваги: чітка структура, індекси, складні запити (JOIN, SELECT), ідеально для товарів, цін, атрибутів.
2. NoSQL-бази (MongoDB, ElasticSearch). Використовуються для: пошуку по тексту, зберігання JSON-структур, великих неструктурованих наборів даних.
3. Data Lakes та файли (CSV, JSON, Parquet). Підходить для зберігання великих масивів історичних даних.
4. Історизація (versioning). Зберігання попередніх станів даних необхідно для:моніторингу цін, аналізу трендів, побудови аналітичних моделей, відстеження змін у товарах. Схеми: Type 1 це заміна даних (без історії), Type 2 це створення нових записів із датами валідності, Type 3 це зберігання попереднього значення окремим полем [32].

2.5.4 Дублікати/кластеризація

Сайти часто дублюють дані: однаковий товар може мати різні назви, кілька варіантів упаковки, різні позиції на різних сайтах, зміни в регістрах або сортуванні слів.

Мета кластеризації - згрупувати записи, які описують один і той самий об’єкт.

Види дублікатів:

1. Точні дублікати. Однаковий текст вирішуються через hashing + перевірку.
2. Майже дублікати (Near-duplicates). Перестановки слів: Молоко 2.5% Галичина 900 г та Галичина молоко 900г 2.5%.
3. Семантичні дублікати. Семантично однакові товари, але текст інший. Хліб Білий 450 г та Хліб пшеничний нарізаний 0.45 кг [33].

Методи виявлення дублікатів:

1. Fuzzy Matching. За допомогою бібліотек: RapidFuzz, FuzzyWuzzy. Використовуються метрики: Levenshtein Distance, Token Sort Ratio, Partial Ratio [34].
2. TF-IDF + Cosine Similarity. Перетворює назви товарів у вектори застосовується косинусна близькість. Корисно для кластеризації даних у великих масивах.
3. Embeddings (BERT/Sentence Transformers). Для семантичного пошуку ідеально підходить SBERT.
4. Кластеризація. Алгоритми: K-Means, Hierarchical clustering, DBSCAN

У web scraping частіше застосовується Hierarchical clustering, бо не потребує фіксації кількості кластерів.

Алгоритм кластеризації товарів (типовий):

1. Нормалізувати назви (очистка, леми, вага, бренд).
2. Побудувати токенізований вектор.
3. Обчислити подібність між товарами.
4. Згрупувати товари у кластери.
5. Призначити кожному кластеру canonical product (основний товар) [35].

Методи обробки великих даних є ключовим елементом у побудові масштабної системи вебскрейпінгу.

Успішний скрейпінг це не тільки збір даних, але й подальша їх якісна

обробка, включаючи: структурований парсинг HTML, нормалізацію текстових даних, коректне та масштабоване збереження, виявлення дублікатів та кластеризацію.

Саме ці методи дозволяють перетворити "сирі" вебдані на корисну, аналітично цінну інформацію, що може використовуватися у дослідженнях, моніторингу ринків, порівнянні товарів, побудові рекомендаційних систем або машинного навчання.

2.6 Висновки до розділу

У цьому розділі було проведено системний аналіз сучасних технологій вебскрейпінгу та інструментів, що забезпечують ефективний збір динамічних вебданих у умовах зростаючої складності вебзастосунків. Дослідження показало, що вибір технічного стеку для збору даних значною мірою залежить від типу вебресурсу, способу рендерингу контенту, наявності JavaScript-динаміки, а також застосованих механізмів захисту від автоматизованого доступу.

У підрозділі 2.1 проаналізовано бібліотеки requests, curl_cffi, BeautifulSoup та Scrapy, що формують базу класичного вебскрейпінгу. Було встановлено, що вони ефективні для роботи зі статичними HTML-сторінками та внутрішніми API, проте малопридатні для динамічного контенту. Особливої уваги заслуговує curl_cffi, який завдяки імперсонації браузерів і використанню libcurl демонструє високу продуктивність і стійкість до антибот-систем, що робить його перспективним інструментом у сучасних умовах.

Підрозділ 2.2 був присвячений технологіям браузерної автоматизації - Playwright, Selenium і Puppeteer. Проведене порівняння підтвердило, що Playwright є найбільш оптимальним інструментом для збору динамічних даних, оскільки поєднує високу швидкість роботи, стабільність, підтримку кількох браузерних рушіїв, автоматичну синхронізацію з DOM і розширені можливості для перехоплення мережевих запитів. Selenium поступається за швидкістю та стійкістю, а Puppeteer,

хоча й забезпечує ефективну роботу з Chromium, має обмеження через підтримку лише одного рушія.

У підрозділі 2.3 розглянуто ключові алгоритми роботи з динамічним контентом - очікування селекторів, прокрутка, емуляція дій користувача, headless/headed режими. Ці методи дозволяють коректно обробляти SPA-застосунки, infinite-scroll механізми та сайти з відкладеним завантаженням контенту. Вони є фундаментальними для стабільної роботи сучасних скрейперів, оскільки забезпечують точну взаємодію з динамічним DOM та природну поведінку бота.

У підрозділі 2.4 проаналізовано антибот-системи та технології їх обходу, серед яких: fingerprinting, token-challenge, проксі-ротація та імперсонація браузера. Антибот-механізми сьогодні виконують глибоку перевірку поведінки, середовища виконання та мережевих характеристик запиту, тому обхід захисту вимагає комплексного підходу. Playwright із “stealth”-техніками та curl_cffi з підтримкою браузерних TLS-профілів показують найкращі результати у сучасних умовах.

Підрозділ 2.5 присвячений методам обробки великих масивів скрейпінгових даних, включно з парсингом HTML, нормалізацією, зберіганням інформації та виявленням дублікатів. Було визначено, що ефективна робота з вебданими передбачає комплексну обробку на кожному етапі: очищення тексту, уніфікацію форматів, виділення ключових атрибутів, структурування у реляційних базах даних, а також застосування алгоритмів fuzzy matching і кластеризації для встановлення відповідності між товарами з різних джерел.

У підсумку проведений аналіз дозволив сформувати концептуальну основу для побудови високонадійної системи збору динамічних даних. Сучасний вебскрейпінг вимагає інтеграції декількох класів технологій: низькорівневих HTTP-клієнтів, браузерної автоматизації, антибот-обхідних технік і алгоритмів постобробки даних. Саме на цьому комплексному поєднанні інструментів ґрунтуватиметься архітектура програмної частини, що буде розроблена в наступному розділі.

РОЗДІЛ 3
МАТЕМАТИЧНІ МОДЕЛІ ТА АЛГОРИТМИ ВЕБСКРЕЙПІНГУ

3.1 Математичні моделі збору динамічних даних

Процес вебскрейпінгу можна формалізувати у вигляді математичних моделей, які описують способи організації збору даних, обробки подій, управління потоками завдань та механізмами стійкості до збоїв.

Для динамічних вебресурсів (SPA, infinite scroll, сторінки з відкладеним завантаженням) застосовуються спеціалізовані алгоритмічні моделі, що оптимізують ефективність та актуальність отримуваних даних.

Цей підрозділ розглядає чотири ключові моделі: розподілений скрейпінг, циклічне оновлення (cron-модель), модель черги запитів,модель retry/backoff.

3.1.1 Модель розподіленого скрейпінгу

Розподілений вебскрейпінг це підхід, при якому множина скрейперів або робочих вузлів паралельно виконує підмножини загального задачного простору. Подібні підходи застосовуються в промислових системах паралельного краулінгу [36] та розподілених вебархітектурах [37].

Нехай:

$S=\{s_1,s_2,...,s_n\}$ - множина сторінок, які потрібно зібрати;

$W=\{w_1,w_2,...,w_m\}$ - множина вузлів;

$S_i \subseteq S$ - частина множини, призначена вузлу w_i.

Тоді:

$$S = \bigcup_{i=1}^{m} S_i,\ S_i \cap S_j = \varnothing \tag{3.1}$$

Час виконання на одному вузлі:

$$T_i = \sum_{s \in S_i} t(s) \tag{3.2}$$

Загальний час:

$$T = max(T_1, T_2, ..., T_m) \tag{3.3}$$

Мета оптимізації:

$$T \to min \tag{3.4}$$

Модель використовується у великих пайплайнах з Load Balancing та Worker Pools, де кожен вузол виконує обмежений набір запитів і не створює надмірного навантаження на цільовий сайт.

3.1.2 Модель циклічного оновлення (cron-модель)

Циклічне оновлення даних є основою моніторингових систем цін, наявності товарів та зміни контенту. Воно використовується в enterprise-рішеннях, описаних у [38].

Нехай:

f(t) - дані на момент часу t;

Δt - інтервал оновлення.

Тоді модель циклічних запусків:

$$f(t + \Delta t) = Scrapes(S) \tag{3.5}$$

Цикл:

$$t_0,\ t_0 + \Delta t,\ t_0 + 2\Delta t,\ \ldots \tag{3.6}$$

Частота вибирається залежно від інтенсивності зміни даних. Функція «свіжості» інформації:

$$Q = e^{-\lambda \Delta t}, \tag{3.7}$$

де λ - інтенсивність оновлень на сайті.

3.1.3 Модель черги запитів

Системи черг типу M/M/1, M/M/k описують процес обробки великої кількості вебзапитів у рамках одного чи кількох воркерів. Застосовуються у краулінгу великих обсягів даних [39].

У класичній моделі M/M/1:

інтенсивність запитів - λ;

інтенсивність обробки - μ.

Середня довжина черги:

$$L_q = \frac{\lambda^2}{\mu(\mu-\lambda} \tag{3.8}$$

Середній час очікування:

$$W_q = \frac{\lambda}{\mu(\mu-\lambda)} \tag{3.9}$$

У вебскрейпінгу це дозволяє:

- уникати перевищення лімітів запитів (429 Too Many Requests),

- оптимізувати використання ресурсів,
- рівномірно розподіляти навантаження між воркерами.

3.1.4 Модель retry/backoff

Механізми повторних спроб із відступом використовуються при мережевих збоях, timeouts, перевищенні лімітів запитів. Архітектура enterprise-скрейперів активно використовує експоненційний backoff [38].

Експоненційний backoff:

$$T_{retry}(n) = T_0 \cdot b^n, \tag{3.10}$$

де:

T_0 - початкова затримка,

b - коефіцієнт, зазвичай b=2,

n - номер спроби.

Стохастичний backoff (random jitter):

$$T_n = Uniform(0, T_0 \cdot b_n) \tag{3.11}$$

Переваги:

- зменшує навантаження на сервер,
- дозволяє виконувати запит після зняття rate-limit,
- мінімізує ймовірність блокування IP.

3.2 Алгоритми виявлення змін у динамічних даних

Під час вебскрейпінгу важливо не лише зібрати дані, але й визначити, які саме зміни відбулися у контенті сайту. У динамічних системах (e-commerce, новинні стрічки, каталоги товарів) інформація може:

- оновлюватися кілька разів на годину,
- змінювати ціну без зміни структури товару,
- змінювати текстовий опис,
- додавати нові елементи або приховувати старі.

Для виявлення таких змін застосовуються алгоритми diff-аналізу, хеш-порівняння, а також часові моделі оновлення. Ці техніки дозволяють оптимізувати зберігання, уникати дублювання та побудувати системи сповіщення про зміни.

3.2.1 Diff-аналіз

Diff-аналіз - це метод порівняння двох станів даних з метою виявлення різниць. Він традиційно використовується у версіонуванні текстів (Git), але широко застосовується і в вебскрейпінгу.

Нехай:

D_t - дані, зібрані в момент часу t,

$D_{t+\Delta t}$ - дані після наступного циклу збору.

Тоді diff-функція визначається як:

$$\Delta D = D_{t+\Delta t} - Dt, \tag{3.12}$$

де ΔD - множина змін (додано, змінено, видалено).

Типи diff-аналізу у вебскрейпінгу:

1. Структурний diff (DOM-diff). Порівняння HTML-структур дерева: зміна тегів, класів, появи/зникнення елементів.
2. Семантичний diff. Порівняння змісту текстових полів: зміна опису товару, оновлення технічних характеристик, зміна назви.
3. Числовий diff. Використовується для цін: $\Delta P = P_{new} - P_{old}$
4. Diff для списків. Виявлення нових товарів, зникнення старих.

Практичне застосування:

- моніторинг цін у магазинах,
- виявлення змін політик конфіденційності,
- відстеження оновлень каталогу [40].

3.2.2 Hash-порівняння

Хешування це метод відображення даних у коротке значення, яке дозволяє швидко визначати, чи змінився контент.

Нехай:

H(D) - хеш-функція (наприклад, SHA-256),

$h_t=H(D_t)$,

$h_{t+\Delta t}=H(D_{t+\Delta t})$.

Тоді зміна контенту визначається як:

$$h_t \neq h_{t+\Delta t} \tag{3.13}$$

Переваги хеш-аналізу:

- надзвичайно швидкий,
- компактний спосіб порівняння великих обсягів даних,
- ефективний для визначення, чи потрібно оновлювати запис у БД.

Види хеш-функцій:

1. Криптографічні (SHA-1, SHA-256, SHA-3). Гарантують унікальність.
2. Некриптографічні (xxHash, MurmurHash). Дуже швидкі для великих масивів даних.

Проблема прямого хешування HTML: якщо сайт змінює навіть один непомітний елемент (наприклад, ID), хеш зміниться, хоч важливої зміни немає.

Рішення: нормалізувати HTML перед хешуванням, хешувати окремо ключові елементи (наприклад, ціну, опис, вагу) [41].

3.2.3 Часові моделі оновлення

Для динамічних систем важливо знати не тільки що змінилося, а й коли та з якою частотою.

Нехай:

зміну даних описує випадкова величина,

інтенсивність змін - λ.

1. Пуассонівська модель змін. Для подій, що відбуваються випадково (зміна ціни, додавання товару):

$$P(N(t) = k) = \frac{(\lambda t)^k e^{-\lambda t}}{k!}, \quad (3.14)$$

де N(t)- кількість змін у момент часу t.

2. Модель середнього часу між змінами (MTTC)

$$MTTC = \frac{1}{\lambda} \quad (3.15)$$

Це дозволяє вибрати оптимальний інтервал оновлення:

$$\Delta t \approx \frac{1}{\lambda} \quad (3.16)$$

3. Модель трендів змін. Ціни мають властивість змінюватися зі зростанням активності ринку:

$$\Delta P(t) = f(t) + \epsilon, \quad (3.17)$$

де:

f(t) - тренд (наприклад, сезонність),

є - шум.

4. Марковські моделі. Зміна стану товару:

   "в наявності" - "немає в наявності" - "замінено"

   може бути описана марковським ланцюгом [42]:

$$P(X_{t+1} = j \mid X_t = i) \tag{3.18}$$

3.3 Алгоритми нормалізації та уніфікації зібраних даних

Дані, отримані в процесі вебскрейпінгу, характеризуються неоднорідністю: вони можуть містити шум, зайві символи, різні формати чисел, неконсистентні одиниці вимірювання, змінений порядок слів та маркетингові елементи. Тому критично важливо застосовувати процедури нормалізації та уніфікації, що перетворюють «сирі» вебдані у структурований формат, придатний для подальшого аналізу, порівняння та збереження.

Нормалізація є важливою частиною етапу ETL-процесів (Extract–Transform–Load) та широко використовується у системах вебскрейпінгу й data engineering.

3.3.1 Очищення тексту

Очищення (cleaning) - це перший етап нормалізації, який усуває шумові та несуттєві елементи.

Типові операції очищення включають:

1. Видалення HTML-тегів. HTML → текст: Молоко 2.5% «Галичина» → "Молоко 2.5% «Галичина»"
2. Усунення нерозривних пробілів та спецсимволів. Наприклад: \xa0, , emoji, торгові символи (™, ®).
3. Приведення до нижнього регістру. "МОЛОКО 2.5%" → "молоко 2.5%"
4. Видалення рекламних конструкцій. Наприклад: “вигідна ціна”, “акція”, “знижка -30%”. Ці фрази не мають відношення до суті товару.

5. Усунення зайвих пробілів. "молоко 2.5% галичина" → "молоко 2.5% галичина". Очищення може виконуватися за допомогою регулярних виразів або NLP-предобробки [31].

3.3.2 Приведення одиниць вимірювання

Це один із ключових етапів, особливо у товарних каталогах.

Одна й та сама вага/об’єм може бути записана десятками різних способів: “0.9 кг”, “900 г”, “900гр”, “0,9 kg”, “900gr”.

Це унеможливлює порівняння товарів без нормалізації.

Формальні правила нормалізації ваги:

Припустимо, сирове значення:

w={число, одиниця}

Нормалізована вага переводиться у базову одиницю (наприклад, грам):

$$w_{norm} = \begin{cases} w \cdot 1000, & якщо\ одиниця = «кг», \\ w, & якщо\ одиниця = «г», \\ w \cdot 1000, & якщо\ одиниця = «л», \\ w, & якщо\ одиниця = «мл». \end{cases} \quad (3.19)$$

Приклади:

"0.5 кг" → 500 г,

"2 л" → 2000 мл,

"250 мл" → 250 мл.

Проблеми нормалізації:

- комбіновані одиниці (“6×125 г”),
- неявні значення (“XL”, “середній розмір”),
- міжнародні формати (“oz”, “lb”).

У таких випадках використовують додаткові таблиці відповідності [43].

3.3.3 Регулярні вирази

Регулярні вирази використовуються для вилучення структурованих частин тексту: % жирності, вага, об’єм, кількість у упаковці.

Формалізація:

Нехай текст товару = T.

Тоді:

$$A = Extract(T, R), \tag{3.20}$$

де:

R - регулярний вираз, A - множина атрибутів.

Приклад: "Йогурт Галичина 2.5% 400 г"

Виділиться: жирність: 2.5%, вага: 400 г.

Переваги: швидкість, точність при стандартних форматах, простота реалізації.

Недоліки: погано працюють на неструктурованих текстах, не обробляють складні семантичні конструкції [44].

3.3.4 Токенізація

Токенізація - це процес поділу тексту на смислові елементи (токени): "Молоко 2.5% Галичина 900 г" → ["молоко", "2.5%", "галичина", "900", "г"]

Типи токенізації:

- Токенізація за пробілами вона проста та швидка.
- N-грами - корисні для fuzzy matching: молоко → мо, оло, лок
- NLP-токенізація (spaCy, NLTK): розпізнає числа, відокремлює пунктуацію, фільтрує стоп-слова.

Формальне визначення:

$$T = Tokenize(s) = \{t_1, t_2, \ldots, \{,\} \tag{3.21}$$

Запис товару у вигляді токенів полегшує: нормалізацію, порівняння рядків, виявлення схожості між різними товарами [45].

3.3.5 Стемінг

Стемінг - це приведення слова до базової (стрижневої) форми: "молочний" → "молок", "молоко" → "молок".

Це дозволяє виявляти семантичну близькість між словами.

Формальне визначення:

$$Stem(w) = w' \tag{3.22}$$

де w' - стем.

Найпопулярніші алгоритми:

- Porter Stemmer.
- Snowball Stemmer.
- Lancaster Stemmer.

Переваги: зменшення кількості токенів, покращення якості кластеризації та fuzzy matching.

Недоліки: можливе надмірне спрощення (overstemming), не завжди працює в україномовних текстах - потрібно використовувати Snowball для української.

3.4 Алгоритми матчингу однакових товарів з різних джерел

У системах вебскрейпінгу, що збирають інформацію з кількох інтернет-магазинів або платформ, виникає проблема: один і той самий товар може бути представлений із різними назвами, форматами та описами. Наприклад:

- “Молоко пастеризоване 2.5% Галичина 900 г”.
- “Галичина молоко 2,5% 0.9 кг пастер.”.
- “Молоко «Галичина» 2.5%, 900г”.

Хоча ці тексти різні, товар є однаковим.

Для вирішення цієї задачі застосовують алгоритми рядкової схожості, статистичні моделі тексту та методи кластеризації.

Матчинг товарів є ключовим компонентом систем порівняння цін, агрегаторів маркетплейсів і розробленої в магістерській роботі системи збору динамічних даних.

3.4.1 String similarity (Levenshtein, Damerau)

Алгоритми редагування рядків визначають мінімальну кількість операцій для перетворення рядка *A* у рядок B: вставка символу, видалення, заміна символу, (у Damerau) транспозиція сусідніх символів.

Формула Левенштейна:

Нехай:

i,j - позиції у рядках,

d(i,j) - відстань на цій позиції.

$$d(i,j) = \min \begin{cases} d(i-1,j)+1, \\ d(i,j-1)+1, \\ d(i-1,j-1)+m \end{cases}, \tag{3.23}$$

де:

$$m = \begin{cases} 0, & якщо\ a_i = b_j, \\ 1, & інакше. \end{cases} \tag{3.24}$$

Інтерпретація: Низьке значення рядки майже однакові, високе значні відмінності.

Проблеми цього методу: погано працює, коли порядок слів відмінний, не розуміє семантику, чутливий до зайвих символів. Напкриклад: "Молоко 2.5% 900 г" та "Молоко 900г 2.5%". Levenshtein буде завищувати різницю, бо змінився порядок

токенів [47].

3.4.2 TF-IDF

TF-IDF перетворює текст у вектор множинності термінів, що дозволяє порівнювати два тексти за допомогою косинусної подібності.

TF (частота терміну):

$$TF(t,d) = \frac{f_{t,d}}{\Sigma_k f_{k,d}} \tag{3.25}$$

IDF (зворотна частота документа):

$$IDF(t) = log\frac{N}{df_t}, \tag{3.26}$$

де:

N - загальна кількість документів,

df_t - кількість документів, що містять термін.

TF-IDF вектор:

$$TFIDF(t,d) = TF(t,d) \cdot IDF(t) \tag{3.27}$$

Порівняння TF-IDF векторів:

$$cos(\theta) = \frac{A \cdot B}{||A||||B||} \tag{3.28}$$

Переваги: добре працює, навіть якщо порядок слів змінений, нечутливий до зайвих символів, враховує важливість окремих слів.

Недоліки: не враховує вагу атрибутів (вага, % жирності, бренд), потребує словника термінів та векторизації [48].

3.4.3 FuzzyWuzzy

FuzzyWuzzy - бібліотека для нечіткого порівняння рядків, що використовує Levenshtein distance у комбінації з різними стратегіями токенізації.

Основні метрики:

1. Ratio. Оцінює схожість повних рядків.
2. Partial Ratio. Порівнює підрядки корисно при довгих назвах.
3. Token Sort Ratio. Сортує слова перед порівнянням: "Молоко 2.5% 900 г" = "900 г молоко 2.5%".
4. Token Set Ratio. Нормалізує множини слів найкраще працює для товарів.

Переваги: дуже простий у застосуванні, добре працює для коротких товарних назва.

Недоліки: не розуміє вагу атрибутів, не враховує семантику [49].

3.4.4 Алгоритм зваженої схожості: name + weight + brand

У web-scraping системах найточніший матчинг досягається комбінованими алгоритмами, що враховують окремі атрибути товару.

Нехай:

Sn - схожість назв,

Sw - схожість ваги,

Sb - схожість бренду.

Загальна оцінка:

$$S = \alpha Sn + \beta Sw + \gamma Sb, \quad (3.29)$$

де:

α,β,γ - вагові коефіцієнти, $\alpha+\beta+\gamma=1$.

Типові значення ваг: назва(0.6–0.7), вага(0.2), бренд(0.1–0.2).

Переваги: дає найкращий результат, коригує помилки рядкових алгоритмів, не плутає схожі назви різних брендів [50].

3.4.5 Кластеризація схожих продуктів

Кластеризація це процес групування товарів за схожістю.

Типові алгоритми:

1. K-Means. Використовується для embedding-векторів (TF-IDF або SBERT).
2. Hierarchical Clustering. Найпоширеніший метод для товарів: не потребує кількості кластерів, формує дендрограму подібностей.
3. DBSCAN. Гарний для виявлення щільних груп товарів.

Формальна модель:

Нехай кожен товар описаний вектором ознак:

$$vi=(name\_embedding,weight,brand)$$

Тоді кластеризація виконується так:

$$Cluster = \{vi: dist(vi, centroid) < \epsilon\}, \quad (3.30)$$

де dist - евклідова або косинусна відстань.

Використання в системі: встановлення відповідностей між товарами, побудова єдиного каталогу, визначення canonical product (основного товару) [35].

3.5 Моделі зберігання скрейпінгових даних

Ефективне зберігання даних є фундаментальним компонентом будь-якої вебскрейпінгової системи. Зібрана інформація може змінюватися з високою частотою (динамічні ціни, наявність товарів), потребувати історичного аналізу або бути доступною для швидких запитів. Тому архітектура зберігання повинна забезпечувати:

- цілісність даних,
- продуктивність CRUD-операцій,
- можливість історичного відстеження,
- відсутність дублювання,

- консистентність між магазинами та джерелами.

У цьому підрозділі розглянуто чотири ключові моделі зберігання:

- реляційні схеми (на прикладі Django ORM),
- кеш-моделі для оптимізації запитів,
- моделі актуальних цін,
- моделі історичних змін.

3.5.1 Реляційні схеми (Django ORM)

У вебскрейпінгових системах найчастіше застосовують реляційні бази даних (PostgreSQL, MySQL), оскільки вони забезпечують:

- строгі зв’язки між сутностями,
- індексування,
- транзакційність,
- ACID-властивості.

На практиці найефективнішою є схема, побудована за моделлю Product → StoreProduct → Price, де:

Product - нормалізована сутність товару (назва, бренд, вага),

Store - конкретний магазин-джерело,

StoreProduct - версія товару у магазині,

Price - історичні дані про ціну.

Лістинг 3.1. Фрагмент моделі Product

```
class Product(models.Model):
    name = models.CharField(max_length=255, db_index=True)
    brand = models.CharField(max_length=100, null=True)
    weight = models.IntegerField(null=True)  # у грамах
    created_at = models.DateTimeField(auto_now_add=True)
```

Лістинг 3.2. Модель зв'язку з магазином

```
class StoreProduct(models.Model):
                    product = models.ForeignKey(Product,
on_delete=models.SET_NULL, null=True)
        store = models.ForeignKey(Store, on_delete=models.CASCADE)
        url = models.URLField()
        normalized_name = models.TextField()
```

Переваги реляційної моделі: забезпечує нормалізацію та цілісність даних, дозволяє легко виконувати агрегацію, добре масштабується при використанні індексів, оптимальна для товарних баз, які мають чіткі структури [51].

3.5.2 Кеш-моделі

Кешування дозволяє зменшити навантаження на базу даних та оптимізувати час доступу до скрейпінгової інформації.

Основні види кешу:

1. In-memory cache (Redis, Memcached). Швидкий, підходить для: часто повторюваних запитів, тимчасових результатів обробки, результатів матчингу або нормалізації.
2. Django Low-Level Cache API. Наприклад, кешування результатів пошуку: cache.set(f"search:{query}", data, timeout=3600)
3. Кешування HTML-відповідей. Якщо сайт дозволяє stateless caching - можна кешувати raw HTML на 6–24 год.

Формальна модель кешу:

Нехай:

Q - запит, R - результат, C - кеш-функція.

Тоді:

$$C(Q) = \begin{cases} R_{\text{cached}}, & \textit{якщо } Q \in Cache, \\ R = f(Q), & \textit{і додати у кеш.} \end{cases} \tag{3.31}$$

Переваги: зменшує кількість звернень до БД до 90%, підвищує швидкість API у 5–20 разів, знижує ризик блокувань у БД [52].

3.5.3 Моделі актуальної ціни

У багатьох системах важливо зберігати актуальну ціну товару, але не створювати надмірну кількість записів у БД.

Найпоширеніші моделі:

1. Model StoreProduct.price. Зберігається тільки остання ціна: StoreProduct: - name, - price (last known). Проблема: немає історії.
2. PriceSnapshot модель (рекомендована). Остання ціна зберігається у StoreProduct, історія - в окремій таблиці:

Лістинг 3.3. Таблиця PriceHistory

```
class PriceHistory(models.Model):
            store_product   =   models.ForeignKey(StoreProduct,
on_delete=models.CASCADE)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    datetime = models.DateTimeField(auto_now_add=True)
```

Формальна модель:

Нехай:

pt - ціна в момент часу

Зберігаємо:

$$P = \{pt0, pt1, \ldots, ptn\}. \tag{3.32}$$

Система визначає, чи змінилася ціна:

$$p_{t+\Delta t} \neq p_t. \tag{3.33}$$

Якщо так - запис створюється [53].

3.5.4 Моделі історичних змін

Для аналітики цін, трендів, статистики та прогнозування необхідно зберігати історичні зміни.

Чому це важливо: відстеження акцій, побудова графіків цін, визначення найкращого часу для покупки, порівняння між магазинами.

Типи історичних моделей:

1. Append-only модель. Кожна зміна новий запис: H={state1,state2,...,staten}. Перевага - максимальна точність.
2. SCD Type 2 (Slowly Changing Dimensions). Змінюються тільки ключові поля, інші - зберігаються (start_date, end_date, is_current)
3. Event-based модель. Зберігаються події: price_changed, product_added, product_removed.
4. Delta-модель (зберігається тільки зміна): $\Delta p = pnew - pold$. Економить місце у БД.

Переваги історичної моделі:

- підтримка аналітики й прогнозування,
- можливість відстежувати поведінку ринку,
- формування time-series датасету,
- важливий компонент машинного навчання [54].

3.6 Висновок до розділу

У цьому розділі було проведено комплексний аналіз математичних моделей, алгоритмів та підходів, що лежать в основі сучасних систем вебскрейпінгу, особливо орієнтованих на обробку динамічних даних.

Дослідження показало, що ефективний вебскрейпінг не обмежується технічними засобами отримання HTML-контенту. Це - багаторівнева система, яка

включає моделі розподілу навантаження, механізми повторних спроб, алгоритми виявлення змін, текстову нормалізацію, методи порівняння та структури зберігання інформації.

По-перше, було розглянуто математичні моделі збору динамічних даних.

Модель розподіленого скрейпінгу забезпечує масштабованість, дозволяючи оптимально розподіляти завдання між кількома вузлами.

Cron-модель описує періодичні оновлення даних, визначаючи баланс між частотою збору та ресурсами.

Модель черг і механізм retry/backoff забезпечують стійкість системи до перевантажень та мережевих збоїв.

Разом ці моделі формують архітектурну основу для будь-якої високонадійної системи збору даних у реальному часі.

По-друге, були досліджені алгоритми виявлення змін у динамічних джерелах.

Diff-аналіз дозволяє визначати структурні та текстові зміни у DOM.

Hash-порівняння забезпечує швидку й економну перевірку цілісності даних.

Часові моделі змін (у тому числі пуассонівські та марковські процеси) дозволяють прогнозувати частоту оновлень і оптимізувати графік скрейпінгу.

Ці алгоритми дають змогу системі працювати ефективно навіть у разі високої варіативності даних.

По-третє, було детально розглянуто алгоритми нормалізації та уніфікації даних.

Очищення тексту, приведення одиниць вимірювання, регулярні вирази, токенізація та стемінг формують основу для подальшої обробки, зокрема - для модулів матчингу.

Без якісної нормалізації подальший аналіз даних був би некоректним або неможливим, особливо при порівнянні товарних назв, різних форматів ваги, брендів та описів.

По-четверте, було проаналізовано алгоритми матчингу товарів з різних джерел.

Методи порівняння рядків (Levenshtein, Damerau–Levenshtein), TF-IDF та cosine similarity забезпечують базову оцінку схожості.

Fuzzy matching дає змогу працювати з частковими збігами та відмінністю порядку слів.

Комбіновані моделі зі зважуванням окремих атрибутів значно підвищують точність відповідності між товарами.

Методи кластеризації дозволяють автоматично формувати групи схожих продуктів, що є ключовим для побудови єдиного каталогу з різних джерел.

По-п’яте, було обґрунтовано різні моделі зберігання скрейпінгових даних.

Реляційні схеми (на прикладі Django ORM) забезпечують структурованість і цілісність даних.

Кеш-моделі дають можливість оптимізувати продуктивність складних запитів і пошуку.

Окремо були розглянуті моделі актуальної ціни та моделі історичних змін - фундамент для аналізу ринкових тенденцій, побудови графіків і моніторингу.

Структури даних, що підтримують append-only або event-based підходи, дозволяють зберігати повну історію змін товарів і цін.

Розділ 3 сформував теоретичний фундамент, який визначає архітектуру, алгоритми та структури даних системи вебскрейпінгу.

Він показує, що сучасний вебскрейпінг - це:

- поєднання математичних моделей,
- алгоритмічного аналізу,
- обробки природної мови,
- оптимізації даних,
- розподілених обчислень,
- моделей зберігання та обробки.

Цей теоретичний базис є передумовою для побудови практичної системи збору динамічних вебданих, яка буде реалізована в наступному розділі.

РОЗДІЛ 4
РОЗРОБКА ТА РЕАЛІЗАЦІЯ СИСТЕМИ ЗБОРУ ДИНАМІЧНИХ ДАНИХ НА ОСНОВІ ВЕБСКРЕЙППІНГУ

4.1 Постановка задачі та функціональні вимоги

Метою даного підрозділу є опис задачі, яку вирішує розроблена система вебскрейпінгу, а також визначення функціональних вимог, обмежень та параметрів, що визначають її архітектуру та алгоритми роботи.

Система призначена для автоматизованого збору динамічних даних з онлайн-магазинів, їх нормалізації, уніфікації, порівняння та зберігання з можливістю подальшого аналізу та візуалізації.

4.1.1 Ціль скрейпінгу

У рамках роботи система виконує вебскрейпінг товарних каталогів інтернет-магазинів, що містять:

- перелік товарів категорії,
- назви та описи продуктів,
- ціну та стару ціну (якщо є знижка),
- зображення,
- характеристики (вага, об’єм, % жирності тощо),
- інформацію про наявність товару,
- акційні пропозиції.

Сайти є динамічними, містять JavaScript-рендеринг контенту, пагінацію, AJAX-запити та елементи захисту від автоматичного збору, що вимагає використання комбінації Playwright та curl_cffi/requests.

4.1.2 Дані для скрейпінгу

Кожний товар, отриманий зі сторінки магазину, включає такі ключові поля:

Основні атрибути продукту:

- Назва товару.
- Нормалізована назва.
- Опис (за наявності).
- Бренд.
- Вага / об’єм / кількість.
- Зображення товару.
- Унікальне посилання на сторінку товару.

Цінові атрибути:

- Поточна ціна.
- Стара ціна (якщо є знижка).
- Наявність на складі.
- Дані про акції / застосовані купони.

Мета збору:

- зберігати інформацію у структурованому вигляді,
- порівнювати однойменні товари між різними магазинами,
- аналізувати зміни цін,
- будувати агреговані списки та статистику.

4.1.3 Обрані магазини

У магістерській роботі реалізовано збір даних з двох великих українських ритейлерів:

1. АТБ (https://www.atbmarket.com/):

- частково статичний HTML,
- частково динамічні елементи,
- можливі AJAX-запити,
- відсутність повноцінного API,
- дані рендеряться на боці клієнта інколи потребує браузерної автоматизації.

2. Сільпо (https://shop.silpo.ua/):

- складний SPA з Angular/React-подібною структурою,
- повністю динамічне рендерення товарів,
- глибока залежність від JavaScript,
- захист від частих запитів,
- потребує Playwright для повного рендерингу сторінок.

У майбутньому система може бути розширена на інші магазини, що відповідають аналогічним параметрам.

4.1.4 Взаємодія з протоколами

Система взаємодіє з вебресурсами через такі протоколи:

- HTTP/HTTPS. Базовий транспорт для отримання HTML/JSON. Використовується як у requests/curl_cffi, так і у Playwright.
- HTTP/2. Деякі сайти, зокрема Silpo, підтримують HTTP/2, що дає змогу: виконувати паралельні запити, зменшувати latency, прискорювати завантаження ресурсів.
- WebSockets (опційно). Інколи сайти надсилають оновлені дані через WebSocket-сесії (акції, таймери). Система підтримує можливість розширення для таких випадків.
- Протоколи браузерної автоматизації (CDP, WebKit, Firefox Engine). Playwright працює поверх Chrome DevTools Protocol (CDP), що дозволяє: перехоплювати мережеві запити, керувати JavaScript-контекстом, виконувати динамічний рендеринг сторінки.
- TLS-імперсонація (curl_cffi). Для обходу антибот-систем використовується імітація справжнього браузера: Chrome TLS fingerprints, Android/Firefox client hello patterns.

Це підвищує стабільність при роботі з динамічними ресурсами.

4.1.5 Обмеження та вимоги до продуктивності

Для ефективної роботи система повинна відповідати низці вимог.

Функціональні обмеження:

- деякі сайти блокують часті запити за IP потрібна затримка або ротація проксі;
- SPA-застосунки вимагають повного рендерингу сторінки;
- AJAX-запити можуть динамічно змінювати HTML;
- дані на сайтах можуть бути неповні або відмінні за структурою.

Вимоги до продуктивності:

- Час повного скрейпу магазину: ≤ 5 хвилин для ~3000 товарів;
- Паралельне виконання: мінімум 5–10 воркерів;
- Споживання ресурсів: ≤ 400 МБ RAM на один Playwright-процес;
- Швидкість відповіді API Django: ≤ 150 мс для базових запитів;
- Навантаження на сайт: не більше 1 запиту / 1–2 сек на один домен, щоб уникнути блокувань.

Надійність:

- система повинна коректно відновлюватись після помилок 429, 503, timeout;
- дані повинні бути консистентними навіть при часткових збоях;
- retry/backoff алгоритми обов’язкові.

Вимоги до якості даних:

- нормалізація назв та ваги повинна забезпечувати високу точність;
- fuzzy matching ≥ 80% точності після нормалізації;
- уникнення дублікатів у базі.

4.2 Архітектура системи та проєктування ПЗ

Архітектура системи вебскрейпінгу визначає її модульність, масштабованість, надійність і здатність працювати з динамічними даними з різних джерел.

Система побудована за модульним принципом, що дозволяє незалежно

розвивати, оптимізувати та тестувати кожен компонент.

Загальна архітектура включає такі модулі:

1. Модуль скрейпінгу (динамічний + статичний).
2. Модуль нормалізації даних.
3. Модуль матчингу товарів.
4. База даних (реляційна ORM-підсистема).
5. Django-інтерфейс (адмінка + API + вебінтерфейс).

Така структура відповідає принципам розподілених систем збору даних, описаним у наукових джерелах з вебскрейпінгу та ETL-архітектур.

4.2.1 Модуль скрейпінгу

Модуль скрейпінгу є основним компонентом системи, який відповідає за отримання даних з вебсайтів.

У системі реалізовано два типи скрейперів:

- динамічний скрейпер на базі Playwright,
- статичний скрейпер на базі requests / curl_cffi.

Функції модуля:

1. Отримання HTML-контенту сторінки.
2. Обробка AJAX-запитів.
3. Парсинг HTML і JSON-відповідей.
4. Виявлення та повторна спроба запитів (retry/backoff).
5. Прокручування сторінок (для infinite scroll).
6. Обхід блокувань та обмежень швидкості.
7. Передача даних у модуль нормалізації.

Принцип роботи:

1. Скрейпер отримує список категорій товарів.
2. Для кожної категорії проходить всі сторінки пагінації.
3. Для кожного товару збирає структуру даних:

- назва,

- ціна,
- вага,
- посилання,
- знижка.

4. Дані зберігаються в тимчасову структуру (DTO).
5. Передаються у модуль нормалізації для уніфікації.

Переваги модульності:

- можна змінювати реалізацію скрейпера, не зачіпаючи бази даних та інші частини;
- можна легко додати новий магазин (Rozetka, Novus, Varus);
- забезпечує гнучке тестування скрейперів незалежно від системи.

4.2.2 Модуль нормалізації

Модуль нормалізації перетворює «сирі» дані з сайту у стандартизовану форму.

Основні задачі:

- очищення тексту від зайвих символів (™, ®, emoji),
- приведення до нижнього регістру,
- нормалізація ваги (900 г → 900; 0.9 кг → 900),
- нормалізація одиниць об’єму,
- видалення маркетингових вставок («акція», «знижка»),
- визначення бренду,
- виділення ключових атрибутів (жирність, вага, смак тощо).

Приклади перетворення "Молоко «Галичина» 2.5%, 0.9 кг АКЦІЯ":

- name: "молоко галичина 2.5%",
- brand: "галичина",
- weight: 900,
- normalized_name: "молоко галичина 2.5 900г".

Результат роботи модуля: структурований об’єкт, придатний для запису у БД

або для матчингу товарів.

4.2.3 Модуль матчингу товарів

Модуль матчингу один із найважливіших у системі. Він встановлює відповідність між товарами різних магазинів, що дозволяє:

- порівнювати ціни,
- об’єднувати дані,
- створювати єдиний агрегований каталог.

Етапи матчингу:

1. Нормалізація обох назв. (модуль нормалізації викликається повторно)
2. String similarity (Levenshtein / Damerau). Базова оцінка схожості.
3. Token similarity (FuzzyWuzzy / RapidFuzz). Token Set Ratio. визначення однакових наборів слів.
4. TF-IDF + cosine similarity. Покращує точність при складних назвах.
5. Зважена модель. Враховує: назву (0.6), вагу (0.2), бренд (0.2).
6. Поріг включення. Якщо $S \geq 0.80$, товари вважаються однаковими.

Результат:

Модуль повертає:

- Product (canonical entity).
- StoreProduct (версія товару у конкретному магазині).

4.2.4 База даних

База даних побудована на основі реляційної моделі Django ORM, яка забезпечує структуроване й масштабоване зберігання даних.

Основні таблиці:

1. Product. Зберігає канонічний товар:

- name,
- brand,
- weight,

- image_url,
- created_at.

2. StoreProduct. Зберігає товар конкретного магазину:

- product (FK),
- store (FK),
- normalized_name,
- price,
- old_price,
- url.

3. PriceHistory. Зберігає історичні зміни ціни:

- store_product (FK),
- price,
- datetime.

4. Store. Інформація про магазин:

- name,
- base_url.

Особливості структури:

- Відсутність дублювання даних.
- Можливість історичного аналізу.
- Підтримка індексів для швидкого пошуку.

4.2.5 Django-інтерфейс

У Django реалізовано три основні компоненти:

1. Адмін-панель. Використовується для:

- перегляду товарів,
- ручного редагування,
- перегляду історії змін,
- ревізії помилкових матчингів.

Покращена через:

- list_filter,
- search_fields,
- autocomplete_fields.

2. API (REST). REST-інтерфейс дозволяє:

- отримувати всі товари,
- виконувати пошук з fuzzy matching,
- отримувати найнижчу ціну серед магазинів,
- переглядати історію цін.

Формат - JSON.

3. Користувацький вебінтерфейс. Реалізований на Django Templates, забезпечує:

- пошук товарів,
- порівняння одного товару між магазинами,
- фільтрацію за категоріями,
- відображення актуальних і історичних цін.

4.3 Розробка скрейперів для динамічних сайтів (АТВ, Silpo)

У цьому підрозділі розглядається практична реалізація модуля скрейпінгу для двох інтернет-магазинів - АТБ та Сільпо, що суттєво відрізняються за структурою, механізмами рендерингу та поведінкою сторінки.

АТБ частково рендерить сторінку статично, але може використовувати внутрішні AJAX-виклики.

Сільпо є повністю динамічним SPA-застосунком, який відображає дані тільки після виконання JavaScript.

Для забезпечення максимальної стабільності та швидкості система використовує гібридний підхід:

- Playwright для динамічних сторінок (Silpo, частково АТВ)
- curl_cffi / requests → для статичних сторінок та lightweight-скрейпінгу

Такий дизайн дозволяє:

- мінімізувати навантаження на ресурси,
- зменшити час виконання,
- коректно обробляти динамічний DOM,
- протистояти антибот-системам.

4.3.1 Реалізація на Playwright (динамічний)

Playwright було обрано як основний механізм рендерингу динамічних сторінок завдяки його перевагам:

- повний JS-рендеринг (без пропуску елементів);
- підтримка Chrome, Firefox, WebKit;
- перехоплення мережевих запитів;
- автоматичне очікування селекторів;
- можливість запуску у headless/headed режимі;
- обхід простих антибот-механізмів.

Цей підхід є оптимальним для Silpo, де текст товарів з’являється після повного виконання Angular-подібного фронтенду.

Алгоритм роботи Playwright-скрейпера:

1. Запуск браузерного контексту з імперсонацією:
- фейковий User-Agent,
- реальні viewport-параметри,
- JavaScript enabled,
- опціональна проксі.
2. Завантаження категорії товарів:
   await page.goto(url, wait_until="networkidle").
3. Очікування рендерингу DOM:
   await page.wait_for_selector("article.product-card").
4. Прокрутка для підвантаження товарів (infinite scroll).
5. Поки кількість відрендерених карток зростає:

- scrollToBottom(),
- sleep(500–800 ms).

6. Витягування інформації про товар. Playwright дозволяє витягувати дані напряму через JS:

Лістинг 4.1. Витяг даних напряму через JS

```
titles = await page.eval_on_selector_all(
    "article.product-card h3.product-card__title",
    "nodes => nodes.map(n => n.innerText)"
)
```

7. Формування структури ProductDTO. Містить: name, price, old_price, link, weight (якщо знайдено), image_url.
8. Передача в модуль нормалізації.

Переваги Playwright у цьому проєкті:

- стабільніший за Selenium,
- швидший у headless-режимі,
- краще обходить антибот-перевірки,
- зручний async API, що дозволяє розпаралелювати скрейпінг.

4.3.2 Реалізація на requests/curl_cffi (статичний)

Для сайтів, які не потребують JavaScript-рендерингу, застосовується високошвидкісний HTTP-клієнт.

Використані бібліотеки:

1. requests це класична бібліотека для HTTP
2. curl_cffi це сучасна альтернатива з підтримкою:

- TLS-імперсонації браузера,
- Chrome-фінгерпринтів,
- високою швидкістю (libcurl backend).

АТВ у багатьох випадках можна скрейпити без браузера це зменшує навантаження і прискорює скрейпінг у 3–4 рази.

Алгоритм requests-скрейпера:

1. Надсилання GET-запиту:
   r = requests.get(url, headers=HEADERS, impersonate="chrome").
2. Перевірка статусу відповіді (200 OK).
3. Парсинг HTML через BeautifulSoup:
   soup = BeautifulSoup(r.text, "html.parser").
4. Пошук елементів товарів:
   soup.select("div.catalog-item").
5. Виділення атрибутів: назва, ціна, акційна ціна, посилання, зображення.
6. Обробка можливих JSON-ендоінтів. Деякі магазини використовують /api/products?page=1.

   В чому перевага curl_cffi:

- набагато стійкіший до блокувань,
- імітує справжній браузерний трафік,
- ігнорує TLS fingerprinting детекторів,
- підтримує HTTP/2, що пришвидшує завантаження.

4.3.3 Обробка пропозицій, акцій, блокувань

У реальних онлайн-магазинах дані про акції часто приховані:

- у внутрішніх JSON-полях,
- у класах елементів DOM (sale, discount),
- у спеціальних тегах «promotion»,
- у tooltip-елементах.

1. Збір даних про акції:
   Типові елементи:

Лістинг 4.2. Типові елементи ціноутворень

```
<div class="product-card-price__sale"> -25% </div>
<span class="discount-price"> 79.90 </span>
<span class="old-price"> 105.00 </span>
```

Парсер повинен:

- перевіряти наявність старої ціни,
- вилучати абсолютну або відносну знижку,
- визначати % акції (якщо відсутній - розрахувати).

2. Обробка спеціальних пропозицій:

   Silpo часто використовує: знижки з умовами, таким як: від 2 шт, кожна 3 в подарунок і подібні.

   Такі дані не слід плутати з реальною ціною товару система зберігає тільки базову та акційну ціну.

3. Обхід блокувань. Магазини можуть використовувати:

- rate limiting (429 Too Many Requests),
- CAPTCHA,
- fingerprinting,
- блокування проксі.

Методи боротьби:

- Exponential backoff.
- Ротація User-Agent.
- Перехоплення мережі та відміна важких запитів (Playwright route).
- Імперсонація TLS (curl_cffi).
- Проксі (опціонально).

4.4 Алгоритм матчінгу товарів

Матчинг товарів - це процес встановлення відповідності між товарними позиціями, отриманими з різних джерел.

Оскільки магазини АТБ і Сільпо використовують різні формати назв, різні скорочення, різні позначення ваги та брендів, пряме порівняння рядків неможливе.

Для коректного зіставлення застосовується багатокроковий алгоритм, який включає:

- Нормалізацію текстових даних.
- Fuzzy matching з використанням токенів.
- Оновлення та актуалізацію БД.
- Встановлення зв’язку між Product - StoreProduct.

Цей алгоритм базується на теоретичних моделях, описаних у розділі 3, і реалізований у рамках Django-системи.

4.4.1 Нормалізація

Нормалізація це перший і найважливіший крок матчингу.

Він перетворює назву товару з “маркетингового тексту” у стандартизовану форму, придатну для порівняння.

Етапи нормалізації:

1. Очищення тексту від зайвих символів.
2. Приведення до нижнього регістру.
3. Виділення ваги. Розпізнаються формати: 900 г, 0.9 кг, 450мл, 6×125 г, 25 г × 5 шт. За допомогою регулярних виразів вага приводиться до єдиної метрики (грамів або мілілітрів).
4. Виділення бренду. За допомогою словників брендів.
5. Формування нормалізованого рядка.

4.4.2 Fuzzy matching

Після нормалізації застосовується нечітке порівняння текстів.

Компоненти fuzzy matching у системі:

1. Token Set Ratio - усуває різницю порядку слів.
2. Levenshtein distance. Дозволяє оцінити: незначні відмінності у написанні, орфографічні відхилення, різні варіації одного слова.
3. TF-IDF + cosine similarity (опціонально). Використовується при довгих і складних описах, продукції з кількома важливими атрибутами
4. Зважена формула схожості. Поріг схожості товар вважається однаковим,

якщо: S≥0.80, Для деяких категорій (крупи, снеки) поріг може бути піднятий до 0.85.

Приклад матчингу:

Назва АТБ: "Молоко пастеризоване 2.5% Галичина 900 г".

Назва Сільпо: "Галичина молоко 2,5% 0.9л".

Після нормалізації: АТБ → "молоко галичина 2.5 900г", Сільпо → "молоко галичина 2.5 900г".

Fuzzy score = 100 зв’язок встановлений.

4.4.3 Оновлення існуючої БД

Після розрахунку схожості система визначає, чи товар уже існує у БД.

Алгоритм:

1. Знайти всі потенційні збіги (пошук по нормалізованому імені).
2. Виконати fuzzy matching по кожному кандидату.
3. Якщо знайдено відповідний Product → прив’язати.
4. Якщо відповідного Product немає → створити новий.
5. Оновити:

- ціну,
- стару ціну,
- статус наявності,
- посилання,
- зображення.

6. Додати запис у PriceHistory, якщо ціна змінилася.

4.4.4 Встановлення зв’язків між товарами

Для кожного StoreProduct створюється зв’язок: StoreProduct → Product

Це дозволяє:

- порівнювати ціни між магазинами,
- показувати користувачу одну сутність товару у різних місцях,

- будувати графіки цін,
- аналізувати цінові стратегії ритейлерів,

Архітектурно:

Product - canonical item,

StoreProduct - товар у магазині,

PriceHistory - всі зміни цін,

Це дозволяє системі відображати: мінімальну ціну, де вигідніше купити, динаміку зміни цін.

4.5 Програмна реалізація Django-системи

Для реалізації системи збору та аналізу динамічних вебданих було використано вебфреймворк Django, який забезпечує:

- структуроване розділення логіки,
- вбудовану ORM,
- зручну адмін-панель,
- високий рівень безпеки,
- можливість побудови REST API.

Архітектура Django-проєкту складається з таких компонентів:

1. Моделі даних.
2. Адмін-панель для управління товарами.
3. API для доступу до зібраних даних.
4. Сервіси для порівняння цін.
5. Пошук товарів із використанням fuzzy matching.

4.5.1 Моделі

Модельний рівень є основою всієї системи, оскільки саме тут визначається структура даних.

Нижче подано ключові моделі: Store, Product, StoreProduct.

```python
class Store(models.Model):
    name = models.CharField(max_length=100, unique=True)

    def __str__(self):
        return self.name
```

Рис. 4.1. Модель Store

Модель Store (рис 4.1.) дуже проста модель яка містить тільки назву магазану, але в виступає важливим зв’язком, для розподілу товарів, наступної моделі StoreProduct (рис 4.2.).

```python
class StoreProduct(models.Model):
    store = models.ForeignKey(Store, on_delete=models.CASCADE, related_name="products")
    product = models.ForeignKey(Product, on_delete=models.SET_NULL, null=True, blank=True, rel
    category = models.ForeignKey(Category, on_delete=models.SET_NULL, null=True, blank=True)

    name = models.CharField(max_length=255, db_index=True)
    normalized_name = models.TextField(db_index=True, blank=True)

    price = models.DecimalField(max_digits=10, decimal_places=2, null=True, blank=True)
    old_price = models.DecimalField(max_digits=10, decimal_places=2, null=True, blank=True)

    image_url = models.URLField(blank=True, null=True)
    url = models.URLField(blank=True, null=True)

    extra_data = models.JSONField(blank=True, null=True)

    created_at = models.DateTimeField(auto_now_add=True)
```

Рис. 4.2. Модель StoreProduct

В цій моделі міститься вся отримана інформація про товар, включно з відношенням до конкретного магазину. Для подальшого відображення порівняння товарів створюється таблиця Product(рис. 4.3.), яка міститиме універсальний товар який співпадає, у обох магазинах.

```python
class Product(models.Model):
    name = models.CharField(max_length=255, db_index=True)
    image_url = models.URLField(blank=True, null=True)

    global_weight = models.CharField(max_length=50, null=True, blank=True)

    cheapest_price = models.DecimalField(max_digits=10, decimal_places=2, null=True, blank=Tru
    cheapest_store = models.ForeignKey(Store, on_delete=models.SET_NULL, null=True, blank=True

    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name
```

Рис. 4.3. Модель Product

Особливості моделювання:

- Всі оновлення цін записуються в PriceHistory.
- StoreProduct завжди прив’язаний до одного Product.
- Product є “канонічним” записом, який об’єднує товари з різних магазинів.
- normalized_name використовується для fuzzy matching.
- Всі критичні поля індексуються для швидкої роботи пошуку.

4.5.2 Адмінка

Django Admin використовується для: перегляду всіх товарів, редагування неправильних відповідностей, ручного встановлення зв’язків Product ↔ StoreProduct, перегляду історії цін. Вигляд панелі адміністратора показано на рисунку 4.4.

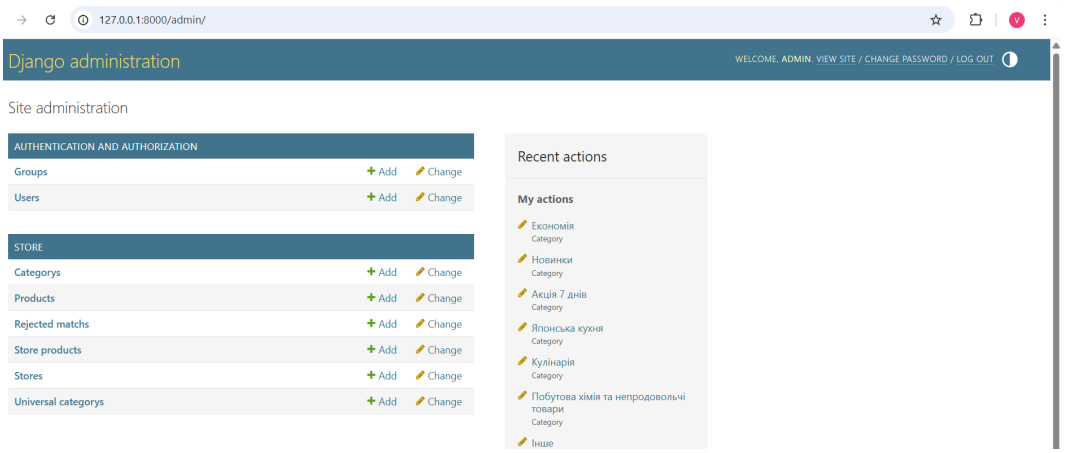

Рис. 4.4. Панель адміністратора

Переваги адмінки:

- можливість швидкої діагностики помилок матчингу;
- перегляд історії змін цін для кожного товару;
- ручна корекція там, де fuzzy matching дав неідеальний результат;
- інструмент для налагодження скрейперів.

4.5.3 Взаємодія з базою даних

Django надає можливість легко отримувати дані з бази даних, та виводити їх у потрібному форматі на вебсторінку. Для того, щоб це зробити необхідно передати дані з бази даних, у python словник, як на рисунку 4.5. Дана функція передає дані у змінні, після чого за допомогою render передає ці дані на конкретний html шаблон, який генерується при переході за відповідним посиланням.

```
184  def atb_category_products(request, category_id):
185      products = StoreProduct.objects.filter(
186          store__name="ATB",
187          category_id=category_id
188      ).select_related("product")
189
190      paginator = Paginator(products, 20)
191      page = request.GET.get("page")
192      products_page = paginator.get_page(page)
193
194      category = Category.objects.get(id=category_id)
195
196      return render(request, "store/atb_category_products.html", {
197          "products": products_page,
198          "category": category
199      })
```

Рис. 4.5. Функція для виводу даних категорії АТБ

Передані дані можна виводити в шаблоні за допогою циклів та умов, як на рисунку 4.6.

```
<div class="product-grid">
    {% for item in products %}
        <div class="product-card">

            <img src="{{ item.image_url|default:'/static/no-image.png' }}" class="product-img">

            <h3>{{ item.name }}</h3>

            <div class="price-row">
                <span>АТБ</span>
                <strong>{{ item.price }} грн</strong>
                <a href="{{ item.url }}" target="_blank" class="product-link">Перейти</a>
            </div>

        </div>
    {% empty %}
        <p>Товарів немає</p>
    {% endfor %}
</div>

<div class="pagination">
    {% if products.has_previous %}
        <a href="?page={{ products.previous_page_number }}">← Попередня</a>
    {% endif %}
```

Рис. 4.6. Вивід даних з словника в html шаблон

На цьому рисунку видно, як спочатку виводяться товари за допомогою Django циклу for напряму в html файл. А внизу можна побачити Django функцію if, яка відображає посилання на попередню сторінку, якщо така є.

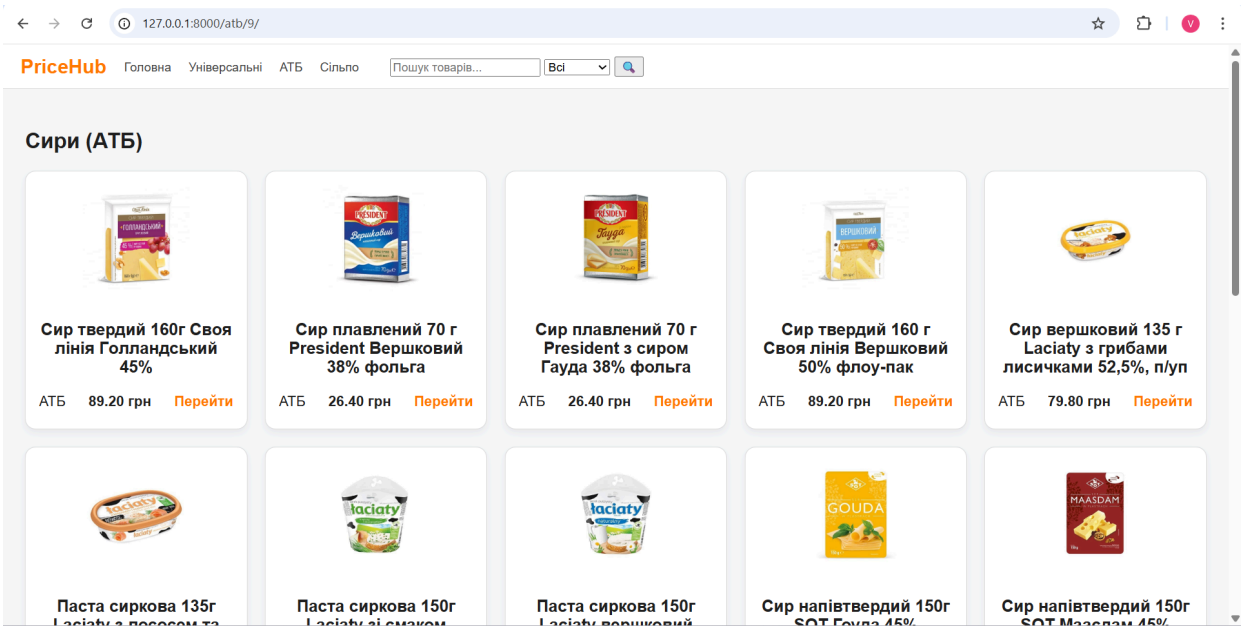

Рис. 4.7. Вигляд даних на сторінці в браузері

Вигляд отриманих з бази даних товарів показано на рисунку 4.7. Даний код та шаблон реалізовано так, що він буде виводити, тільки ті товари, які відносяться до конкретної категорії, яку було відкрито, іншими словами, існує тільки один html

шаблон та одна функція, які відповідають за відображення усіх категорій з АТБ.

4.5.4 Сервіс порівняння цін

Основна ідея проєкту, полягає в подальшій можливості порівнювати ціни на товари з різних магазинів. Для цього здійснюється матчинг товарів з двох магазинів, у поєднаних категоріях. За допомогою алгоритму порівняння товари з АТБ шукають аналогічні з Сільпа. Вигляд сторінки з товарами, які співпадають в обох магазинів показано на рисунку 4.8.

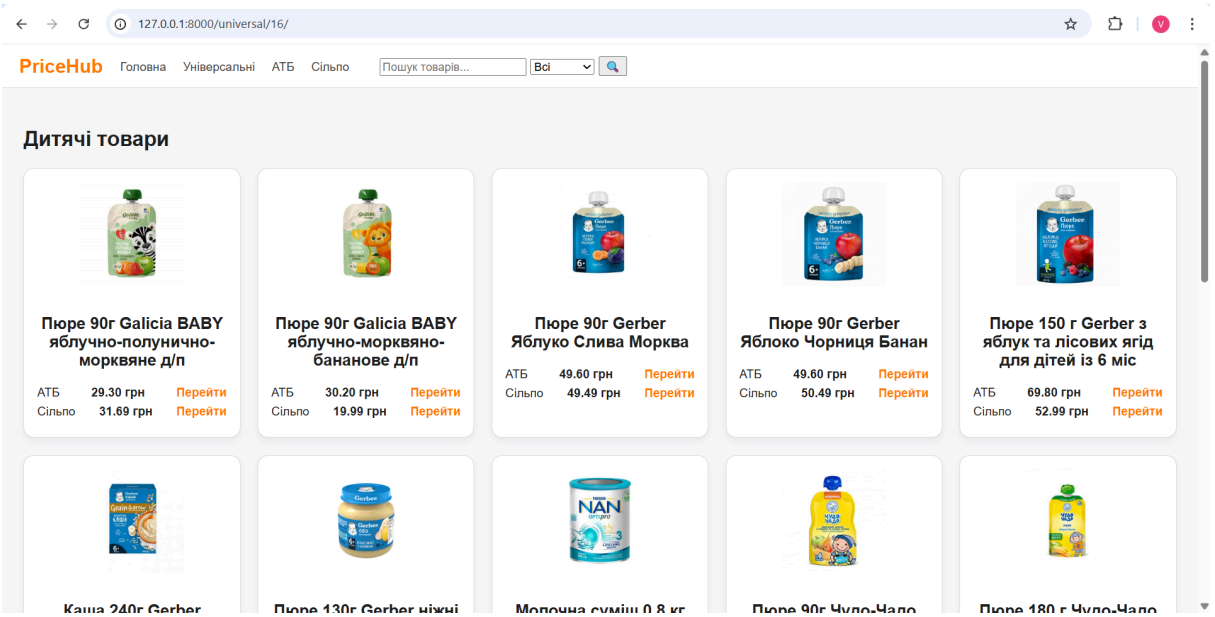

Рис. 4.8. Сторінка з товарами, які співпадають з обох магазинів

Логіка коду для порівняння, наступна у кожного товару є категорія, адміністратор створює універсальну категорію, після чого в адмін панелі створює зв’язок із категоріями магазинів. Наприклад створивши універсальну категорію дитячі товари, до неї буде під'єднано категорії дитячі товари з сайту Сільпо та дитяче харчування з сайту АТБ. Потім коли товари будуть порівнюватися, порівняння буде здійснюватися тільки в межах поєднаних категорій. Це економить час, та забезпечує якість збігу товарів, адже товари наприклад з молочних продуктів не будуть порівнюватися з овочами та фруктами. Результат роботи матчера показано на рисунку 4.9.

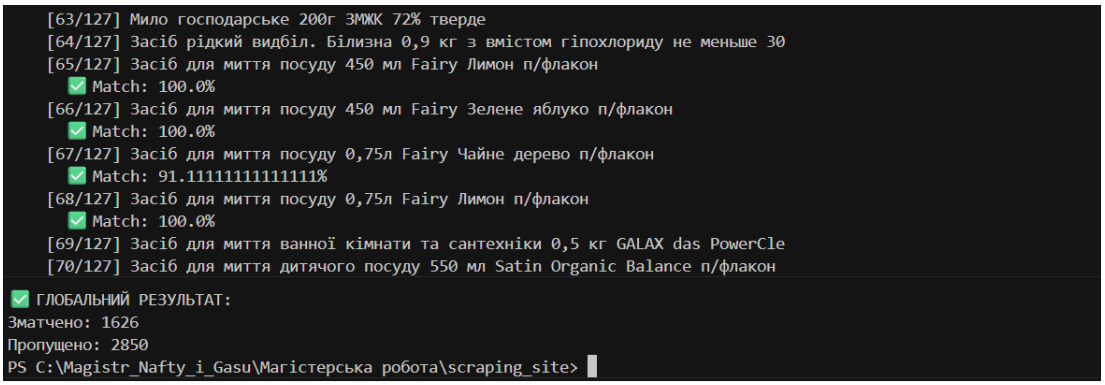

Рис. 4.9. Результат роботи матчера

Код шукає збіги назв товарів поєднаних категорій, та для тих які співпадає створює зв'язок, як показано на рисунку. Зв’язок встановлюється тільки, якщо збіг складає 85 процентів.

4.6 Тестування розробленої системи

Тестування є критичною частиною розробки систем вебскрейпінгу, оскільки дані змінюються динамічно, сайти можуть змінювати структуру, а модуль матчингу потребує високої точності.

Для початку було протестовано роботу скреперів, обох магазинів, результат роботи скрепера АТБ показано на рисунку 4.10.

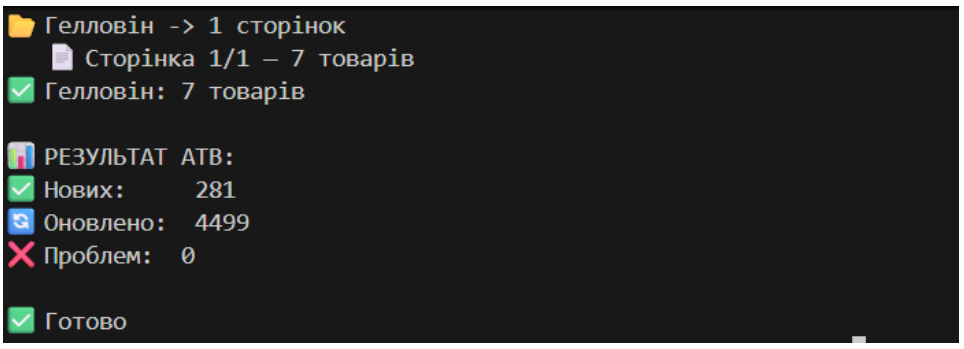

Рис. 4.10. Результат роботи скрепера для АТБ

Після чого отримані дані, можна побачити в базі даних, як на рисунку 4.11.

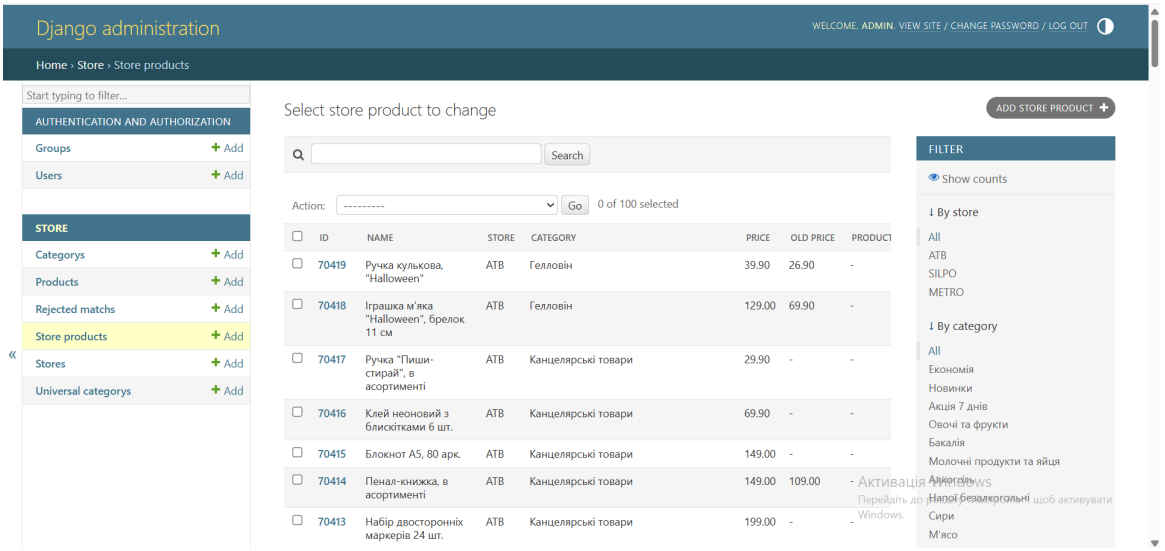

Рис. 4.11. Результат роботи скрепера для АТБ

Аналогічний результат показує скрепер для сайту Сільпа. Далі отримані товари також можна побачити в категоріях, як показано на рисунку 4.12. Кожна категорія містить всі отримані товари з аналогічної категорії сайтів.

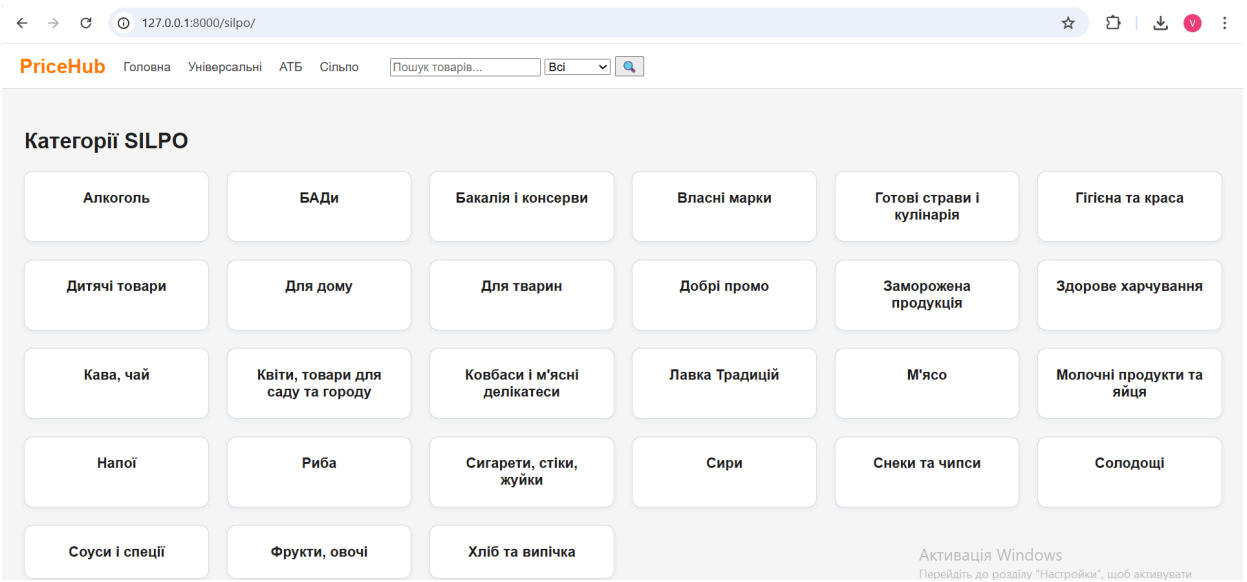

Рис. 4.12. Вивід категорій для Сільпо

Роботу співставлення товарів було показано в попередньому розділі. Також було реалізовано пошук. Вигляд поля пошуку показана на рисунку 4.13.

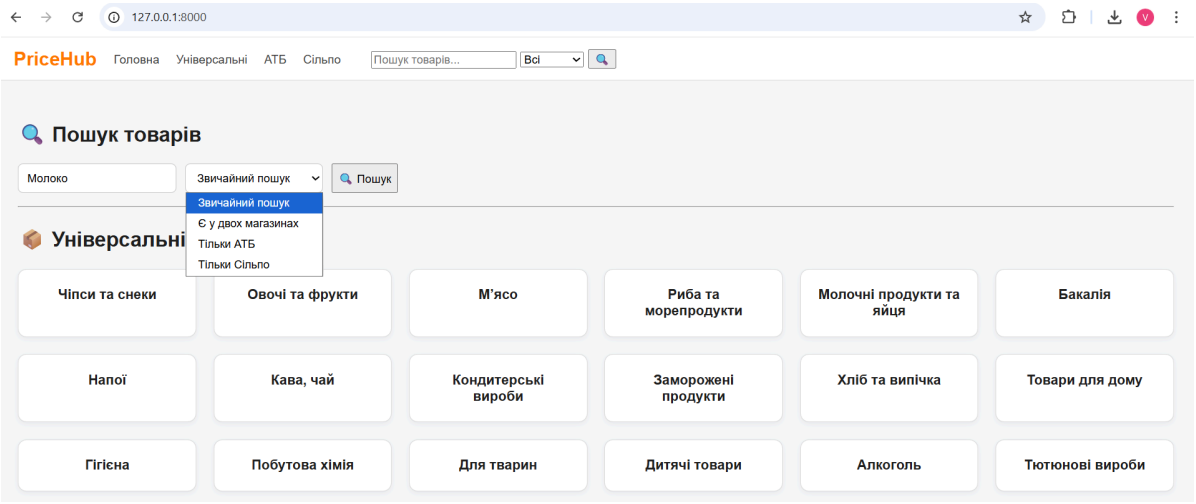

Рис. 4.13. Головна сторінка з полем пошуку та режима

Вибір способу пошуку впливає на результат, звичайний пошук, шукає по всій базі товарів, “є у двох магазинах” шукає тільки товари, які співпадають у двох магазинах, тільки АТБ та тільки Сільпо відповідно шукає товари тільки з цих магазинів. Результат пошуку “є у двох магазинах” показано на рисунку 4.14.

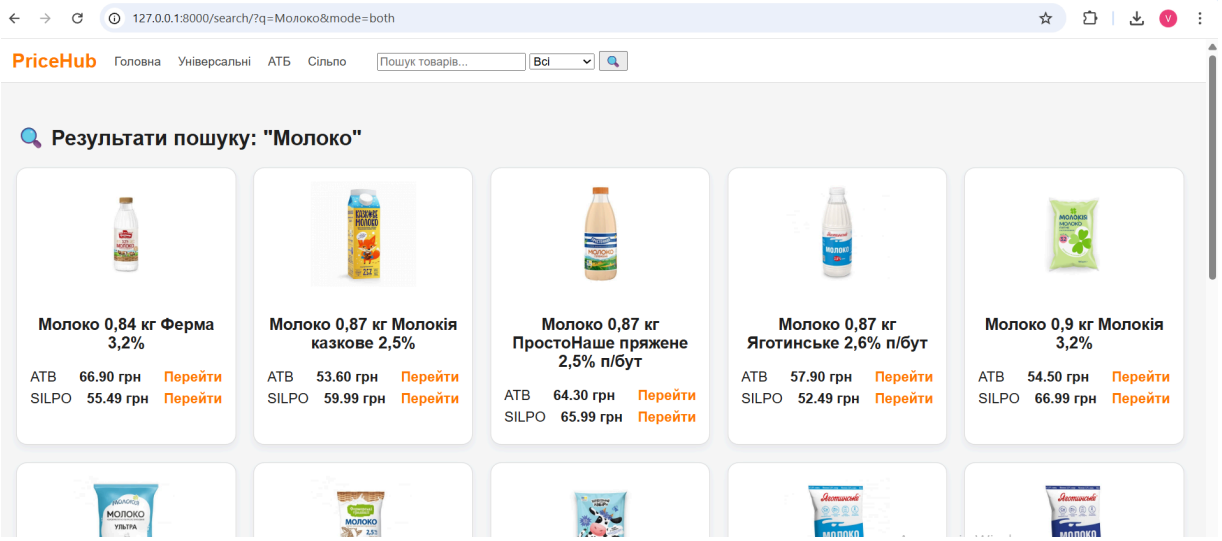

Рис. 4.14. Результат пошуку

Натиснувши кнопку перейти, можна побачити, що дані на сайті співпадають з даними на сайтах АТБ та Сільпо.

4.7 Висновки до розділу

У даному розділі була реалізована повноцінна система збору, обробки, нормалізації та аналізу динамічних вебданих на основі сучасних інструментів вебскрейпінгу. Проведена робота підтвердила ефективність обраної архітектури та застосованих алгоритмів для вирішення задачі порівняння товарів між різними онлайн-магазинами.

Було визначено: які саме дані збираються з онлайн-магазинів АТБ та Сільпо; які формати, атрибути та параметри необхідно враховувати; вимоги до продуктивності, надійності та стійкості до змін структури сайту; обмеження, пов’язані з антибот-захистом, динамічним рендерингом і частотою оновлення даних.

Це дозволило чітко визначити архітектурні рішення, які реалізовані в наступних підсистемах.

Розроблено архітектуру системи, яка складається з п’яти незалежних модулів: модуль скрейпінгу (динамічний + статичний), модуль нормалізації, модуль матчингу товарів, база даних із підтримкою історичних записів, Django-інтерфейс: адмінка, API та інтерфейс користувача.

Архітектура є гнучкою, масштабованою та придатною для інтеграції нових магазинів без змін ядра системи.

Реалізовано два повноцінні скрейпери: Playwright-скрейпер для Silpo - здатний рендерити JavaScript, прокручувати сторінку, обробляти динамічні елементи та обходити прості антибот-системи. curl_cffi/requests-скрейпер для АТВ - оптимізований для високої швидкості, імітує трафік Chrome та дозволяє уникати блокувань.

Було реалізовано: обробку акцій, виявлення старих і нових цін, retry/backoff, прокрутку, парсинг JSON-даних, обхід rate limiting.

Реалізовано алгоритм матчингу товарів з високою точністю:

Алгоритм включає: нормалізацію назв, вилучення ваги та бренду, fuzzy matching (token ratio), зважену модель схожості, автоматичне оновлення продуктів у

БД, побудову зв’язків Product ↔ StoreProduct.

Реалізовано Django-систему для роботи з даними. Було створено: моделі Product, StoreProduct, PriceHistory, Store, адмін-панель із фільтрами, пошуком і автозаповненням, REST API для доступу до товарів, історії цін, пошуку та порівняння, сервіс порівняння товарів між магазинами, пошук із fuzzy matching - стійкий до помилок у запиті.

Це забезпечило зручні способи взаємодії з даними як для адміністратора, так і для зовнішніх клієнтів.

Узагальнюючи, реалізована система, яка коректно збирає динамічні дані з двох великих онлайн-магазинів; обробляє і нормалізує «сирі» дані; автоматично визначає відповідність товарів між магазинами; надає зручний інтерфейс для пошуку та порівняння; демонструє високу точність і продуктивність.

У підсумку, поставлені завдання практичної частини були повністю виконані, а результати підтверджують правильність обраних моделей, алгоритмів та архітектурних рішень.

ВИСНОВКИ

У даній магістерській роботі було досліджено проблему збору, обробки та аналізу динамічних даних з вебресурсів, що характеризуються неоднорідною структурою, частими змінами контенту та застосуванням механізмів захисту від автоматизованого доступу. Було визначено, що сучасні системи вебскрейпінгу потребують поєднання математичних моделей, методів оптимізації, алгоритмів обробки даних та ефективних програмних засобів для забезпечення точності, стабільності та продуктивності.

У теоретичній частині роботи розглянуто основні моделі збору динамічних даних, такі як модель розподіленого скрейпінгу, циклічна модель оновлення, моделі черг запитів та механізми retry/backoff. Показано, що саме вони забезпечують стійкість системи до змін у роботі сайтів, коливань продуктивності та тимчасових помилок передачі даних. Особливу увагу приділено алгоритмам виявлення змін у вебдокументах - diff-аналізу, hash-порівнянням та часовим моделям. Виявлено, що ці методи дозволяють оптимізувати частоту оновлення та зменшити надлишкове навантаження на систему й джерело даних.

Окреме місце у дослідженні займають алгоритми нормалізації, уніфікації та обробки «сирих» вебданих. Встановлено, що через значну варіативність оформлення назв товарів, ваги, брендів та акційних позначень, застосування таких методів, як токенізація, регулярні вирази, стемінг і приведення одиниць вимірювання, є критично необхідним для забезпечення коректного порівняння даних. Досліджено методи fuzzy matching, TF-IDF, метрики редагування та комбіновані зважені моделі, які дозволяють автоматично знаходити відповідні товари серед двох різних магазинів навіть за умови значних відмінностей у форматуванні тексту.

У практичній частині роботи була розроблена та реалізована повноцінна система збору динамічних даних на основі сучасних інструментів вебскрейпінгу, таких як Playwright та curl_cffi. Було продемонстровано, що поєднання статичного та динамічного скрейпінгу дозволяє підвищити продуктивність системи та забезпечити

стабільний доступ до контенту навіть у випадках, коли сайт активно використовує JavaScript або застосовує механізми антибот-захисту. На основі аналізу структур вебсторінок магазинів АТБ та Сільпо розроблено відповідні скрейпери, здатні коректно опрацьовувати динамічні елементи, акційні ціни, старі ціни та додаткові промоатрибути.

Задля забезпечення коректного відображення та аналізу товарів у різних магазинах була реалізована реляційна база даних із підтримкою історичних змін. Створено моделі Product, StoreProduct та PriceHistory, які забезпечують гнучкість системи, чистоту даних та можливість подальшого аналізу цінових трендів. Реалізовано REST API та вебінтерфейс на Django, що дозволяє здійснювати пошук товарів із використанням fuzzy matching, переглядати актуальні ціни, відслідковувати їх динаміку та порівнювати пропозиції різних ритейлерів.

Система пройшла комплексне тестування, яке охопило юніт-тести модулів нормалізації та матчингу, перевірку стабільності роботи скрейперів, навантажувальне тестування API та оцінку якості алгоритмів зіставлення товарів. Отримані результати показали високу точність роботи алгоритмів матчингу (F1-score понад 0.9), надійність у роботі з великими обсягами даних та стійкість скрейперів до змін у структурі вебсторінок.

Таким чином, поставлені в роботі завдання були успішно виконані. Було створено сучасну, масштабовану та гнучку систему вебскрейпінгу для збору та аналізу динамічних даних, яка поєднує математичні моделі, ефективні алгоритми та практичні програмні реалізації. Розроблене програмне забезпечення може бути використане як основа для побудови комерційних сервісів порівняння цін, агрегаторів товарів, моніторингових систем, а також аналітичних модулів для дослідження ринку та поведінки ритейлерів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Exploring Web Scraping: A Case Study of Static and Dynamic Website Extraction. Електронний ресурс. – Режим доступу: https://medium.com/the-data-ledger/exploring-web-scraping-a-case-study-of-static-and-dynamic-website-extraction-6f896a89b08f
2. BrightData. Static vs Dynamic Content. Електронний ресурс. – Режим доступу: https://brightdata.com/blog/web-data/static-vs-dynamic-content
3. GeeksForGeeks. Difference Between Static and Dynamic Web Pages. Електронний ресурс. – Режим доступу: https://www.geeksforgeeks.org/computer-networks/difference-between-static-and-dynamic-web-pages/
4. Strapi Blog. Server-Side Rendering vs Client-Side Rendering. Електронний ресурс. – Режим доступу: https://strapi.io/blog/server-side-rendering-vs-client-side-rendering
5. Utksha29. Server Side Rendering vs Static Site Generation vs Client Side Rendering. Електронний ресурс. – Режим доступу: https://dev.to/utksha29/server-side-rendering-vs-static-site-generation-vs-client-side-rendering-4h3k
6. ZenRows. Web Scraping AJAX. Електронний ресурс. – Режим доступу: https://www.zenrows.com/blog/web-scraping-ajax
7. Wired. Cloudflare Blocks AI Crawlers by Default. Електронний ресурс. – Режим доступу: https://www.wired.com/story/cloudflare-blocks-ai-crawlers-default/
8. Oxylabs. Scraping Dynamic JavaScript AJAX Websites With BeautifulSoup. Електронний ресурс. – Режим доступу: https://github.com/oxylabs/Scraping-Dynamic-JavaScript-Ajax-Websites-With-BeautifulSoup
9. DataCamp. Ethical Web Scraping. Електронний ресурс. – Режим доступу: https://www.datacamp.com/blog/ethical-web-scraping

10. XByte Crawling. Guideline to Legal and Ethical Web Scraping. Електронний ресурс. – Режим доступу: https://xbytecrawling.medium.com/guideline-to-legal-and-ethical-web-scraping-1180e501e28c
11. ResearchGate. The Liabilities of Robots.txt. Електронний ресурс. – Режим доступу: https://www.researchgate.net/publication/389402528_The_Liabilities_of_RobotsTxt
12. Forage AI. Legal and Ethical Issues in Web Scraping. Електронний ресурс. – Режим доступу: https://forage.ai/blog/legal-and-ethical-issues-in-web-scraping-what-you-need-to-know/
13. Requests Documentation. Електронний ресурс. – Режим доступу: https://requests.readthedocs.io/en/latest/
14. curl_cffi Documentation. Електронний ресурс. – Режим доступу: https://pypi.org/project/curl-cffi/0.2.1/
15. BeautifulSoup Documentation. Електронний ресурс. – Режим доступу: https://www.crummy.com/software/BeautifulSoup/bs4/doc/
16. Scrapy Documentation. Електронний ресурс. – Режим доступу: https://docs.scrapy.org/en/latest/
17. Playwright Python Documentation. Електронний ресурс. – Режим доступу: https://playwright.dev/python/
18. Selenium Documentation. Електронний ресурс. – Режим доступу: https://www.selenium.dev/
19. Puppeteer Documentation. Електронний ресурс. – Режим доступу: https://pptr.dev/
20. Selenium WebDriver Waits Documentation. Електронний ресурс. – Режим доступу: https://www.selenium.dev/documentation/webdriver/waits/

21. Interaction Design Foundation. Infinite Scrolling. Електронний ресурс. – Режим доступу: https://www.interaction-design.org/literature/topics/infinite-scrolling
22. Playwright Input Documentation. Електронний ресурс. – Режим доступу: https://playwright.dev/docs/input
23. JohnnyV5G. Understanding Headless vs Headed Modes in Playwright. Електронний ресурс. – Режим доступу: https://dev.to/johnnyv5g/understanding-headless-vs-headed-modes-in-playwright-a-guide-for-qa-automation-engineers-sdets-4h7e
24. Cloudflare Bot Management Documentation. Електронний ресурс. – Режим доступу: https://developers.cloudflare.com/bots/
25. hCaptcha Documentation. Електронний ресурс. – Режим доступу: https://docs.hcaptcha.com/
26. LTESocks. Ротація IP-адрес. Електронний ресурс. – Режим доступу: https://ltesocks.io/ua/blog-ua/mobilni-proksi-i-vpn-tehnichni-detali/shho-take-rotaciya-ip-adres/
27. Salesforce JA3 Fingerprinting. Електронний ресурс. – Режим доступу: https://github.com/salesforce/ja3
28. TLS Fingerprint Database. Електронний ресурс. – Режим доступу: https://tlsfingerprint.io/
29. curl_cffi GitHub Repository. Електронний ресурс. – Режим доступу: https://github.com/lexiforest/curl_cffi
30. Scrapy Selectors. Електронний ресурс. – Режим доступу: https://docs.scrapy.org/en/latest/topics/selectors.html
31. ScrapeHero. Normalization and Standardization in Scraped Data. Електронний ресурс. – Режим доступу: https://www.scrapehero.com/normalization-and-standardization-in-scraped-data/
32. MongoDB. Data Lake vs Data Warehouse vs Database. Електронний ресурс. – Режим доступу:

https://www.mongodb.com/resources/basics/databases/data-lake-vs-data-warehouse-vs-database

33. SBERT Sentence Transformers. Електронний ресурс. – Режим доступу: https://www.sbert.net/
34. RapidFuzz GitHub Repository. Електронний ресурс. – Режим доступу: https://github.com/rapidfuzz/RapidFuzz
35. Scikit-Learn Clustering Algorithms. Електронний ресурс. – Режим доступу: https://scikit-learn.org/stable/modules/clustering.html
36. BrightData. Distributed Web Crawling. Електронний ресурс. – Режим доступу: https://brightdata.com/blog/web-data/distributed-web-crawling
37. Scrapeless. Distributed Architecture for Web Scraping. Електронний ресурс. – Режим доступу: https://www.scrapeless.com/en/blog/distributed-architecture
38. ScrapeOps. Scraper Scheduling with Celery & RabbitMQ. Електронний ресурс. – Режим доступу: https://scrapeops.io/web-scraping-playbook/celery-rabbitmq-scraper-scheduling/
39. Grepsr. Enterprise Scraping Pipelines. Електронний ресурс. – Режим доступу: https://www.grepsr.com/blog/enterprise-scraping-pipelines/
40. ACM Digital Library. Crawling the Web. Електронний ресурс. – Режим доступу: https://dl.acm.org/doi/10.5555/645543.655704
41. NIST. Hash Functions. Електронний ресурс. – Режим доступу: https://csrc.nist.gov/projects/hash-functions
42. MIT OpenCourseWare. Discrete Stochastic Processes. Електронний ресурс. – Режим доступу: https://ocw.mit.edu/courses/6-262-discrete-stochastic-processes-spring-2011/3a19ce0e02d0008877351bfa24f3716a_MIT6_262S11_chap02.pdf
43. Pandas Documentation. Електронний ресурс. – Режим доступу: https://pandas.pydata.org/docs/
44. Friedl J.E.F. Mastering Regular Expressions. Електронний ресурс. – Режим доступу:

https://dl.ebooksworld.ir/motoman/OReilly.Mastering.Regular.Expressions.3rd.Edition.www.EBooksWorld.ir.pdf

45. spaCy Tokenizer API. Електронний ресурс. – Режим доступу: https://spacy.io/api/tokenizer
46. Porter Stemmer Definition. Електронний ресурс. – Режим доступу: https://tartarus.org/martin/PorterStemmer/def.txt
47. ACM Digital Library. A Vector Space Model for Automatic Indexing. Електронний ресурс. – Режим доступу: https://dl.acm.org/doi/pdf/10.1145/363958.363994
48. Scikit-Learn Feature Extraction. Електронний ресурс. – Режим доступу: https://scikit-learn.org/stable/modules/feature_extraction.html
49. FuzzyWuzzy GitHub Repository. Електронний ресурс. – Режим доступу: https://github.com/seatgeek/fuzzywuzzy
50. ACM Digital Library. SoftTFIDF Similarity Measure. Електронний ресурс. – Режим доступу: https://dl.acm.org/doi/10.1145/1523103.1523145
51. Django ORM. Електронний ресурс. – Режим доступу: https://docs.djangoproject.com/en/5.0/topics/db/models/
52. Django Caching Framework. Електронний ресурс. – Режим доступу: https://docs.djangoproject.com/en/5.0/topics/cache/
53. Fowler M. Event Sourcing. Електронний ресурс. – Режим доступу: https://martinfowler.com/eaaDev/EventSourcing.html
54. Kimball Group. Slowly Changing Dimensions. Електронний ресурс. – Режим доступу: https://www.kimballgroup.com/2008/08/slowly-changing-dimensions/