МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проекту (роботи)
магістра
(ступінь вищої освіти (освітній ступінь))

на тему СИСТЕМА АВТОМАТИЗАЦІЇ ПРОЦЕСІВ ЗБОРУ ТА АНАЛІЗУ ІНФОРМАЦІЇ З ВЕБСАЙТІВ

Виконав: студент 2курсу, групи КНТ-513м
спеціальності
123 Комп'ютерна інженерія

Освітня програма (спеціалізація)
Комп'ютерні системи та мережі
(код і назва спеціальності)

ФЕДОРОВ В.А
(прізвище та ініціали)
Керівник ІЛЬЯШЕНКО М.Б.
(прізвище та ініціали)
Рецензент МАЛИЙ О.Ю.
(прізвище та ініціали)

2024 рік

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

(повне найменування вищого навчального закладу)

Факультет Комп'ютерних наук і технологій

Кафедра «Комп'ютерні системи та мережі»

Ступінь вищої освіти (освітній ступінь) магістерський

Спеціальність 123 Комп'ютерна інженерія

(код і назва)

Освітня програма (спеціалізація) Комп'ютерні системи та мережі

(назва)

ЗАТВЕРДЖУЮ

Зав. кафедри Кудерметов Р.К.

_____

"____" _____ 2024 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

ФЕДОРОВ Віталій Анатолійович

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Система автоматизації процесів збору та аналізу інформації з вебсайтів

керівник проекту (роботи) ІЛЬЯШЕНКО Матвій Борисович, к. т. н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "18" жовтня 2024 року № 149

2. Строк подання студентом проекту (роботи) 01 грудня 2024 року

3. Вихідні дані до проекту (роботи) Вебсайт, запит, СКБД, HTTP, HTML, UML

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

1) Аналіз проблем комп'ютерних систем для автоматизованого збору та обробки інформації з веб-ресурсів;

2) Дослідження інструментів для автоматизованого збору та обробки даних з веб-ресурсів;

3) Обгрунтування для розробки;

4) Дослідження результатів функціонування комп'ютерної системи для автоматизованого збору і аналізу інформації з вебресурсів

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

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3	ІЛЬЯШЕНКО М.Б., к. т. н., доцент		
Нормоконтроль	ЩЕРБАК Н.В., ст. викл.		

7. Дата видачі завдання 01.11.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз технічного завдання	10.11.2024 р.	
2	Вибір інструментів для виконання роботи	15.11. 2024 р.	
3	Проектування системи	17.11. 2024 р.	
4	Побудова системи	25.11. 2024 р.	
5	Дослідження результатів	01.12. 2024 р.	
6	Оформлення отриманих результатів у ПЗ	01.12. 2024 р.	
7	Оформлення графічного матеріалу	10.12. 2024 р.	
8	Захист роботи	16.12. 2024 р.	

Студент _____ **Віталій ФЕДОРОВ**
(підпис) (ініціали та прізвище)

Керівник проекту (роботи) _____ **Матвій ІЛЬЯШЕНКО**
(підпис) (ініціали та прізвище)

РЕФЕРАТ

ПЗ: 75 с., 31 рис., 7 лістин., 24 джерел.

HTTP, HTML, UML, ВЕБСАЙТ, ВЕБРЕСУРСИ, ЗАПИТ, СКБД

Мета та завдання дослідження полягають у створенні системи автоматизованого збору та аналізу інформації. Основними цілями є пошук аналогів на ринку та виявлення їх недоліків. Також передбачено дослідження варіантів використання синтаксичних аналізаторів для аналізу веб-сайтів та оцінка релевантності їх застосування. Планується вивчити етапи роботи подібних систем на ринку та досягти універсальності, пропонуючи не готовий застосунок, а конструктор запитів, який орієнтується на звичайних користувачів без професійних навичок.

У процесі дослідження було розглянуто питання переваг і недоліків програмних продуктів, що реалізують аналогічний функціонал. В результаті було розроблено програмний застосунок, основна ідея якого полягає в максимальній універсальності та багатофункціональності, щоб задовольнити якомога більше запитів звичайних користувачів. Цей програмний продукт призначений для надання можливості користувачам поетапно створювати запити, які в подальшому будуть автоматично виконувати пошук, аналізувати та зберігати дані в заданому користувачем форматі з обраного веб-сайту.

ABSTRACT

Explanatory note to the master’s work: 75 p.; 31 figures; 7 listings; 24 sources.

HTTP, HTML, UML, WEBSITE, WEBRESOURCEY, QUERY, DBD,

The aim and objectives of the study are to create a system for automated data collection and analysis. The main objectives are to search for analogues in the market and identify their shortcomings. The study also envisages exploring options for using parsers to analyse web resources and assessing the relevance of their use. It is planned to study the stages of operation of similar systems on the market and achieve universality by offering not a ready-made application, but a query builder that is aimed at ordinary users without professional skills.

The study examined the advantages and disadvantages of software products that implement similar functionality. As a result, a software application was developed, the main idea of which is to maximise versatility and multifunctionality in order to satisfy as many requests of ordinary users as possible. This software product is designed to enable users to create queries step by step, which will then automatically search, analyse and store data in a user-defined format from a selected web resource.

ЗМІСТ

Вступ 8
1 Аналіз проблем комп'ютерних систем для автоматизованого збору та обробки інформації з веб-сайтів.....12
1.1 Аналіз питань автоматизованого збору та обробки даних з вебресурсів 12
1.2 Аналіз процесу взаємодії користувача з вебресурсом..... 12
1.3 Аналіз типів вебзастосунків і механізмів пагінації вебсторінок 21
1.4 Аналіз теоретичної комп'ютерної системи для автоматизованого збору та обробки інформації з вебресурсів..... 24
1.5 Огляд наявних рішень..... 28
1.5.1 ParseHub..... 28
1.5.2 Puppeteer 29
1.5.3 Scrapy 30
1.6 Висновки до розділу..... 33
2 Дослідження інструментів для автоматизованого збору та обробки даних з веб-сайтів 34
2.1 Засоби для розробки програм автоматизованого збору та обробки даних..... 34
2.2 Бібліотеки та інструменти, застосовані в системі автоматизованого збору й аналізу даних..... 36
2.2.1 Основні характеристики HttpWebRequest..... 37
2.2.2 Основні характеристики HtmlAgilityPack 38
2.2.3 Основні характеристики Windows Presentation Foundation 39
2.3 Обгрунтування для розробки 41
3 Створення комп'ютерної системи для автоматизованого збору та аналізу даних з веб-сайтів 42
3.1 Призначення системи, вибір сервісів для реалізації 42
3.2 Вимоги до системи 43
3.3 Діаграми використання у системі 45

3.3.1 Діаграми використання в статичній автоматизованій системі збору та аналізу даних з вебресурсів 45
3.3.2 Взаємодія користувача з запитами 46
3.4 Вимоги до користувацього інтерфуйсу 49
3.5 Діаграми виконання послідовностей 50
3.6 Діаграма класів 55
3.6.1 Діаграма класів у статистичній автоматизованій системі сбору та аналізу даних 55
3.6.2 Діаграми класів у динамічній автоматизованій системі збору та аналізу даних 56
3.7 Засоби реалізації системи 59
3.7.1 Модулі і алгоритми у динамічній автоматизованій системі збору та аналізу даних з веб-сайтів 62
3.8 Інтерфейс проєкту 65
3.8.1 Проектування інтерфейсу статичної системи автоматизованого збору та аналізу інформації 65
3.9 Висновки по розділу 68
4 Дослідження результатів функціонування комп'ютерної системи для автоматизованого збору і аналізу інформації з вебресурсів 69
4.1 Оцінка працездатності програмного застосунку для динамічного автоматизованого збору та аналізу інформації з вебресурсів 69
4.2 Висновки до розділу 71
Висновки 72
Перелік джерел посилання 73

ВСТУП

У сучасному світі, де інтернет став невід'ємною складовою нашого повсякденного життя, і майже кожна частина світу має доступ до глобальної мережі, виникає все більше нових викликів, які потребують вирішення. Одним із таких викликів є задача збору та аналізу інформації. Оскільки інтернет містить величезні обсяги даних, актуальним стає питання отримання впорядкованої та навіть базово проаналізованої інформації без безпосередньої участі людини в цьому процесі.

Рішення для автоматизованого збору та аналізу інформації дійсно існують, однак більшість програмних застосунків, що вирішують ці завдання, мають суттєвий недолік — відсутність гнучкості. Це означає, що коли структура веб-сторінки змінюється, процес автоматизації збору та аналізу інформації зазвичай припиняється. Але проблема негнучкості полягає не лише в цьому. Навіть якщо розробники стверджують, що їхня програма універсальна, часто це є помилковим твердженням, що вводить користувачів в оману [1].

Більшість таких програм являють собою велику кількість спеціалізованих синтаксичних аналізаторів, розроблених для конкретної групи веб-сайтів. Коли ж користувач потребує аналізу даних із менш відомого ресурсу, він стикається з тим, що цей ресурс не підтримується системою, і процес аналізу стає неможливим без втручання розробників. Вони можуть вирішити проблему шляхом створення нового парсера, хоча такий парсер лише частково інтегрується у "універсальну" систему.

У випадках, коли програмний застосунок дійсно є універсальним, він, як правило, представляє лише базову структуру парсера. Однак такий застосунок зазвичай не надає користувачу можливості створювати власні запити для збору інформації, що відповідають його потребам, без залучення фахівців або розробників. У більшості випадків для цього необхідно звертатися до

спеціалістів, які створюють нові синтаксичні аналізатори або налаштовують запити в межах програми.

Таким чином, хоча на ринку дійсно існують рішення для автоматизації збору інформації з веб-сайтів, вони або не є по-справжньому універсальними, або не надають користувачу можливості самостійно створювати й налаштовувати запити до ресурсів, які його цікавлять, без залучення додаткових спеціалістів.

У ситуаціях, коли програмний застосунок дійсно позиціонується як універсальний, насправді він зазвичай є лише основним каркасом для парсера. Однак такий застосунок часто не дозволяє користувачу самостійно формувати індивідуальні запити для збору даних, які б задовольнили його конкретні потреби, без залучення розробників або фахівців. У більшості випадків для виконання таких завдань необхідно звертатися до професіоналів, які створюють нові синтаксичні аналізатори або налаштовують необхідні запити в рамках наявної програми.

Отже, незважаючи на те, що на ринку існують рішення для автоматизації збору інформації з веб-сайтів, вони або не є повністю універсальними, або не пропонують користувачу можливості самостійного створення й налаштування запитів до ресурсів, які його цікавлять, без потреби у залученні додаткових фахівців.

Об'єктом дослідження є система автоматизованого збору та аналізу інформації з веб-сайтів. Зокрема, це стосується процесу розробки такої системи, механізмів взаємодії програмного застосунку з веб-сайтами, а також принципів обміну даними між веб-сайтом і веб-сервером. У ході дослідження розглядаються різні методи представлення інформації користувачеві, які використовують веб-сайти, а також вимоги до користувача, необхідні для забезпечення коректної роботи веб-сайту та надання послуг. Окрему увагу приділяється сценаріям, у яких веб-сайт може не надати потрібну користувачеві інформацію через певні технічні або програмні обмеження.

Предметом дослідження є визначення характеристик програмного забезпечення, створеного за допомогою автоматизованого конструктора запитів, а також аналіз особливостей типової програми для автоматизованого збору та аналізу даних з веб-сайтів.

Для вирішення поставленої задачі було застосовано низку методів дослідження [2]:

- аналіз існуючих рішень та їх особливостей - вивчення вже наявних систем для автоматизованого збору та аналізу інформації з веб-сайтів, з метою визначення їх характеристик, функціональних можливостей та обмежень;

- аналіз переваг та недоліків існуючих рішень - оцінка сильних та слабких сторін поточних систем збору та аналізу даних з веб-сайтів, для виявлення можливостей вдосконалення та оптимізації;

- аналіз різноманітних бібліотек, призначених для реалізації таких програмних застосунків – дослідження доступних програмних бібліотек та фреймворків, які використовуються для створення інструментів автоматизованого збору та аналізу веб-даних;

- аналіз різних інструментів, які призначені для створення таких програмних застосунків - оцінка інструментів розробки, таких як середовища програмування, мови, інтерпретатори та інші платформи, які можуть бути використані для побудови систем збору та аналізу інформації;

- експериментування з різними інструментами, які призначені для нагляду за статусом роботи систем для автоматичного збору та аналізу даних з веб-сайтів - проведення експериментальних досліджень за допомогою інструментів моніторингу та діагностики, які дозволяють відстежувати ефективність роботи систем збору даних;

- аналіз принципів роботи веб-сайтів для покращення автоматизованого збору та аналізу інформації - вивчення того, як веб-сайти взаємодіють із користувачами та системами автоматизованого збору даних, з метою покращення процесу збору та аналізу інформації.

Наукова новизна результатів цього дослідження полягає в тому, що під час розробки системи автоматизованого збору та аналізу інформації було досягнуто реальної універсальності застосунку. Основна особливість запропонованої системи полягає в її здатності гнучко адаптуватися до різних веб-сайтів, незалежно від їхньої структури чи змін у HTML-розмітці. Це забезпечується алгоритмом, який дозволяє програмному застосунку автоматично підлаштовуватись під різноманітні формати та структури веб-сторінок, усуваючи необхідність постійного залучення розробників для оновлення парсерів або налаштувань. Такий підхід надає можливість більш ефективно й автоматизовано отримувати та аналізувати дані з різних веб-сайтів, що раніше вимагало втручання фахівців.

Практичне значення одержаних результатів полягає в розробці програмного застосунку для автоматизованого збору та аналізу інформації з веб-сайтів, який вирішує основні проблеми існуючих на ринку рішень. На відміну від існуючих систем, які є вузькоспеціалізованими та не мають належної гнучкості, запропонований програмний застосунок надає користувачам можливість продовжувати роботу навіть у випадку змін у структурі веб-сайтів. Це означає, що користувач не втрачає доступ до аналізованої інформації при зміні HTML-розмітки веб-сторінок і не потребує залучення розробників для оновлення чи налаштування програмного забезпечення. Це значно зменшує час і витрати на підтримку роботи таких систем, роблячи їх більш універсальними та зручними у використанні.

1 АНАЛІЗ ПРОБЛЕМ КОМП'ЮТЕРНИХ СИСТЕМ ДЛЯ АВТОМАТИЗОВАНОГО ЗБОРУ ТА ОБРОБКИ ІНФОРМАЦІЇ З ВЕБ-САЙТІВ

1.1 Аналіз питань автоматизованого збору та обробки даних з веб-сайтів

Програмні рішення для автоматизованого збору та аналізу інформації з веб-сайтів призначені для надання користувачеві можливості отримувати структуровані дані з конкретних веб-сайтів. Для досягнення цієї мети такі програми імітують поведінку реального користувача: вони здійснюють запити з визначеними атрибутами, перевіряють статуси відповідей від веб-сайтів і виконують синтаксичний аналіз отриманих результатів. Щоб зрозуміти, які процеси відбуваються під час функціонування програмного застосунку, необхідно дослідити, як саме користувач отримує інформацію. Це також включає аналіз ролі веб-браузера у взаємодії користувача з веб-сайтами.

1.2 Аналіз процесу взаємодії користувача з веб-сайтом

Для отримання даних з веб-сайту користувачі використовують веб-браузер. Вони вводять у рядок пошуку URL-адресу цікавого ресурсу і натискають кнопку підтвердження. У цей момент відбувається наступний процес (див. рис. 1). Клієнт, яким є веб-браузер, надсилає HTTP-запит за вказаною адресою з певними параметрами, отримує відповідь, здійснює її обробку та форматує у зручний для користувача вигляд [3].

Рисунок 1.1 - Схема основного представлення взаємодії користувача з веб-сайтом

URL (Uniform Resource Locator) — це адреса, яка надана певному унікальному ресурсу у мережі інтернет. Такими URL-адресами (рис. 1..2) можуть слугувати HTML-сторінки, JS-скріпти, чи файли CSS. В наш час існує вірогідність, що URL-адреса може вести на ті ресурси, які більше не існують або були перенесені.

https://moodle.zp.ua
https://google.com
https://zp.edu.ua

Рисунок 1.2 - Приклади адрес сайтів

URL-адреса складається з кількох частин, деякі з яких є обов'язковими. Розглянемо їх на прикладах (рис.1.3).

Протокол `https://` вказує, який саме протокол передачі даних має використовувати веб-браузер. Найчастіше використовуються HTTP-протокол і HTTPS-протокол — його модифікована версія, орієнтована на безпеку даних [4]. Сьогодні дедалі більше веб-сайтів переходять на захищену версію протоколу.

Веб-браузери за замовчуванням підтримують обидва протоколи, тому якщо користувач не вказує протокол в адресному рядку, браузер автоматично підставляє його.

Доменне ім’я дозволяє користувачеві знаходити веб-сайт, не потребуючи запам'ятовувати IP-адресу. Замість цього користувач вводить зручну для запам'ятовування назву ресурсу, за якою веб-браузер звертається до DNS-сервера, отримуючи відповідну IP-адресу.

Протоколи функціонують у тісному зв'язку з портами. За замовчуванням HTTP-протокол використовує 80 порт, тоді як HTTPS-протокол працює на 443 порті. Ці значення є стандартними, тому в більшості випадків користувачеві не потрібно вказувати порти самостійно. Проте деякі веб-сайти можуть використовувати інші порти, і в таких випадках користувачеві слід вказати відповідний порт, щоб отримати потрібну інформацію.

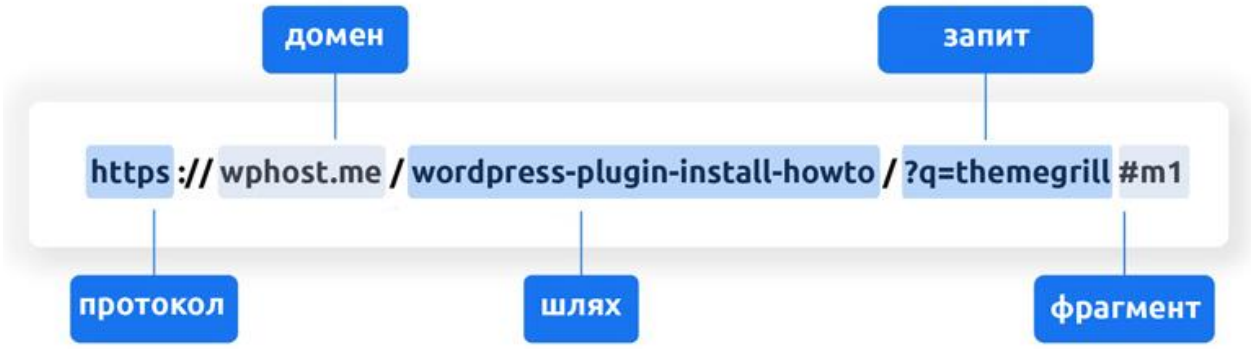


Рисунок 1.3 – Склад URL-адреси

Шлях до сторінки є частиною URL-адреси, що слідує після доменного імені (рис.1.3). Він забезпечує можливість користувачу перейти на конкретну сторінку веб-сайту. У простих реалізаціях веб-сайтів шляхи до сторінок часто відображають реальне розташування HTML- або CSS-файлів на веб-сервері. Головна сторінка зазвичай вказується лише через протокол і доменне ім’я.

Параметри - це інформація, яку веб-браузер передає веб-сайту (рис.1.4). Вони використовуються для того, щоб вказати, які додаткові дії має виконати веб-сайт перед тим, як надати відповідь.

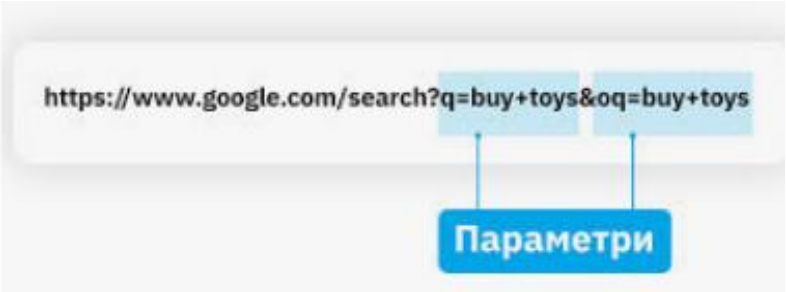


Рисунок 1.4 - Складова URL-адреси сайту

URL-адреси класифікуються на два типи: статичні та динамічні.

Статична URL-адреса - це незмінна адреса, яка вказує на конкретну сторінку. Як правило, такі адреси не містять параметрів [5].

Динамічні адреси формують контент на основі параметрів, наданих у запиті. Ці адреси зазвичай зустрічаються на сайтах, створених на основі CMS. Вміст таких сайтів зберігається в базі даних і виводиться за запитом. Динамічні адреси також формуються під час використання фільтрів чи пошуку на сайті.

Формати URL. Сучасні URL-адреси можна поділити на такі формати:

- латиниця;
- кирилиця;
- транслітеровані;
- зрозумілі людині адреси.

Коли користувач вводить URL-адресу в адресний рядок, веб-браузер розпочинає процес, що включає запит до DNS-сервера для отримання IP-адреси сервера, на якому розміщується вказаний веб-сайт [6].

Спочатку браузер відправляє запит до DNS-сервера з метою дізнатися IP-адресу, відповідну зазначеному доменному імені. Після того, як DNS-сервер надає браузеру цю IP-адресу, веб-браузер формує запит до сервера і надсилає його, очікуючи відповідь (рис. 1.5).

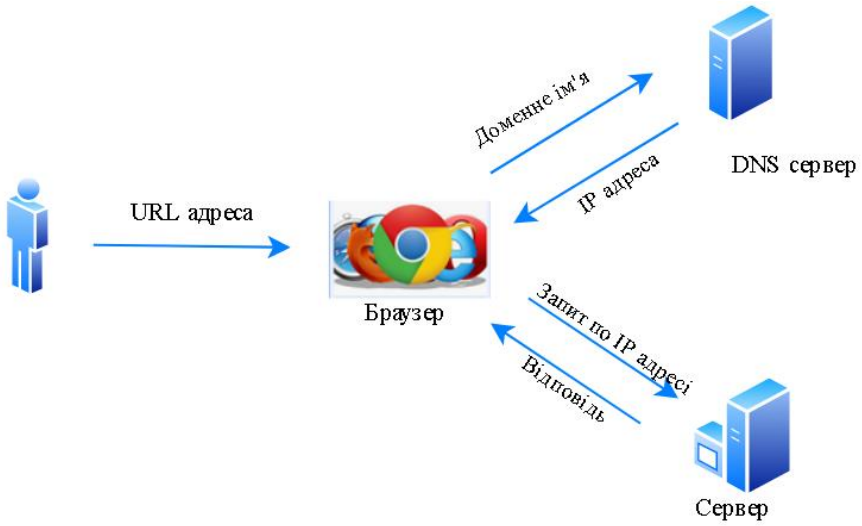


Рисунок 1.5 - Процес роботи веб-браузеру з URL

HTTP - це протокол для передачі гіпертекстових документів, заснований на взаємодії між клієнтом і сервером. Клієнт ініціює підключення і відправляє запит серверу [6]. Сервер, очікуючи на підключення, приймає запит, виконує необхідні дії відповідно до запиту та повертає клієнту відповідь з результатом. Найпоширенішими типами запитів, які використовують звичайні користувачі, є запити GET і POST.

GET-запит (рис. 1.6) призначений для отримання інформації з веб-сайту. Його ключовою особливістю є те, що він не повинен змінювати структуру або дані веб-сайту.

```
const Http = new XMLHttpRequest();
const url='https://jsonplaceholder.typicode.com/posts';
Http.open("GET", url);
Http.send();

Http.onreadystatechange=(e)=>{
    console.log(Http.responseText)
}
```

Рисунок 1.6 - Приклад Get запиту

POST-запит (рис.1.7) використовується для передачі даних на сервер. Зазвичай він надсилається через HTML-форми і призводить до змін на сервері.

```
const Url='https://jsonplaceholder.typicode.com/posts/';
const data={
    name:"Said",
    id:23
}

$('.btn').click(function(){
    $.post(Url,data, function(data, status){
        console.log(`${data} and status is ${status}`)
    });
})
```

Рисунок 1.7 - Приклад Post запиту

Заголовки дозволяють клієнту і серверу додавати додаткові дані до HTTP-запиту або відповіді. Інформація в заголовках передається у форматі: ім'я поля (не чутливе до регістру), після якого йде роздільник ":" і значення поля. Заголовки супроводжують процес обміну даними через HTTP-протокол [7] і можуть містити інформацію, опис даних або інші необхідні деталі для забезпечення коректної взаємодії між клієнтом і сервером.

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Host: www.example.com
```

Рисунок 1.8 - Приклад HTTP заголовка

HTTP-заголовки можна класифікувати за наступними категоріями:

- основні заголовки - це заголовки, які використовуються як у запитах, так і у відповідях. Вони не стосуються даних, що передаються в тілі запиту;

- заголовки запиту - містять інформацію про клієнта, ресурс, до якого виконується запит, або додаткові дані про цей ресурс;

- заголовки відповіді - надають інформацію про відповідь на запит, наприклад, про розташування ресурсів або веб-сервер;

- заголовки, пов'язані з тілом ресурсу - включають інформацію про довжину контенту, мову тексту або його кодування.

Протокол HTTP включає набір кодів відповіді (статусів), які вказують на те, чи був успішно виконаний запит (рис. 1.9). Ці коди поділяються на п'ять категорій:

- «100-199 Інформаційні» - вказують на те, що запит отримано і він валідний. Обробка продовжується;

- «200-299 Успішні» - запит був прийнятий, оброблений і виконаний успішно;

- «300-399 Перенаправлення» - для завершення виконання запиту необхідно вжити додаткових дій;

- «400-499 Помилки клієнта» - запит не може бути виконаним через помилки, наприклад, некоректний синтаксис;

- «500-599 Помилки сервера».

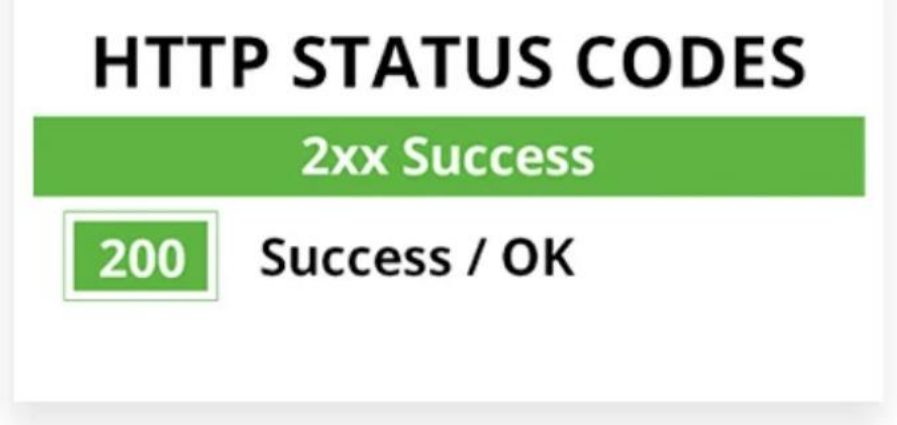


Рисунок. 1.9 - Приклад статус коду

Існує ймовірність, що сервер може надіслати код відповіді, який не входить до цього переліку. У такому випадку цей код є нестандартизованим і не відповідає офіційним специфікаціям HTTP.

Після обробки запиту клієнта веб-сайт надсилає йому відповідь, яка складається з кількох основних елементів:

- рядок статусу (Status line) - це перший рядок у відповіді HTTP, який містить інформацію про версію протоколу, код стану запиту та текстове пояснення статус-коду;

- HTTP-заголовки - мають таку ж структуру, як і в HTTP-запиті, і містять додаткову інформацію про відповідь;

- тіло відповіді - не всі відповіді HTTP містять тіло, наприклад, відповідь зі статус-кодом 204 (Немає контенту) не має вмісту.

Кожен HTML-документ починається з декларації типу документа (DOCTYPE). Вона потрібна для того, щоб веб-браузер знав, яку версію HTML використовувати (рис.1.10), що дозволяє правильно відобразити сторінку.

```
<!DOCTYPE html>

<html  dir="ltr" lang="uk" xml:lang="uk">
<head>
    <title>На головну | Moodle: НУЗП</title>
    <link rel="shortcut icon" href="https://moodle.zp.edu.ua/theme/image.php/classic/theme/1728165302/favicon" />
    <meta name="apple-itunes-app" content="app-id=633359593, app-argument=https://moodle.zp.edu.ua/"/><link rel="manifest"
href="https://moodle.zp.edu.ua/admin/tool/mobile/mobile.webmanifest.php" /><style>.tool_courserating-stars { color: #e59819; }
.tool_courserating-ratingcolor { color: #b4690e;}
.tool_courserating-norating .tool_courserating-stars { color: #a0a0a0;}
.tool_courserating-barcolor { background-color: #e59819;}
</style><meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta name="keywords" content="moodle, На головну | Moodle: НУЗП" />
<link rel="stylesheet" type="text/css" href="https://moodle.zp.edu.ua/theme/yui_combo.php?rollup/3.18.1/yui-moodlesimple-min.css" /><script id="firstthemesheet" type="text/css">/**
Required in order to fix style inclusion problems in IE with YUI **/</script><link rel="stylesheet" type="text/css"
href="https://moodle.zp.edu.ua/theme/styles.php/classic/1728165302_1/all" />
<link rel="stylesheet" type="text/css" href="https://moodle.zp.edu.ua/filter/syntaxhighlighter/styles/idea.min.css" />
<link rel="stylesheet" type="text/css" href="https://moodle.zp.edu.ua/course/format/onetopic/styles.php" />
<script>
```

Рисунок 1.10 - Приклад HTML документа

Примітивна HTML-сторінка зазвичай складається щонайменше з трьох основних тегів: `<html>`, `<head>`, та `<body>`. Тег `<head>` містить службову інформацію, зокрема заголовок сторінки, ключові слова та мета-дані. Тут також підключаються зовнішні ресурси, такі як стилі або JavaScript-скрипти. Вміст тегу `<head>` не відображається на сторінці після її завантаження у веб-браузері [8].

Тег `<body>` відповідає за вміст сторінки, який бачить користувач. Для підключення CSS стилів у HTML використовується тег `<link>`, де атрибут `href`

вказує на адресу CSS-файлу, а атрибут `rel` зі значенням `stylesheet` дає зрозуміти браузеру, що цей файл містить стилі.

У тілі сторінки, `<body>`, зазвичай використовуються такі теги, як: `<header>`, `<main>`, `<footer>`. Тег `<header>` містить вступну частину сторінки, яка зазвичай включає логотип та меню навігації. Тег `<main>` призначений для основного вмісту, що відповідає головній темі сторінки. А тег `<footer>` містить заключну частину, яка зазвичай включає інформацію про права власності, контактні дані та посилання на політики сайту.

DOM (Document Object Model) — це модель, відповідно до якої кожен HTML-тег вважається об'єктом. Вона має ієрархічну структуру, в якій кожен тег, що міститься в іншому, є "дитиною" батьківського елемента. Кожен з цих елементів також є об'єктом [9].

Коли веб-сайт відповідає на запит, він надсилає веб-браузеру HTML-код у вигляді байтів. Щоб відобразити сторінку, веб-браузер спочатку конвертує ці байти в текст. Потім цей текст перетворюється на вузли, які в свою чергу стають об'єктами. В результаті формуються DOM-дерево (див. рис. 1.11, 1.12).

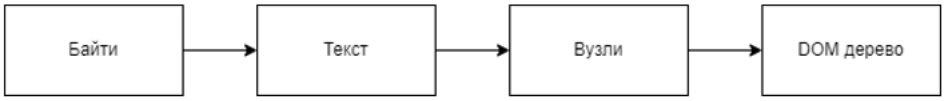


Рисунок 1.11 - Етапи перетворення відповіді у DOM

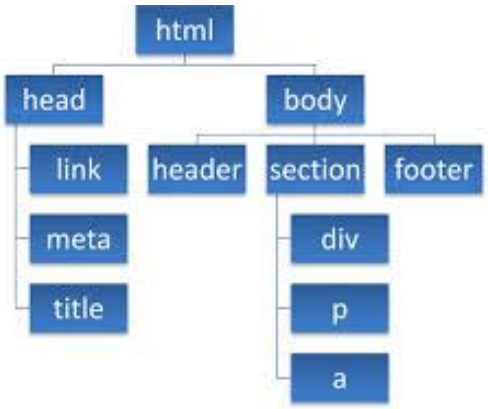


Рисунок 1.12 - Зображення структури DOM дерева

1.3 Аналіз типів веб-застосунків і механізмів пагінації веб-сторінок

Будь-який веб-застосунок в Інтернеті можна класифікувати на один із трьох типів: багато-сторінкові (MPA), односторінкові (SPA) та прогресивні (PWA) [10].

Односторінкові веб-застосунки, або SPA (Single Page Application), представляють собою веб-застосунок, який надає користувачу одну HTML-сторінку (рис. 1.13). Завдяки динамічному оновленню вмісту за допомогою JavaScript, зокрема AJAX, веб-браузеру не потрібно постійно перезавантажувати або завантажувати нові сторінки під час роботи з цим застосунком. Це означає, що коли користувач переглядає контент веб-сторінки або переходить до інших розділів, необхідні елементи завантажуються через AJAX і додаються до вже відображеної сторінки в браузері. Такий підхід створює більш комфортний досвід для користувача, адже взаємодія з односторінковим веб-застосунком надає враження роботи з десктопною програмою, оскільки застосунок миттєво реагує на дії користувача без затримок.

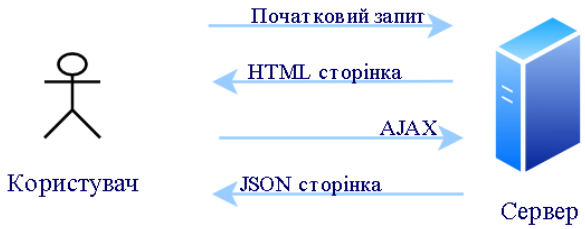


Рисунок 1.13 - Зображення процесу роботи SPA застосунку

Переваги односторінкових веб-застосунків включають:

- висока швидкість реагування - усі ресурси завантажуються за один раз, а під час взаємодії з сторінкою дані просто змінюються, що економить час користувача;

- гнучкість інтерфейсу - оскільки веб-сторінка є єдиною, розробникам легше налаштовувати та змінювати її вигляд і структуру, а також управляти даними, станами та анімаціями;

- кешування даних - веб-застосунок може зберігати дані після надання відповіді, що дозволяє йому працювати без підключення до Інтернету.

Недоліки цього типу веб-застосунків:

- навантаження на веб-браузер - через великий обсяг даних завантаження може займати значний час, іноді більше 500 мс, що негативно впливає на враження користувача;

- необхідність підтримки JavaScript - без JavaScript неможливо повноцінно використовувати функціонал веб-застосунку;

- витоки пам'яті в JavaScript - це може призводити до зниження продуктивності та стабільності веб-застосунків.

Багатосторінкові веб-застосунки (MPA, або Multi Page Application) — це тип веб-застосунків, який працює за традиційною схемою (рис. 1.14). У таких випадках, коли користувач або застосунок вносить зміни в дані, або під час завантаження нової сторінки, веб-браузер повністю перебудовує сторінку.

Переваги багатосторінкових веб-застосунків:

- звичний інтерфейс - користувачі звикли до класичної навігації та простоти використання, що робить такі застосунки зручними.

Недоліки:

- тісна інтеграція між бекендом і фронтендом - це може ускладнити розробку та підтримку коду;

- складність розробки - створення багатосторінкових застосунків може бути більш трудомістким через необхідність налаштування навігації між численними сторінками.

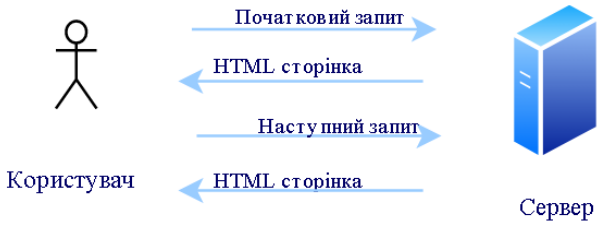


Рисунок 1.14 - Зображення процесу роботи MPA застосунку

Для ефективного надання користувачеві згрупованих даних в умовах обмежених часових та ресурсних витрат використовують два підходи: пагінацію та нескінченний скролінг. Кожен з цих методів має свої переваги та недоліки, що впливають на користувацький досвід.

Пагінація (англ. Pagination) — це система порядкової нумерації сторінок (рис. 1.15), яка зазвичай розміщується зверху або знизу, а іноді і з обох боків веб-сайту [11].

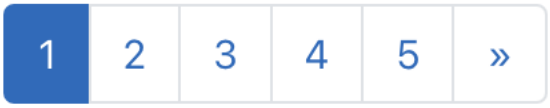


Рисунок 1.15 - Приклад пагінації

У нашому випадку пагінація полягає в структуризації даних, які діляться на певну кількість сторінок, наприклад, по 20 товарів на сторінку. Це робиться для зменшення обсягу інформації, яку потрібно обробляти веб-браузеру, а також для зручності користувача, щоб він міг швидко знайти потрібний товар, не повторюючи пошук серед усіх товарів.

Крім того, розподіл даних на сторінки сприяє прискоренню надання відповіді веб-сервером на запит користувача. Хоча HTML-сторінка може містити велику кількість інформації, наприклад, 1000 товарів одночасно, це значно вплине на час передачі даних.

Види пагінації в представленні користувачу:

- пряма порядкова нумерація: 1-2-3-4;

- сторінки, вказані в певному діапазоні: 1-10, 10-20, 20-30.

Види пагінації в реалізації:

- серверна пагінація - це коли веб-сервер, на якому розміщено дані, повертає лише підмножину, яка відповідає критеріям, наданим клієнтом. Разом із підмножиною сервер також повертає загальну кількість результатів, що відповідають критеріям запиту, і позицію клієнта в межах цього запиту. Тобто сервер надає дані, які запитував клієнт, скільки результатів було знайдено і де в списку знаходяться ці дані;

- клієнтська пагінація - це коли веб-сервер при виконанні запиту повертає клієнту всі дані одразу одним великим шматком. Нумерація сторінок на стороні клієнта дозволяє гортати дані практично миттєво, оскільки клієнту не потрібно звертатися до сервера для отримання нової підмножини.

Інструмент, який дозволяє користувачеві переглядати результати запиту, розміщені на веб-сторінці, у повному обсязі, забезпечує динамічне відображення контенту. Коли користувач прокручує сторінку до самого низу або натискає кнопку «Показати ще», нові дані завантажуються автоматично. Цей підхід до надання даних користувачу часто реалізується в односторінкових веб-застосунках.

1.4 Аналіз теоретичної комп'ютерної системи для автоматизованого збору та обробки інформації з веб-сайтів

Скрейпінг — це автоматизований процес збору структурованих даних (рис. 1.16), відомий також як інтелектуальний аналіз даних [12]. Зазвичай цей метод використовують індивіди та компанії, які прагнуть скористатися загальнодоступними веб-даними для ухвалення більш обґрунтованих рішень.

Коли ви копіюєте і вставляєте інформацію з веб-сайту, ви виконуєте аналогічну функцію, як і при скрейпінгу веб-сайтів, але в ручному режимі.

На відміну від трудомісткого процесу ручного вилучення даних, парсер веб-сайтів застосовує автоматизацію для збору сотень, мільйонів або навіть мільярдів одиниць структурованих даних з, здавалося б, безмежних просторів Інтернету.

Вилучення веб-даних має широкий спектр застосувань та форм реалізації.

Рисунок 1.16 - Робота скрейперу

Процес роботи програмного застосунку для збору та аналізу інформації з веб-сайтів можна умовно розділити на такі етапи:

- отримання переліку адрес для обробки - на цьому етапі програмний застосунок використовує попередньо підготовлений список URL-адрес, який може бути створений користувачем або розробником, а також додаткову інформацію про типи цих сторінок. Адресами можуть бути не тільки веб-сторінки, але й Ajax-оточення для взаємодії між веб-сервером та клієнтом;

- формування запитів до веб-сервера - на основі заздалегідь розроблених шаблонів запитів програмний застосунок використовує ці шаблони для створення запитів до зазначених адрес;

- отримання та зберігання відповідей - на цьому етапі застосунок надсилає запит до веб-сайту, перевіряє статус коди, і, якщо все в порядку, зберігає відповідь для подальшого використання;

- аналіз отриманих даних - програмний застосунок проводить обробку відповіді від веб-сайту, перетворюючи HTML-код на необхідні об'єкти, а також застосовуючи регулярні вирази, XPath тощо, для формування об'єктів результату обробки;

- зберігання даних - відповідно до заздалегідь розробленої логіки програмний застосунок зберігає відформатовані дані у файл або базу даних.

В залежності від типу веб-застосунку, принципи отримання даних системою збору можуть варіюватися. Наприклад, для отримання інформації з HTML-сторінки у веб-застосунках типу WPA зазвичай надсилається GET-запит (рис.1.17). Багатосторінкові веб-застосунки зазвичай надають повну сторінку з усією інформацією, однак в деяких випадках вони можуть використовувати AJAX. У таких ситуаціях розробнику програмного застосунку потрібно аналізувати весь трафік, що генерується веб-браузером і веб-сайтом під час взаємодії. Мета цього аналізу - виокремлення ключових етапів взаємодії для подальшої імітації дій користувача, включаючи порядок запитів та їх атрибути. Нажаль, багато веб-застосунків можуть заблокувати роботу програмного застосунку при виявленні підозрілої активності. Веб-застосунки типу SPA також використовують AJAX, тому етап аналізу трафіка та виокремлення важливих етапів є критично важливим.

Огляд процесу побудови системи аналізу інформації можна представити наступним чином:

- вибір веб-сайту - на першому етапі необхідно знайти ресурс, з якого ми будемо отримувати дані. Цей процес може бути поділений на кілька підетапів, таких як аналіз структури URL-адрес ресурсу та дослідження впливу фільтрів і їх значень на структуру запиту;

- аналіз вмісту сторінки - на цьому етапі слід проаналізувати вміст сторінки або сторінок, щоб виявити інформацію, яка нас цікавить. Зазвичай дані містяться в певних HTML-тегах, тому потрібно визначити, під якими тегами розташовані необхідні дані;

- вилучення даних з HTML- на цьому етапі необхідно знайти та витягнути дані з HTML-коду сторінки;

- визначення структури даних - після вилучення даних важливо визначити їхню структуру для подальшої обробки;

- вибір формату зберігання даних - останній етап полягає у виборі способу зберігання даних. Формат зберігання залежить від потреб користувача та специфіки застосування.

```
<body>

<div class="header">
  <h1 class="animated-heading">Мій веб-сайт</h1>
  <p class="animated-text">Розробляти сайт - це легко!</p>
</div>

<div class="topnav">
  <a href="#">Мотивація</a>
  <a href="#">Посібники</a>
  <a href="#">Допомога</a>
  <a href="#" style="float:right">Разом з HostPro</a>
</div>

<div class="row">

  <div class="leftcolumn">

    <div class="card">
      <h2>Перші кроки до створення сайту: почніть свій веб-шлях прямо зараз!</h2
          >
      <h5>Ви коли-небудь задумувалися про створення свого власного веб-сайту? Це
          може бути захоплююча подорож у світ веб-розробки, де лише ваша уява
          може стати межею.</h5>

      <div class="image-container">
        <img src="img/1.jpeg" alt="Приклад зображення">
        <img src="img/2.jpeg" alt="Приклад зображення">
      </div>

    </div>

    <div class="card">
      <h3>Підпишіться на мене</h3>
      <p>Отримуйте наші останні новини та оновлення:</p>
      <form action="/subscribe" method="post">
        <div class="input-group">
          <label for="email">Ваш Email:</label>
          <input type="email" id="email" name="email" required>
        </div>
```

Рисунок 1.17 - Поле даних, у структурі HTML

1.5 Огляд наявних рішень

1.5.1 ParseHub

ParseHub - це інструмент для веб-скрейпінгу, який дозволяє користувачам збирати дані з веб-сайтів без необхідності програмування [13]. Він надає зручний візуальний інтерфейс, що дозволяє легко налаштовувати процес збору даних.

Ось деякі ключові особливості та переваги ParseHub:

- візуальний інтерфейс - користувачі можуть просто вказувати елементи на веб-сторінці, які хочуть зібрати, за допомогою графічного інтерфейсу. Це спрощує процес, навіть для тих, хто не має досвіду програмування;

- підтримка JavaScript - ParseHub може обробляти веб-сторінки, які генеруються JavaScript, що робить його корисним для збору даних з динамічних вебсайтів;

- збір даних з кількох сторінок - інструмент дозволяє налаштувати парсинг з декількох сторінок, що корисно для збору даних з великих сайтів з пагінацією;

-експорт даних - зібрані дані можна експортувати в різні формати, такі як CSV, Excel або JSON, що полегшує подальшу обробку і аналіз;

- API - ParseHub надає API, що дозволяє автоматизувати процес збору даних та інтегрувати його з іншими додатками;

- крос-платформність - ParseHub доступний як веб-додаток, а також має версію для Windows та macOS, що робить його доступним на різних платформах.

Переваги використання ParseHub:

- простота використання - завдяки інтуїтивно зрозумілому інтерфейсу навіть новачки можуть швидко навчитися користуватися інструментом;

- гнучкість - користувачі можуть налаштувати збори даних для різних типів сайтів і структур;

- підтримка - ParseHub має активну спільноту та документацію, що допомагає користувачам у вирішенні проблем.

Недоліки:

- обмеження безкоштовної версії - версія має обмеження на кількість проектів і зібраних даних, що може бути проблемою для великих проектів;

- час обробки - для великих зборів даних може знадобитися значний час на обробку.

ParseHub є відмінним вибором для тих, хто потребує ефективного інструменту для збору даних з веб-сайтів без глибоких знань у програмуванні.

1.5.2 Puppeteer

Puppeteer - це бібліотека Node.js, розроблена Google, яка надає високорівневий API для роботи з браузером Google Chrome або Chromium через протокол DevTools [14]. Вона ідеально підходить для автоматизації веб-завдань, таких як тестування веб-додатків, веб-скрейпінг, генерація скріншотів і PDF-документів. Ось детальніший огляд основних можливостей і переваг Puppeteer:

Ключові можливості Puppeteer:

- автоматизація браузера - Puppeteer дозволяє запускати браузер у безголовому режимі (без графічного інтерфейсу) або в нормальному режимі, що робить його ідеальним для автоматизації завдань;

- веб-скрейпінг- бібліотека дозволяє легко збирати дані з веб-сторінок, включаючи динамічно завантажувані елементи, завдяки своїй здатності взаємодіяти з JavaScript;

- генерація скріншотів і PDF - Puppeteer може робити скріншоти веб-сторінок або генерувати PDF-документи з HTML-коду, що є корисним для документування чи звітності;

- тестування - Puppeteer може бути використаний для автоматизації тестування веб-додатків, дозволяючи тестувати функціональність і взаємодію користувачів з інтерфейсом;

- емуляція пристроїв - бібліотека підтримує емуляцію різних мобільних пристроїв, що дозволяє тестувати, як веб-додатки виглядають і працюють на різних екранах;

- складні сценарії - Puppeteer дозволяє взаємодіяти зі сторінками, заповнювати форми, натискати кнопки та виконувати інші дії, що дозволяє виконувати складні сценарії автоматизації.

Переваги використання Puppeteer:

- простота використання - має чистий і зрозумілий API, що спрощує процес написання сценаріїв;

- широкі можливості - завдяки потужним функціям, Puppeteer може використовуватися для різноманітних задач, від простого збору даних до комплексного тестування веб-додатків;

- швидкість - використання безголового режиму браузера дозволяє швидше виконувати завдання.

Недоліки Puppeteer:

- залежність від Node.js - Puppeteer працює лише в середовищі Node.js, що може бути обмеженням для деяких користувачів;

- споживання ресурсів - використання браузера може потребувати більше системних ресурсів у порівнянні з простими HTTP-запитами для збору даних;

- сумісність з іншими браузерами - Puppeteer в основному призначений для роботи з Chrome/Chromium, хоча є експериментальна підтримка інших браузерів.

Puppeteer - це потужний інструмент для автоматизації роботи з браузером, що підходить для різноманітних завдань, таких як веб-скрейпінг, тестування та генерація контенту. Завдяки своїй гнучкості і простоті використання, Puppeteer стає все більш популярним серед розробників і автоматизаторів.

1.5.3 Scrapy

Scrapy - це потужний фреймворк на Python для веб-скрейпінгу, веб-краулінгу та автоматизованої обробки даних з веб-сайтів [15]. Scrapy надає гнучкий інструментарій для збору, обробки та збереження інформації з веб-сторінок, дозволяючи користувачам з легкістю створювати скрейпери для різноманітних потреб.

Ключові можливості Scrapy:

- веб-краулінг - дозволяє створювати павуків (spiders), які автоматично подорожують вебсайтами та збирають інформацію з багатьох сторінок, слідуючи за посиланнями, це особливо корисно для побудови великих баз даних з різних веб-сайтів;

- асинхронна робота - Scrapy використовує асинхронний механізм, що дозволяє ефективно обробляти запити та економити час на завантаженні сторінок, що робить його швидким і продуктивним, особливо для великомасштабних проектів;

- гнучка обробка даних - Scrapy дозволяє витягувати дані в різних форматах, таких як CSV, JSON, XML, і зберігати їх у базах даних або файлах;

- вбудований механізм роботи з XPath і CSS селекторами - Scrapy надає зручні інструменти для доступу до елементів HTML-сторінок за допомогою XPath і CSS селекторів, що спрощує процес вилучення даних;

- підтримка середовища middleware - Scrapy дозволяє використовувати middleware для роботи з запитами та відповідями, що дозволяє налаштувати поведінку фреймворку під специфічні вимоги (наприклад, робота з проксі-серверами або ручне керування кукі);

- зручне налаштування політики обходу сайту - можна налаштувати різні параметри для обходу сайтів, включаючи обмеження швидкості запитів, чергу пріоритетів, кількість одночасних запитів тощо;

- підтримка розширень - Scrapy дозволяє додавати розширення для розширення функціоналу, наприклад, моніторинг продуктивності або підтримка API.

Переваги Scrapy:

- висока продуктивність - завдяки асинхронній архітектурі, Scrapy може одночасно обробляти велику кількість запитів, що дозволяє економити час на скрейпінг;

- широкі можливості обробки даних - можливість налаштувати різні схеми для вилучення та збереження даних, а також підтримка роботи з базами даних та файлами;

- гнучкість і масштабованість - легко налаштовується для різноманітних завдань, від простих до складних проектів з високими вимогами;

- активне співтовариство та підтримка - Scrapy має велику спільноту користувачів та розробників, що забезпечує наявність великої кількості документації та прикладів.

Недоліки Scrapy:

- крута крива навчання для початківців - новачки можуть зіткнутися зі складнощами під час освоєння фреймворку, особливо при налаштуванні складних павуків або middleware;

- обмежена підтримка динамічних сайтів - Scrapy більше орієнтований на скрейпінг статичних сторінок. Для роботи з динамічними сайтами (які використовують JavaScript) може знадобитися інтеграція з іншими інструментами, такими як Splash або Selenium;

- проблеми з обмеженням швидкості на стороні сайту - деякі сайти можуть блокувати ботів або обмежувати кількість запитів з однієї IP-адреси, що може вимагати додаткових налаштувань (проксі, затримки між запитами).

Scrapy — це відмінний вибір для розробників, яким потрібен потужний та гнучкий інструмент для веб-скрейпінгу та збору даних з інтернету. Завдяки асинхронній архітектурі, підтримці роботи з великим обсягом даних та широким можливостям кастомізації, Scrapy є чудовим інструментом для великих проектів, пов'язаних з аналізом веб-даних.

1.6 Висновки до розділу

У цьому розділі проведено аналіз проблеми створення комп'ютерної системи для автоматизованого збору та аналізу даних. Досліджено, як користувач взаємодіє з веб-браузером та веб-сайтом під час збору інформації. Основну увагу приділено ключовим етапам, поняттям і принципам, що використовуються для цієї взаємодії. Визначено важливі фактори, які впливають на функціонування програмного забезпечення, що забезпечує автоматизований збір і аналіз інформації з веб-сайтів. Також проведено аналіз існуючих рішень.

На основі цього були сформульовані наступні висновки:

- визначення ролі програмного забезпечення для збору та аналізу є критично важливим для його розробки, адже цей аспект впливає на успішність процесу створення програмного продукту;

- досліджено, як користувач взаємодіє з веб-сайтом, з детальним розглядом усіх аспектів, які будуть використовуватися при розробці програмного застосунку;

- проведено аналіз етапів взаємодії між веб-браузером і веб-сайтом;

- сформовано етапи роботи програмного застосунку, що здійснює автоматизований збір і аналіз даних, імітуючи взаємодію між веб-браузером і веб-сервером.

- розглянуто, яку інформацію веб-сайт надає у відповідь на запити, а також можливі варіанти відповідей і їх вміст.

- проаналізовано різні типи веб-сайтів, такі як динамічні і статичні, з урахуванням їхнього впливу на роботу програмного застосунку.

- оцінено готові рішення, що реалізують автоматизований збір та аналіз даних з веб-сайтів.

2 ДОСЛІДЖЕННЯ ІНСТРУМЕНТІВ ДЛЯ АВТОМАТИЗОВАНОГО ЗБОРУ ТА ОБРОБКИ ДАНИХ З ВЕБРЕСУРСІВ

2.1 Засоби для розробки програм автоматизованого збору та обробки даних

.NET C# надає розробникам низку інструментів і бібліотек, які можна використовувати для створення програмного забезпечення цього типу.

Для виконання HTTP-запитів доступні декілька варіантів рішень[17]:

- HttpClient — клас для надсилання HTTP-запитів та отримання відповідей від ресурсу за URL. Він підтримує асинхронні запити і може повертати відповіді у вигляді потоків або рядків. Рекомендується використовувати один екземпляр HttpClient для всього життєвого циклу програми, щоб уникнути вичерпання кількості сокетів, що може призвести до помилок. Недоліком є повільність завантаження сторінок;

- HttpWebRequest — клас, пов’язаний із реалізацією WebRequest для HTTP. Основні методи: GetResponse для синхронного запиту, який повертає HttpWebResponse з відповіддю. Файли cookie за замовчуванням вимкнені, але їх можна активувати за потреби;

- RestSharp — це NuGet пакет, призначений для виконання синхронних та асинхронних запитів до веб-сайтів через HTTP-протокол. Підходить для розробників, які працюють із REST API. Окрім цього, RestSharp може працювати з будь-якими API за допомогою HTTP, якщо є URL та параметри запиту відповідно до стандартів W3C HTTP.

Для аналізу HTML є наступні рішення:

- HtmlAgilityPack — одна з найпопулярніших бібліотек для аналізу HTML у .NET. Вона швидка і зручна для використання, особливо для простих запитів XPath;

- CsQuery — це порт JQuery для .NET 4, що підтримує всі селектори CSS2 та CSS3, а також маніпуляцію DOM, як у JQuery. Однак підтримку цього пакета припинено, замість нього рекомендується використовувати AngleSharp;

- AngleSharp — бібліотека .NET для аналізу HTML та SVG, з можливістю аналізу CSS. Вона побудована на основі офіційної специфікації W3C і є портативною, розширюваною та підтримує повнофункціональний DOM;

- Regex — один із найстаріших підходів до аналізу HTML, який дозволяє виконувати його швидше, ніж за допомогою бібліотек для побудови DOM.

Для імітації запитів користувача також використовується віртуальний браузер Headless Chrome. Це потужний інструмент для автоматизованого тестування і серверних середовищ, де немає потреби у видимій оболонці інтерфейсу користувача. Він дозволяє проводити тести на реальних веб-сторінках, генерувати PDF або перевіряти візуалізацію URL у браузері.

Python є чудовим вибором для розробки систем збору та аналізу даних з веб-сайтів, оскільки він пропонує ряд вбудованих бібліотек, які спеціально створені для таких завдань.

Ось кілька причин, чому Python ідеально підходить для розробки подібних програмних застосунків:

- простота у вивченні - синтаксис Python схожий на звичайну англійську мову, тому код на Python легко читається та розуміється, що робить його доступним для початківців;

- економія часу - Python дозволяє розробникам писати короткі та зрозумілі фрагменти коду для складних завдань, що значно зменшує час, витрачений на розробку;

- широка екосистема бібліотек - Python має багату екосистему бібліотек, які включають безліч інструментів для збору та аналізу даних з веб-сайтів;

- потужна підтримка спільноти - Python має одну з найбільших спільнот розробників, що гарантує швидке вирішення проблем і отримання допомоги під час розробки програм.

Ці особливості роблять Python чудовим вибором для реалізації програм автоматизованого збору та аналізу даних.

Існує кілька популярних бібліотек Python, які широко використовуються для ефективного збору та аналізу даних з веб-сайтів:

- Scrapy - потужна бібліотека для збору даних з веб-сайтів та їх вилучення у структурованому форматі. Scrapy може бути корисною для моніторингу та автоматизованого тестування, проте має обмеження у взаємодії з веб-застосунками, які базуються на JavaScript;

- Selenium - відкритий веб-драйвер, що дозволяє автоматизувати взаємодію з браузером. Основним недоліком є те, що системи відстеження трафіку легко можуть його розпізнати, коли використовуються для відкриття великої кількості сторінок, що робить його менш ефективним для збору великих обсягів даних;

- BeautifulSoup - одна з найпопулярніших бібліотек для аналізу HTML та XML документів. Вона дозволяє перетворити їх у деревоподібну структуру, полегшуючи ідентифікацію та вилучення необхідних даних.

2.2 Бібліотеки та інструменти, застосовані в системі автоматизованого збору й аналізу даних

Для реалізації комп’ютерних систем автоматизованого збору та аналізу інформації з веб-сайтів були використані такі інструменти, які сприяють спрощенню та покращенню процесу розробки програмного застосунку:

- HttpWebRequest - забезпечує можливість швидкого виконання запитів з параметрами до веб-сайтів;

- HtmlAgilityPack - пакет, що застосовується для створення DOM-об'єктів;

- WPF - платформа для розробки клієнтських додатків з візуально привабливими елементами взаємодії з користувачем.

- HttpWebRequest - це клас у .NET Framework, який використовується для здійснення HTTP-запитів до веб-сайтів. Цей клас дозволяє програмістам надсилати запити на веб-сервери і отримувати відповіді, які можуть бути у вигляді HTML, JSON, XML або інших форматів. Ось детальніший опис цього класу:

2.2.1 Основні характеристики HttpWebRequest

Синхронні та асинхронні запити. `HttpWebRequest` підтримує як синхронні, так і асинхронні запити. Синхронні запити блокують виконання програми до отримання відповіді, тоді як асинхронні запити дозволяють продовжувати виконання програми під час очікування відповіді [18]:

- налаштування параметрів запиту - клас надає можливість налаштування різних параметрів запиту, таких як HTTP-метод (GET, POST, PUT, DELETE), заголовки (headers), таймаути, параметри аутентифікації, кукі тощо. Це робить його гнучким і зручним для роботи з різними веб-сервісами;

- отримання відповіді - для отримання відповіді від сервера використовується метод `GetResponse()`, який повертає об'єкт типу `HttpWebResponse`і містить статус-код відповіді, заголовки і дані, отримані від сервера;

- обробка даних - дані, отримані від веб-сайту, можуть бути доступні у вигляді потоків (Stream) або як рядок (string), що дозволяє зручно обробляти та аналізувати дані;

- проблеми з безпекою - за замовчуванням `HttpWebRequest` не підтримує файли cookie, однак, якщо ваша програма потребує обробки сесій або збереження інформації про користувача, ви можете ввімкнути їх у налаштуваннях запиту;

- проблеми з продуктивністю - якщо ви створюєте новий екземпляр `HttpWebRequest` для кожного запиту, це може призвести до вичерпання сокетів, що ускладнює обслуговування запитів, також рекомендується використовувати один екземпляр для декількох запитів або використовувати `HttpClient` для більш ефективної роботи.

`HttpWebRequest` є потужним інструментом для роботи з HTTP-запитами в .NET Framework. Він забезпечує гнучкість і контроль над запитами, що робить його корисним для розробників, які працюють із веб-сервісами. Однак для нових проектів розгляньте використання `HttpClient`, який є більш сучасним і рекомендованим рішенням для роботи з HTTP-запитами.

2.2.2 Основні характеристики HtmlAgilityPack

HtmlAgilityPack — це популярна бібліотека для .NET, яка дозволяє розробникам ефективно аналізувати (парсити) HTML-документи і маніпулювати ними. Вона особливо корисна для розробки програм, що займаються збором даних з веб-сторінок, оскільки спрощує процес витягування інформації з HTML.

Основні характеристики HtmlAgilityPack [19]:

- парсинг HTML - HtmlAgilityPack здатна розбирати HTML-документи, перетворюючи їх у структуровану модель, що полегшує доступ до елементів документа. Це важливо для обробки некоректного або неповного HTML, оскільки бібліотека намагається виправити помилки;

- XPath та CSS селектори - бібліотека підтримує запити XPath, що дозволяє здійснювати точний доступ до елементів у документі. Також можливе використання CSS селекторів для пошуку елементів, що робить її зручною для тих, хто знайомий із CSS;

- маніпуляція документом - HtmlAgilityPack надає можливості для модифікації HTML-документів. Користувачі можуть додавати, видаляти або змінювати елементи, атрибути та текст у документі;

- збереження та вивід HTML - бібліотека дозволяє зберігати модифіковані HTML-документи у файл або отримувати їх у вигляді рядка. Це зручно для подальшого використання або аналізу;

- підтримка UTF-8 - HtmlAgilityPack підтримує різні кодування, включаючи UTF-8, що дозволяє працювати з документами на різних мовах без проблем з кодуванням.

Використання HtmlAgilityPack.

Для початку роботи з HtmlAgilityPack вам потрібно встановити бібліотеку через NuGet. Це можна зробити за допомогою команди в консолі Package Manager.

Переваги HtmlAgilityPack:

- гнучкість - HtmlAgilityPack підходить для роботи з різноманітними структурами HTML, навіть якщо вони містять помилки;

- простота використання - API бібліотеки є зрозумілим і зручним, що дозволяє швидко почати роботу без тривалого навчання;

- можливість маніпуляції даними - Ви можете не тільки парсити, але й редагувати HTML-документи.

Недоліки HtmlAgilityPack:

- продуктивність - у деяких випадках, особливо з великими документами, бібліотека може бути менш продуктивною порівняно з іншими, більш легкими рішеннями;

- обмежена підтримка динамічних веб-сторінок - HtmlAgilityPack не може обробляти JavaScript, тому для збору даних з динамічних сайтів може знадобитися комбінування з іншими інструментами, такими як Selenium.

HtmlAgilityPack є потужним інструментом для парсингу HTML у середовищі .NET. Його можливості роблять його ідеальним вибором для розробників, які потребують ефективного рішення для збору та обробки даних з веб-сайтів. Завдяки простоті використання та гнучкості, HtmlAgilityPack залишається однією з найбільш популярних бібліотек для роботи з HTML у .NET-середовищі.

2.2.3 Основні характеристики Windows Presentation Foundation

WPF (Windows Presentation Foundation) - це графічна підсистема для створення інтерфейсів користувача у додатках на платформі .NET [20]. Розроблена компанією Microsoft, WPF забезпечує сучасний і гнучкий підхід до розробки настільних додатків для Windows, дозволяючи програмістам створювати привабливі та функціональні користувацькі інтерфейси.

Основні характеристики WPF:

-XAML (eXtensible Application Markup Language) - WPF використовує XAML для опису елементів інтерфейсу користувача, що дозволяє відокремити логіку програми від її представлення. XAML є декларативною мовою розмітки, яка дозволяє створювати складні UI без написання великої кількості коду;

- дані та прив’язка - WPF підтримує потужну систему прив'язки даних, яка дозволяє автоматично оновлювати інтерфейс при зміні даних у моделі, що значно спрощує управління станом інтерфейсу і знижує кількість коду, необхідного для обробки змін;

- векторна графіка - WPF підтримує векторну графіку, що дозволяє створювати масштабовані елементи інтерфейсу без втрати якості, що забезпечує кращу продуктивність і вигляд на різних роздільних здатностях екранів;

- шаблони і стилі - WPF дозволяє визначати стилі і шаблони для елементів управління, що спрощує зміну вигляду елементів і створення єдиного стилю для всього додатку, щозабезпечує узгодженість у дизайні і знижує повторюваність коду;

- події та командна модель - WPF має розвинуту систему обробки подій, а також підтримує командну модель, що дозволяє розділити логіку обробки подій і бізнес-логіку, що полегшує управління діями користувача;

- сумісність з WinForms - Хоча WPF є самостійною технологією, вона може бути інтегрована з додатками, розробленими за допомогою Windows Forms, дозволяє поступово переходити на WPF у існуючих проектах;

- вбудована підтримка мультимедіа - WPF має вбудовані засоби для роботи з мультимедіа, що дозволяє легко вбудовувати відео, аудіо та анімації в додатки;

- динамічні та адаптивні інтерфейси - за допомогою механізмів адаптації та динамічного розміщення елементів, WPF дозволяє створювати інтерфейси, які автоматично підлаштовуються під різні розміри вікон і пристроїв.

Переваги WPF:

- сучасний інтерфейс - WPF дозволяє створювати привабливі, інтуїтивно зрозумілі інтерфейси з анімацією та векторною графікою;

- потужна архітектура - завдяки відокремленню даних, представлення і логіки, WPF сприяє створенню більш зрозумілого та підтримуваного коду;

- велика спільнота і підтримка - WPF має велику спільноту розробників, що забезпечує широкий спектр ресурсів, документації і прикладів.

Недоліки WPF:

- високі вимоги до системи - WPF-додатки можуть вимагати більше ресурсів, ніж традиційні Windows Forms-додатки, особливо при використанні графіки та анімацій;

- крива навчання - для новачків, які не знайомі з XAML і концепціями WPF, може знадобитися деякий час для освоєння технології.

WPF - це потужна технологія для розробки настільних додатків на платформі Windows, яка забезпечує можливості для створення красивих і функціональних інтерфейсів користувача. Завдяки своїй гнучкості, простоті використання XAML, потужній системі прив'язки даних та підтримці мультимедіа, WPF залишається популярним вибором серед розробників для створення сучасних Windows-додатків.

2.3 Обгрунтування для розробки

Було проведено аналіз інструментів, технологій і бібліотек, що використовуються для створення комп'ютерних систем автоматизованого збору та аналізу інформації з веб-сайтів. На основі проведеного аналізу можна зробити наступні висновки:

- Python є оптимальним вибором для швидкої та зручної розробки систем автоматизованого збору і аналізу, оскільки бібліотеки надають готові реалізації процесів, що спрощує створення застосунків;

- для розробки десктопних додатків, що здійснюють автоматизований збір та аналіз даних, доцільно використовувати мову C#, оскільки вона забезпечує

необхідні засоби для створення користувацького інтерфейсу та містить функціонал для базових процесів взаємодії з веб-сайтами;

- для полегшення розробки програмного забезпечення рекомендується використовувати готові бібліотеки та інструменти;

- ознайомлено і вивчено основні інструменти, які пропонує .NET, для створення базового функціоналу програмного застосунку збору та аналізу даних з веб-сайтів.

3 СТВОРЕННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ ДЛЯ АВТОМАТИЗОВАНОГО ЗБОРУ ТА АНАЛІЗУ ДАНИХ З ВЕБРЕСУРСІВ

3.1 Призначення системи, вибір сервісів для реалізації

Розроблені програмні застосунки для автоматизованого збору та аналізу даних були створені з метою детального вивчення процесу їх розробки, а також дослідження можливостей автоматизації ключових етапів у цьому процесі. Це дозволяє краще зрозуміти, як автоматизовані системи можуть виконувати завдання, пов'язані зі збором інформації з веб-сайтів, її подальшим аналізом та обробкою, що є важливим для підвищення ефективності таких систем.

Однією з основних цілей є визначення переваг та недоліків різних підходів до створення таких систем. Це передбачає глибокий аналіз двох варіантів програмних рішень: в одному може бути реалізована одна архітектура або набір інструментів, а в іншому — інший підхід. Таким чином, можна визначити, який варіант краще підходить для конкретних завдань. Це допоможе сформувати обґрунтоване уявлення про те, в яких випадках один варіант є більш вигідним та ефективним для виконання певних майбутніх завдань, пов'язаних зі збором та обробкою інформації.

Крім того, для кожної системи буде проведено тестування з метою оцінки її продуктивності, надійності та зручності використання. Результати тестування

дозволять порівняти програмні застосунки за такими показниками, як швидкість збору даних, ефективність аналізу, стабільність роботи та масштабованість. Це порівняння дозволить зробити висновки про доцільність використання тієї чи іншої системи залежно від умов та вимог конкретного проекту.

Таким чином, створення цих програмних застосунків дозволить не лише ознайомитися з процесом їх розробки, але й допоможе виявити оптимальні рішення для різних сценаріїв збору та аналізу даних, що важливо для майбутніх досліджень і впроваджень у цій сфері.

3.2 Вимоги до системи

Програмний застосунок, розроблений у рамках дослідження для автоматизованого аналізу та обробки інформації, має забезпечувати такі функціональні вимоги [21]:

- можливість запуску - користувач повинен мати змогу запустити програмний застосунок для початку роботи;

- можливість завершення - користувач повинен мати змогу закрити програмний застосунок за потреби;

- перегляд результатів - користувач має можливість переглядати результати виконання програми у вигляді таблиці з даними, які були оброблені;

- підтримка Get-запитів через HTTPS - програмний застосунок повинен мати здатність здійснювати Get-запити до веб-сайтів через захищений протокол HTTPS;

- аналіз даних - застосунок повинен аналізувати дані, отримані від веб-сайтів, і впорядковувати їх відповідно до визначених критеріїв.

Програмне забезпечення, розроблене в рамках даного дослідження, повинно відповідати наступним функціональним вимогам:

- вибір директорії для збереження - користувач повинен мати можливість вибирати папку для збереження інформації, яка генерується під час роботи програмного застосунку;

- створення нових запитів - користувач повинен мати змогу створювати нові запити для збору та аналізу даних;

- перегляд створених запитів - користувач повинен мати можливість переглядати список раніше створених ним запитів;

- створення нових полів для запиту - користувач повинен мати можливість додавати нові поля до запиту, які будуть використовуватися під час процесу аналізу даних;

- перегляд полів запитів - користувач повинен мати змогу переглядати створені ним поля для запитів;

- редагування полів та оновлення Xpath - користувач повинен мати можливість редагувати існуючі поля запитів, включаючи оновлення значень Xpath для кожного з них;

- запуск процесу збору і аналізу - користувач повинен мати можливість запускати процес збору та аналізу даних для окремого запиту;

- отримання результатів аналізу - користувач повинен мати можливість отримувати результати аналізу у вигляді файлу, який містить структуровані дані.

Нефункціональні вимоги:

- зручний і зрозумілий інтерфейс - інтерфейс програмного застосунку повинен бути інтуїтивно зрозумілим та легким у використанні для забезпечення комфортної роботи користувачів;

- взаємодія через графічний інтерфейс - вся робота з програмним застосунком повинна здійснюватися через користувацький інтерфейс, без потреби у використанні командного рядка чи інших методів взаємодії;

- простота використання - програмний застосунок повинен бути простим у користуванні, щоб навіть непідготовлені користувачі могли легко зрозуміти, як з ним працювати;

- швидкість виконання - система повинна швидко виконувати процес збору та аналізу даних, оперативно надаючи результати користувачам;

- можливість відновлення роботи після збоїв - програмний застосунок повинен мати механізм відновлення роботи після виникнення збоїв, щоб уникнути втрати даних і забезпечити стабільну роботу;

- зручний вибір директорії для збереження даних - користувач повинен мати зручний та інтуїтивно зрозумілий інтерфейс для вибору директорії, де буде зберігатися інформація, необхідна для роботи програмного застосунку.

3.3 Діаграми використання у системі

3.3.1 Діаграми використання в статичній автоматизованій системі збору та аналізу даних з веб-сайтів

Відповідно до специфікації UML [22], діаграма випадків використання (Use Case Diagram) представляє взаємодію між дійовими особами (акторами) та варіантами використання (use cases) в системі (рис.3.1).

Користувач - особа, якій необхідно отримати проаналізовані дані з конкретного веб-сайту.

На основі потреб користувачів, можна виділити такі основні варіанти використання програмного застосунку:

- відкрити програмний застосунок: Користувач повинен мати змогу запустити програму для подальшого використання;

- закрити програмний застосунок: Користувач повинен мати можливість закрити програму після завершення роботи;

- переглянути результат обробки інформації: Користувач може переглядати результати збору та аналізу даних, які надані після обробки веб-сайту;

- почати процес автоматизованого збору та аналізу даних: Користувач має можливість ініціювати процес збору та аналізу даних з веб-сайту автоматично.

Ці випадки використання є базовими операціями, що дозволяють користувачу взаємодіяти із системою автоматизованого збору та аналізу даних з веб-сайтів. UML-діаграма допомагає візуалізувати та структурувати функціональні можливості системи на етапі проектування.

Рисунок 3.1 – Діаграма функціональних можливостей

3.3.2 Взаємодія користувача з запитами

Для взаємодії користувача із запитами в системі автоматизованого збору та аналізу даних виділяються наступні варіанти використання:

- переглянути існуючі запити - користувач може отримати доступ до списку всіх раніше створених запитів для подальшого перегляду або обробки;

- створити новий запит - користувач має можливість додати новий запит, задавши необхідні параметри для збору та аналізу даних;

- редагувати ім’я запиту - користувач може змінити назву існуючого запиту для зручності ідентифікації;

- редагувати текст посилання на елемент веб-сайту: Користувач може коригувати посилання на конкретний елемент на веб-сайті, який є цільовим для збору інформації;

- згенерувати новий Xpath для елемента - користувач може створити або відредагувати Xpath, який вказує на елемент веб-сайту, з якого необхідно отримати дані;

- запустити процес збору та аналізу запиту - користувач може ініціювати процес збору і аналізу даних на основі конкретного запиту;

- робота з полями запиту - користувач може додавати, редагувати або видаляти поля запиту для налаштування параметрів збору даних;

- видалити запит - користувач може видалити запит, якщо він більше не потрібен або є зайвим.

Ці варіанти використання забезпечують повний цикл роботи з запитами, включаючи їх створення, редагування, виконання та видалення, що дозволяє користувачу максимально ефективно працювати з системою.

Рисунок 3.2 - Діаграма взаємодії користувача із запитами

Для ефективної взаємодії користувача з полями запиту виділяються наступні варіанти використання (рис. 3.3):

- переглянути існуючі поля запиту - користувач може переглядати всі раніше створені поля, які використовуються для налаштування запиту;

- створити нове поле запиту - користувач може додати нове поле для запиту, яке буде використовуватися під час збору та аналізу даних;

- редагувати ім’я поля запиту - користувач може змінити назву поля, яке буде використовуватися під час форматування отриманих даних;

- редагувати текстове поле для пошуку - користувач може коригувати текстове поле, яке визначає, як відбуватиметься пошук відповідного елемента на HTML-сторінці;

- оновити Xpath поля - користувач має можливість оновити або змінити Xpath для конкретного поля, що використовується для точного визначення елементу в структурі HTML;

- видалити поле запиту - користувач може видалити поле запиту, якщо воно більше не потрібне для процесу збору та аналізу даних.

Ці варіанти використання дозволяють користувачу гнучко керувати полями запиту, що необхідно для налаштування і точного збору інформації з веб-сайтів.

Рисунок 3.3 - Діаграма створення та редагування полів запиту

3.4 Вимоги до користувацього інтерфуйсу

На основі аналізу функціоналу, який має забезпечувати програмний застосунок, а також вимог до проектування інтерфейсу, було сформульовано низку рекомендацій для створення зрозумілого і зручного для користувача інтерфейсу. Основні рекомендації включають наступне:

- головне вікно програмного застосунку - це вікно повинно відображати список створених користувачем запитів. У ньому також мають бути присутні елементи для вибору директорії, в якій зберігаються запити, кнопки для ініціації процесу створення нового запиту і збереження поточного переліку запитів. Важливо також передбачити функцію перегляду полів запиту, яка дозволить користувачу подвійним натисканням на запит відкривати його деталі.

- вікно створення запиту - це вікно повинно містити необхідні поля, які користувач має заповнити для створення нового запиту, а також елемент для підтвердження його створення;

- вікно перегляду полів запиту - це вікно повинно відображати список існуючих полів запиту на даний момент. Користувач має мати змогу створювати нові поля для запиту, а також починати процес обробки і аналізу даних для обраного запиту;

- вікно створення або редагування поля запиту - це вікно повинно містити поля для введення даних та зрозумілі користувачеві підписи до них, у ньому користувач може не тільки створювати нові поля, але й редагувати вже існуючі, а також зберігати внесені зміни.

Ці рекомендації спрямовані на те, щоб зробити інтерфейс програмного застосунку інтуїтивно зрозумілим, ефективним і простим у використанні для користувача, забезпечуючи при цьому всі необхідні функції для взаємодії із запитами та даними.

3.5 Діаграми виконання послідовностей

Користувач запускає програмний застосунок. Застосунок створює екземпляр класу HttpClient і використовує його для відправлення GET-запиту до вказаного ресурсу [23]. Після отримання відповіді від сервера і перевірки статус-коду, контент відповіді передається в Html Agility Pack, де рядок конвертується у DOM-документ. Далі цей документ обробляється за допомогою XPATH для вилучення необхідних даних. Після отримання бажаних результатів за допомогою XPATH, дані зберігаються та записуються у файл. Після завершення всіх процесів викликається користувацький інтерфейс, в якому відображаються результати парсингу (рис.3.4).

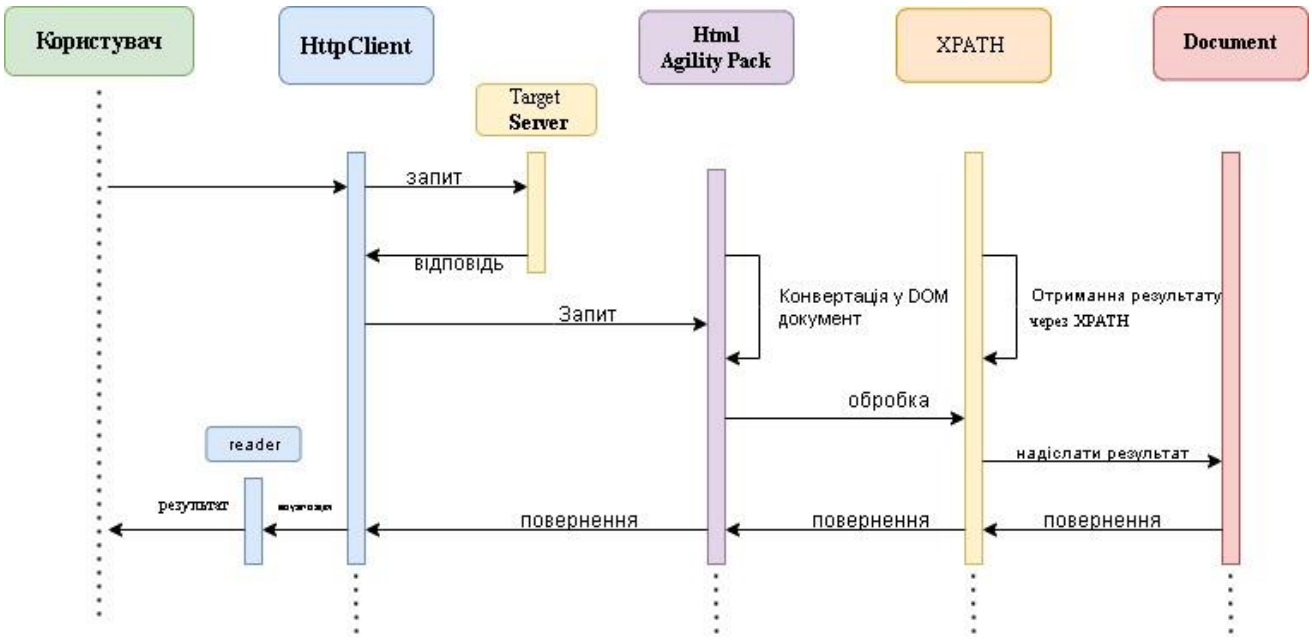


Рисунок 3.4 –Діаграма послідовностей системи автоматизованого збору та аналізу даних

Наступна ілюстрація (рис.3.5) демонструє цілеспрямований запит у двох варіаціях:

- перехід від розділу до розділу за допомогою текстового (голосового) введення настпних підрозділів та кнопок;

- миттєве отримання відповіді за умови, що початковий запит містив ключове слово кінцевої адресації.

Користувач натискає кнопку «Open folder», після чого перед ним з’являється діалогове вікно системного провідника. Він вибирає місце для зберігання даних і натискає кнопку «Обрати». Програмний застосунок отримує шлях до обраної користувачем директорії та перевіряє наявність необхідних папок і файлів для коректної роботи. Якщо потрібні файли відсутні, застосунок автоматично створює їх. Після цього на екрані відображається основна інформація про існуючі запити.

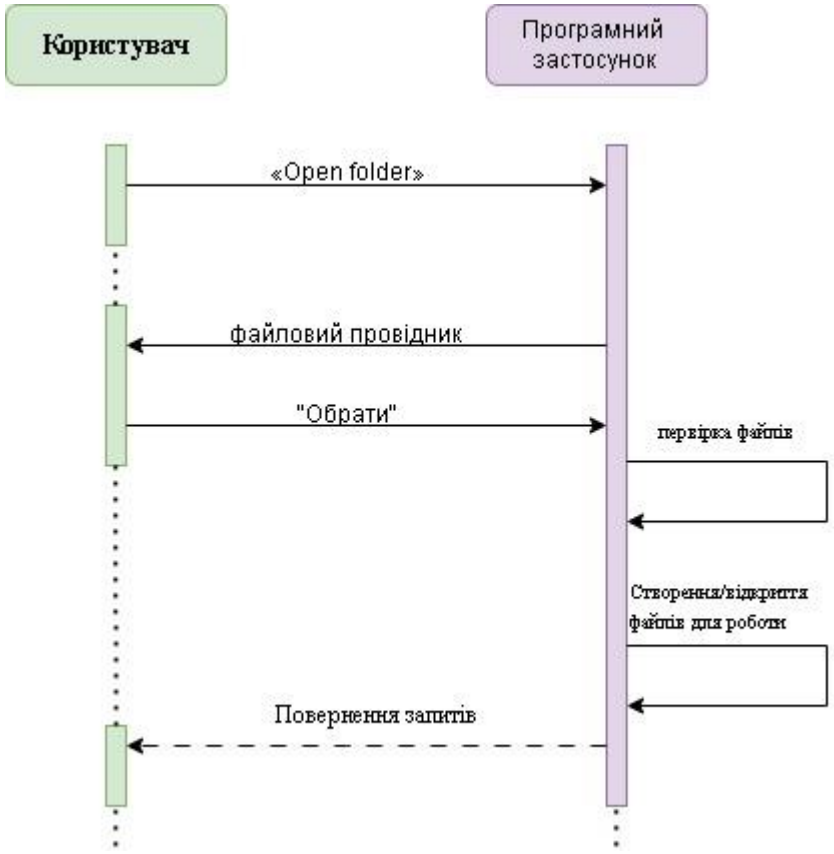


Рисунок 3.5 - Діаграма послідовності про існуючі запити

У діаграмі послідовності для сценарію «Користувач створює новий запит» відображається взаємодія між користувачем та програмним застосунком.

Ось опис процесу:

а) користувач натискає кнопку «Create new request»:

1) запускається процес створення нового запиту;

2) з'являється нове вікно.

б) у вікні користувачу пропонується вказати необхідні дані:

1) назва запиту - користувач вводить назву запиту;

2) головна URL адреса веб-сайту - користувач вводить основну URL адресу;

3) URL адреса зі списком інформації - користувач вводить URL, що містить посилання на інформаційні сторінки;

4) URL посилання на сторінку з інформацією - користувач надає URL для конкретної сторінки;

5) текстовий допис - користувач вводить текст, що супроводжує посилання на сторінку з інформацією;

в) користувач натискає кнопку «Save request»:

1) після введення всіх даних, користувач підтверджує збереження запиту;

г) програмний застосунок обробляє введені дані:

1) застосунок зберігає введену інформацію у файл запитів;

2) генерує Xpath на основі текстового допису, що вводився користувачем;

д) створення файлу для полів запиту:

1) застосунок створює файл, що міститиме поля запиту з назвою, вказаною користувачем;

ж) Підтвердження про успішне збереження:

1) застосунок може відобразити повідомлення про успішне збереження запиту або про створення файлу.

Графічне представлення. В діаграмі послідовності це може бути представлено наступним чином (рис. 3.6):

```
Користувач --> Програмний застосунок: Натискає «Create new request»
Програмний застосунок --> Користувач: Відкриває вікно для введення даних
Користувач --> Програмний застосунок: Вводить назву запиту
Користувач --> Програмний застосунок: Вводить головну URL адресу
Користувач --> Програмний застосунок: Вводить URL адреса зі списком
Користувач --> Програмний застосунок: Вводить URL посилання на сторінку
Користувач --> Програмний застосунок: Вводить текстовий допис
Користувач --> Програмний застосунок: Натискає «Save request»
Програмний застосунок --> Файл запитів: Зберігає інформацію
Програмний застосунок --> Генератор Xpath: Створює Xpath
Програмний застосунок --> Файл полів запиту: Створює файл з назвою запиту
Програмний застосунок --> Користувач: Підтверджує успішне збереження
```

Рисунок 3.6 – Діаграма послідовностей

Ця діаграма послідовності (рис. 3.7) наочно ілюструє кроки, які виконує користувач, і реакції програмного застосунку на ці дії.

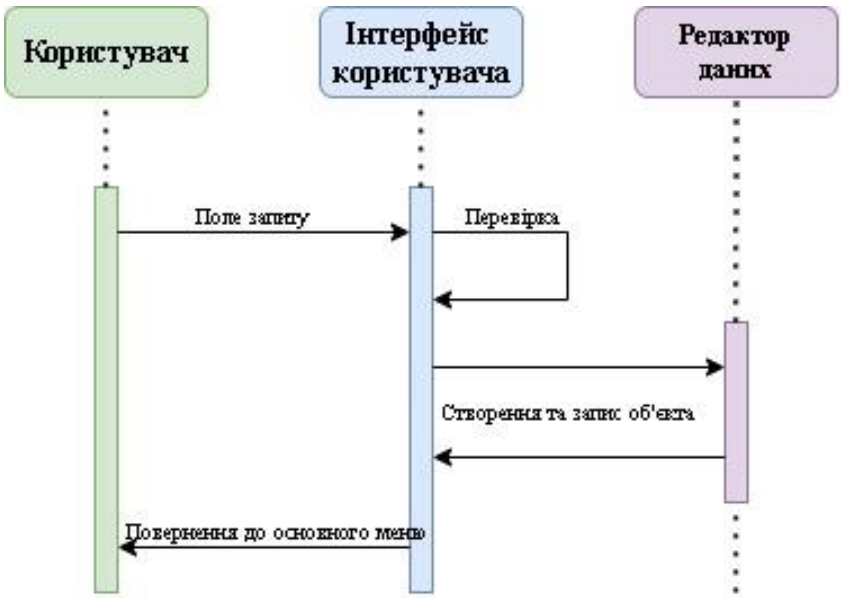


Рисунок 3.7 – Діаграма послідовності створення запиту коричтувачем

У діаграмі послідовності для сценарію «Користувач відкриває інформацію про запит» відображається взаємодія між користувачем і програмним застосунком, коли користувач відкриває деталі запиту.

Ось опис процесу:

- користувач натискає два рази на рядок з запитом - це ініціює відкриття інформації про конкретний запит;

- програмний застосунок відкриває нове вікно - де користувач може переглядати деталі запиту;

- доступ до функцій у новому вікні.

У новому вікні користувач може виконати наступні дії:

- користувач може переглядати всі поля, пов'язані з даним запитом;

- користувач має можливість додати нове поле, яке буде використовуватися у запиті;

- користувач може зберегти будь-які зміни, внесені до полів запиту;

- користувач може ініціювати процес збору і аналізу даних для конкретного запиту.

Графічне представлення.

У діаграмі послідовності може бути представлено наступним чином (рис.3.8):

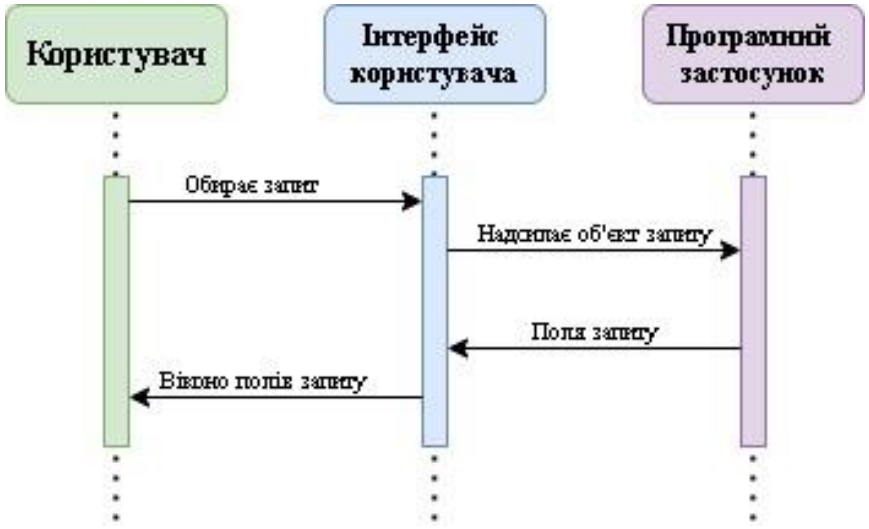


Рисунок 3.8 – Діаграма взаємодії між учасниками процесу про запит

Ця діаграма послідовності наочно ілюструє кроки, які виконує користувач, і реакції програмного застосунку на ці дії, що дозволяє зрозуміти

взаємодію між учасниками в процесі відкриття і роботи з інформацією про запит.

3.6 Діаграма класів

3.6.1 Діаграма класів у статистичній автоматизованій системі збору та аналізу даних

Діаграма класів є важливим елементом моделювання програмного забезпечення, який показує структуру системи шляхом відображення класів, їх атрибутів, методів та відносин між ними. У випадку статичної автоматизованої системи збору та аналізу даних з веб-сайтів, діаграма класів ілюструє основні компоненти системи та їх взаємодії.

Основні елементи діаграми класів (рис.3.9):

а) класи:

1) запит (Request) - включає атрибути, такі як назва запиту, головна URL-адреса, і методи для збереження та редагування запиту;

2) поле Запиту (RequestField) - містить атрибути для зберігання інформації про конкретне поле запиту, наприклад, Xpath, тип даних та методи для обробки;

3) HTTPClient – відповідає за виконання HTTP-запитів та отримання відповідей від веб-сайтів;

4) аналізатор (Analyzer) - включає методи для обробки даних, отриманих з веб-сайтів, і генерації звітів;

5) файл (File) - представляє інформацію про файли, в які зберігаються запити або результати аналізу, з методами для запису та читання даних.

б) відносини:

1) асоціація - клас `Request` може містити список `RequestField`, що показує, що запит може мати кілька полів;

2) залежність - клас `Analyzer` може залежати від класу `HTTPClient`, оскільки для аналізу даних необхідно спочатку отримати їх через HTTP-запити;

3) агрегація - клас `Request` може агрегувати об'єкти класу `File`, що свідчить про те, що запит може бути збережений у файлі.

Діаграма класів є важливим інструментом для проектування системи, що допомагає візуалізувати структуру і взаємозв’язки між різними компонентами. Вона дозволяє розробникам зрозуміти, як класи взаємодіють один з одним, і полегшує процес реалізації функціоналу автоматизованого збору та аналізу даних з веб-сайтів.

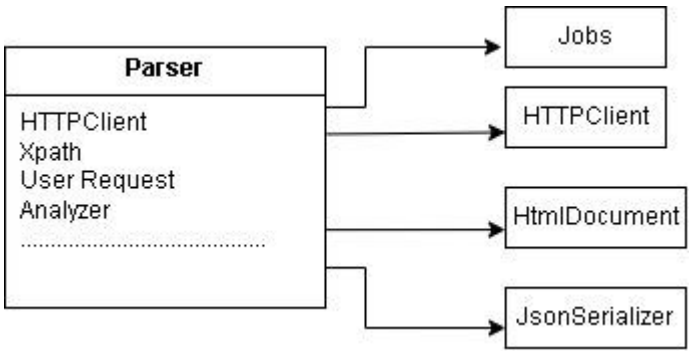


Рисунок 3.9 - Діаграма класів

3.6.2 Діаграми класів у динамічній автоматизованій системі збору та аналізу даних

Діаграма класів для програмного застосунку динамічного автоматизованого збору та аналізу інформації ілюструє структуру класів та їх взаємозв’язки. Основні класи, які використовуються в системі, включають:

а) клас `DataEditor`:

1) опис - цей клас відповідає за операції з файлами типу JSON;

2) методи:

- `writeToFile(data: JsonData) - void` записує дані у файл формату JSON;

- `readFromFile(filePath: string) – JsonData читає дані з файлу формату JSON.

б) клас `DomAnalyzer`:

1) опис - клас взаємодіє з бібліотекою HtmlAgilityPack для побудови DOM-моделі документа та проведення аналізу;

2) методи:

-`buildDomModel(htmlString: string) - HtmlDocument` створює DOM-модель з HTML-рядка;

- `analyzeDom(document: HtmlDocument) - AnalysisResult` виконує аналіз DOM-моделі.

в) клас `WebClient`:

1) опис: Клас відповідає за виконання HTTP-запитів та отримання відповідей від веб-сервера;

2) методи:

- `sendGetRequest(url: string) - string` виконує GET-запит та повертає відповідь у вигляді HTML-рядка - string` виконує POST-запит та повертає відповідь.

г) клас `UserRequest`:

1) опис - цей клас містить основну інформацію, необхідну для формування запиту користувача.

2) атрибути:

- `requestName: string` — назва запиту;

- `url: string` — URL для збору даних;

3) метод `validateRequest(): bool` - перевіряє коректність запиту.

д) клас `UserRequestField`:

1) опис - клас зберігає інформацію про поля запиту, які користувач може налаштувати;

2) атрибути - `fieldName: string` - назва поля;

3) методи - `updateField(newFieldName: string) - void` оновлює назву поля;

е) клас `Parser`:

1) опис - основний клас, відповідальний за управління процесом запитів до сервера, аналізу отриманої інформації та запису результатів у файли;

2) методи:

- `executeRequest(userRequest: UserRequest) - void` виконує запит, аналізує дані і зберігає результати;

- `saveResultsToFile(data: AnalysisResult) - void зберігає результати аналізу у файл.

Взаємозв’язки між класами:

- клас `Parser` використовує `WebClient` для виконання запитів та отримання HTML-коду;

- клас `DomAnalyzer` відповідає за аналіз отриманих HTML-даних та взаємодіє з `HtmlAgilityPack`;

- клас `UserRequest` та `UserRequestField` можуть передаватися в `Parser`, щоб формувати запити на основі налаштувань користувача;

- клас `DataEditor` використовується для запису результатів аналізу у файли JSON.

Діаграма класів для динамічної автоматизованої системи збору та аналізу даних наочно ілюструє структуру класів, їх атрибути, методи та взаємозв’язки (рис. 3.10). Це допомагає розробникам зрозуміти архітектуру системи, спростити процес розробки та забезпечити гнучкість в її подальшій еволюції.

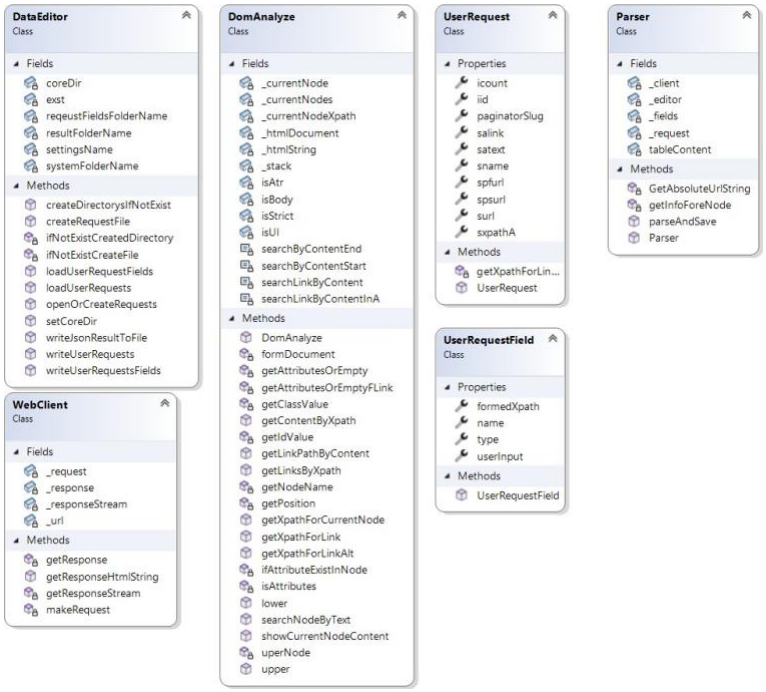


Рисунок 3.10 - Діаграма класів у динамічній системі

3.7 Засоби реалізації системи

Розроблені програмні застосунки були реалізовані в середовищі Visual Studio 2022 Community Edition з використанням об'єктно-орієнтованої мови C#. Для створення користувацького інтерфейсу застосовувалася технологія WPF (Windows Presentation Foundation). Для виконання запитів до веб-сервера був використаний стандартний клас .NET WebClient. Для побудови DOM-об'єкту та взаємодії з ним застосовувалася бібліотека HtmlAgilityPack.

У лістингу 3.1 представлені константи, що використовуються в парсері для XPATH. Введення цих констант дозволяє уникнути помилок розробника в майбутньому при змінах у логіці програми. Кожна константа відповідає конкретному елементу, який нас цікавить, що дає змогу швидко змінювати їх у разі модифікацій на вебсайті.

Змінна Client, яку ми ініціалізуємо на початку, надає можливість виконувати запити до сервера та переглядати їх результати. BaseUrl представляє собою основну частину URL-адреси сайту, а змінна userRequest є рядковою змінною, що містить дані, введені користувачем. Це забезпечує гнучкість та зручність при роботі з даними, що вводяться, і дозволяє ефективно керувати запитами.

Листинг 3.1 – Поля класу workUaParser

```
    private const string COUNT_XPATH = "//div[con-
tains(@class,'card')]/div[contains(@class,'add-
top')]";
            private const string JOBS_XPATH =
«//div[contains(@class,'job-link')]»;
            private const string JOB_LINK_XPATH =
«.//h2/a»;
            private const string JOB_COMPANY_XPATH =
«.//div/span/b»;
            private const string JOB_CITY_XPATH =
«.//div/span[3]»;
            private const string JOB_DESCRIPION_XPATH =
«.//p[contains(@class,'cut-bottom')]»;
            private const string JOB_PAYMENT_XPATH =
«.//div/b»;
            private const int PER_PAGE = 14;
            private const string PAGE_URL = «?page=»;
            private HttpClient client = new HttpCli-
ent();
            private string baseUrl;
            private string userRequest;
            private int jobsCount = 0;
            private int totalPagesCount = 0;
            private int currentPage = 1;
            private List<Job> resultJobs = new
List<Job>();
```

Лістинг 3.2 демонструє метод Parse, який викликається ззовні для ініціації процесу обробки та аналізу даних. Цей метод відповідає за виконання ключових дій, які дозволяють здійснити збір інформації з веб-сайту, її подальшу обробку та аналіз.

Листінг 3.2 – Метод Parse

```
Public async Task Parse(string userRequest)
        {
            this.userRequest = userRequest;
            string      response      =      await
GetResponseString(userRequest);
        this.Analyze(ConvertToHtmlDocument(re-
sponse));
        }
```

Листінг 3.3 демонструє метод Analyze, який виконує перевірку кількості наявних елементів, зокрема, у нашому випадку — вакансій. Метод підраховує, скільки сторінок додатково потрібно запросити у сервера для отримання всіх доступних вакансій.

Листінг 3.3 – Метод Analyze

```
        private   async   void   Analyze(HtmlAgili-
tyPack.HtmlDocument document)
        {
            this.jobsCount      =      this.GetCur-
rentCount(document);
            this.totalPagesCount = this.jobsCount /
PER_PAGE + (this.jobsCount % PER_PAGE);
            LoadJobs(document);
            do
            {
                this.currentPage++;
                string nextUrl =  this.userRequest
+ PAGE_URL + currentPage.ToString();

                Debug.WriteLine(nextUrl);
            }
            while  (this.currentPage  <=  this.to-
talPagesCount);
            WriteData();
        }
```

Наступний лістинг демонструє метод LoadJobs, який приймає DOM-документ і виконує аналіз даних, використовуючи регулярні вирази. Цей метод є важливим етапом у процесі обробки даних, оскільки він відповідає за вилучення інформації про вакансії з отриманого HTML.

Лістинг 3.4 - метод LoadJobs

```
            private void LoadJobs(HtmlAgili-
tyPack.HtmlDocument document)
            {
                HtmlNodeCollection jobs = document.Doc-
umentNode.SelectNodes(JOBS_XPATH);
                foreach (HtmlNode job in jobs)
                {
                    var titleAndLink = job.Se-
lectSingleNode(JOB_LINK_XPATH);
                    string url = this.baseUrl + ti-
tleAndLink.Attributes["href"].Value;
                    string title = titleAndLink.Attrib-
utes["title"].Value;
                    string company = job.SelectSingle-
Node(JOB_COMPANY_XPATH).InnerText;
                    string city = job.SelectSingle-
Node(JOB_CITY_XPATH).InnerText;
                    string description = job.Se-
lectSingleNode(JOB_DESCRIPION_XPATH).InnerText;
                    var prom = job.SelectSingle-
Node(JOB_PAYMENT_XPATH);
                    string payment ="";
                    if(prom != null)
                    {
                        payment = prom.InnerText;
                    }
                    resultJobs.Add(new Job()
                    {
                        url = url,
                        title = title,
                        company = company,
                        city = city,
                        description = description,
                        payment = payment
                    });
                }
                return;
            }
```

3.7.1 Модулі і алгоритми у динамічній автоматизованій системі збору та аналізу даних з веб-сайтів

Лістинг 3.5 демонструє методи класу DataEditor, які відповідають за отримання та запис інформації про запити користувача у файл формату JSON. Цей клас відіграє важливу роль у системі, оскільки забезпечує збереження і

доступ до даних, що вводяться користувачем, що, в свою чергу, сприяє зручному управлінню запитами.

Лістинг 3.5 - Методи класу DataEditor для запису та отримання інформації

```
       public void writeJsonResultToFile(string name, string
json)
            {
                string file = Path.Combine(coreDir,
resultFolderName, name + exst);
                using (StreamWriter outputFile = new
StreamWriter(file))
                {
                    outputFile.WriteLine(json);
                }
            }

            public BindingList<UserRequest>
openOrCreateRequests()
            {
                BindingList<UserRequest> requests = new
BindingList<UserRequest>();
                string file = Path.Combine(coreDir,
systemFolderName, settingsName + exst);

                ifNotExistCreateFile(file);
                using (StreamReader r = new
StreamReader(file))
                {
                    string json = r.ReadToEnd();
                    if (!string.IsNullOrEmpty(json))
                        requests =
JsonSerializer.Deserialize<BindingList<UserRequest>>(json);
                }
                return requests;
            }
```

Лістинг 3.6 демонструє метод parseAndSave класу Parse, який відповідає за процес надсилання запитів до веб-сервера та подальше виокремлення інформації відповідно до сформованих полів запиту. Цей метод є ключовим у системі, оскільки він інтегрує функціонал збору даних з веб-сайтів і обробку отриманої інформації.

Лістинг 3.6 - Метод parseAndSave

```
      string currentLink = _request.spfurl;
                    string htmlResponse =
_client.getResponseHtmlString(currentLink);
                    DomAnalyze dom = new
DomAnalyze(htmlResponse);
                    var response =
dom.getLinksByXpath(_request.sxpathA);
                    foreach (HtmlNode node in response)
                    {
                        string link =
GetAbsoluteUrlString(_request.surl,
node.Attributes["href"].Value);
                        if(node.Attributes["href"].Value !=
"#")
                        {
                            getInfoForeNode(link);
                        }
                    }
                    JsonSerializerOptions options = new
JsonSerializerOptions
                    {
                        Encoder =
JavaScriptEncoder.Create(UnicodeRanges.All)
                    };
                    string json =
JsonSerializer.Serialize(tableContent,options);

_editor.writeJsonResultToFile(_request.sname, json);
```

Лістинг 3.7 демонструє метод getAttributesOrEmptyFLink класу DomAnalyze, який відповідає за формування Xpath відповідно до наявних атрибутів у елементі. Цей метод є важливим етапом у процесі аналізу DOM-документів, оскільки він дозволяє точно вказати на потрібні елементи, використовуючи їх атрибути для побудови Xpath.

Лістинг 3.7 - Метод getAttributesOrEmptyFLink

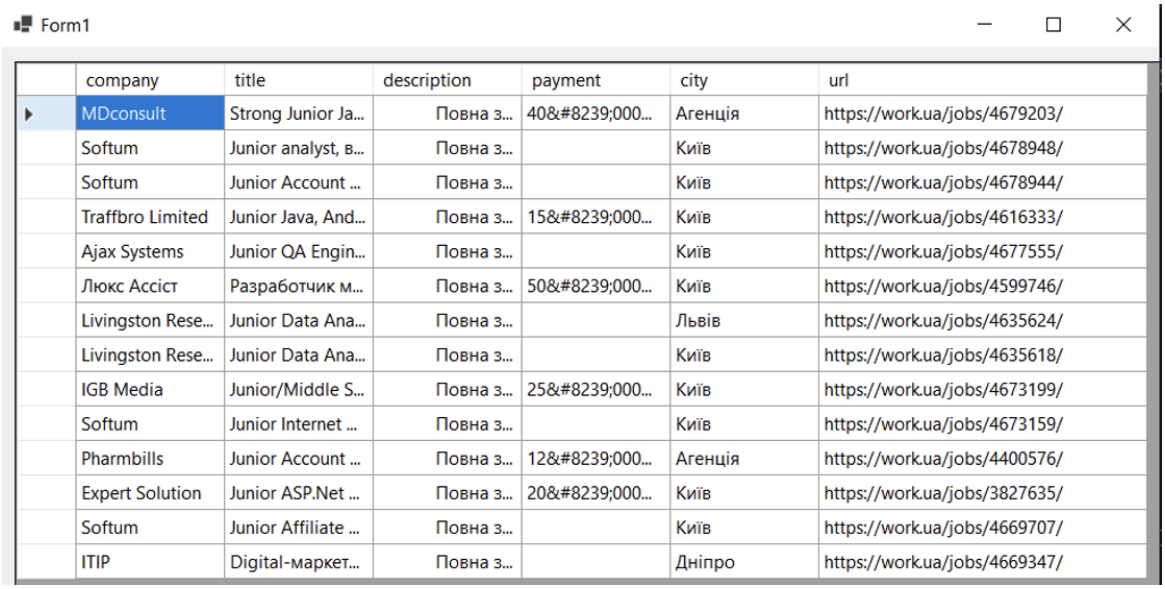
```
    string attributes = string.Empty;
                string nodeName = getNodeName(node);
                string nodeId = getIdValue(node);
                string nodeClass = getClassValue(node);
                if (isAttributes(nodeClass, nodeId) is
false)
                {
                    attributes += nodeName ;
                }
                else
```

3.8 Інтерфейс проєкту

3.8.1 Проектування інтерфейсу статичної системи автоматизованого збору та аналізу інформації

Основним критерієм при створенні інтерфейсу програмного застосунку було забезпечення зручного та зрозумілого способу відображення даних, отриманих в результаті обробки [24]. Інтерфейс розроблено з урахуванням того, що користувачеві не потрібно взаємодіяти з процесом збору та аналізу інформації.

Ілюстрація, наведена нижче (рис. 3.11), показує інтерфейс програмного застосунку для збору та аналізу даних.



company	title	description	payment	city	url
MDconsult	Strong Junior Ja...	Повна з...	40 000...	Агенція	https://work.ua/jobs/4679203/
Softum	Junior analyst, в...	Повна з...		Київ	https://work.ua/jobs/4678948/
Softum	Junior Account ...	Повна з...		Київ	https://work.ua/jobs/4678944/
Traffbro Limited	Junior Java, And...	Повна з...	15 000...	Київ	https://work.ua/jobs/4616333/
Ajax Systems	Junior QA Engin...	Повна з...		Київ	https://work.ua/jobs/4677555/
Люкс Асciст	Разработчик м...	Повна з...	50 000...	Київ	https://work.ua/jobs/4599746/
Livingston Rese...	Junior Data Ana...	Повна з...		Львів	https://work.ua/jobs/4635624/
Livingston Rese...	Junior Data Ana...	Повна з...		Київ	https://work.ua/jobs/4635618/
IGB Media	Junior/Middle S...	Повна з...	25 000...	Київ	https://work.ua/jobs/4673199/
Softum	Junior Internet ...	Повна з...		Київ	https://work.ua/jobs/4673159/
Pharmbills	Junior Account ...	Повна з...	12 000...	Агенція	https://work.ua/jobs/4400576/
Expert Solution	Junior ASP.Net ...	Повна з...	20 000...	Київ	https://work.ua/jobs/3827635/
Softum	Junior Affiliate ...	Повна з...		Київ	https://work.ua/jobs/4669707/
ITIP	Digital-маркет...	Повна з...		Дніпро	https://work.ua/jobs/4669347/

Рисунок 3.11 - Інтерфейс програмного застосунку для збору та аналізу даних

3.8.2 Проектування інтерфейсу динамічної системи автоматизованого збору та аналізу інформації

Основними критеріями при створенні інтерфейсу програмного застосунку були простота та зручність у використанні. Тільки за таких умов користувач зможе комфортно взаємодіяти з програмою, що, в свою чергу, сприятиме її популярності та залученню нових користувачів. Інтерфейс повинен бути розроблений так, щоб користувачам не були потрібні додаткові знання для роботи з програмним застосунком.

Ілюстрації, наведені нижче (рис. 3.12-3.14), показують інтерфейс програмного застосунку, створеного за допомогою WPF.

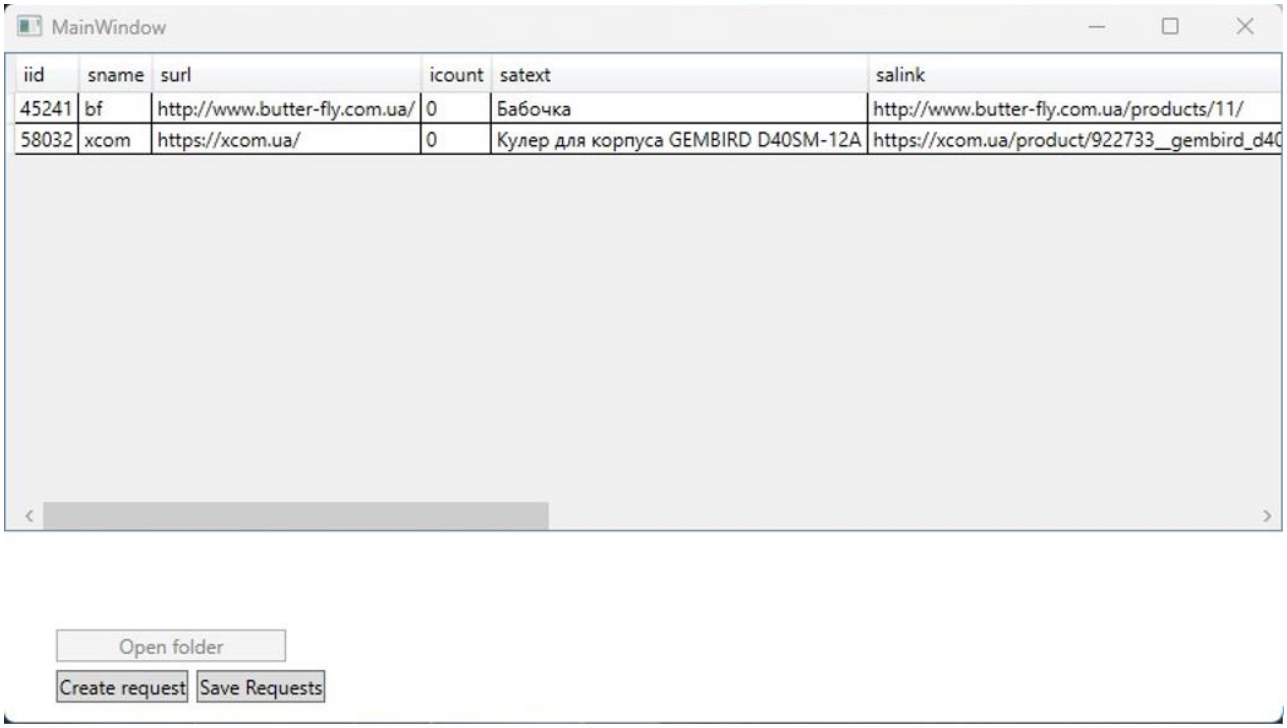


Рисунок 3.12 - Головне вікно

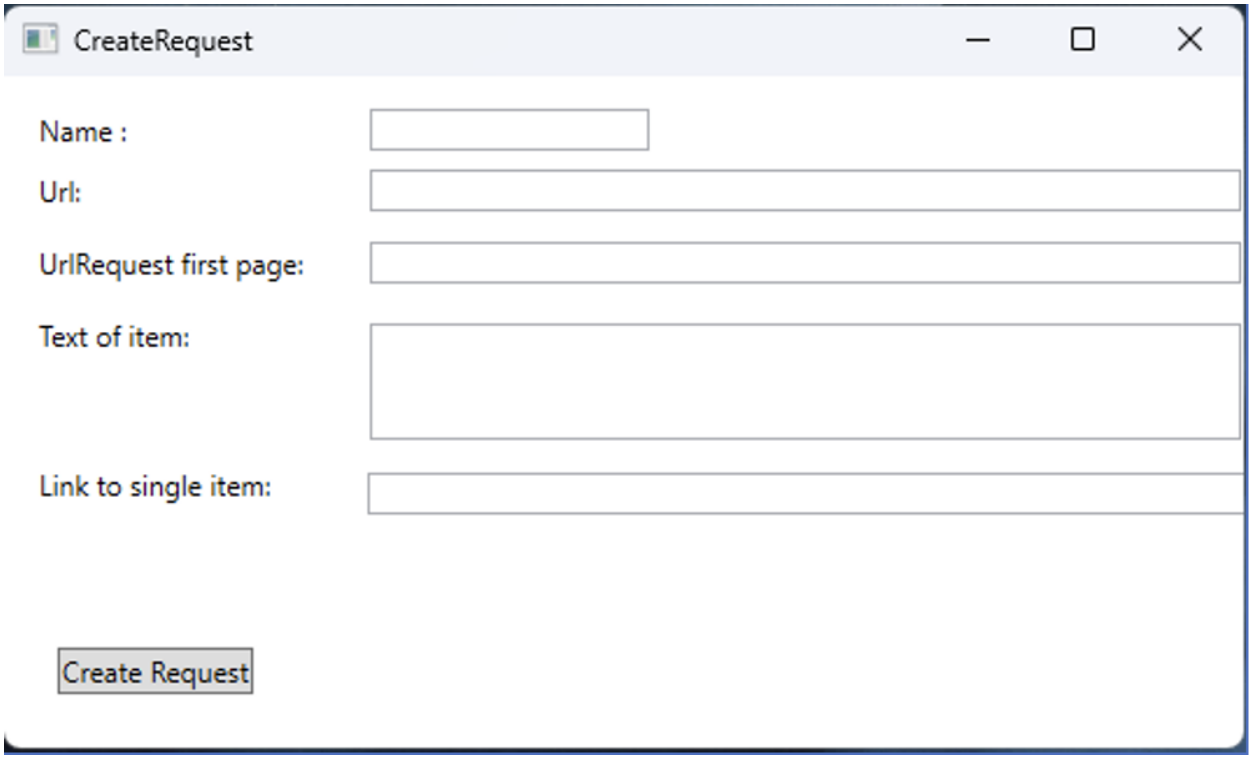


Рисунок 3.13 - Етап створення запиту

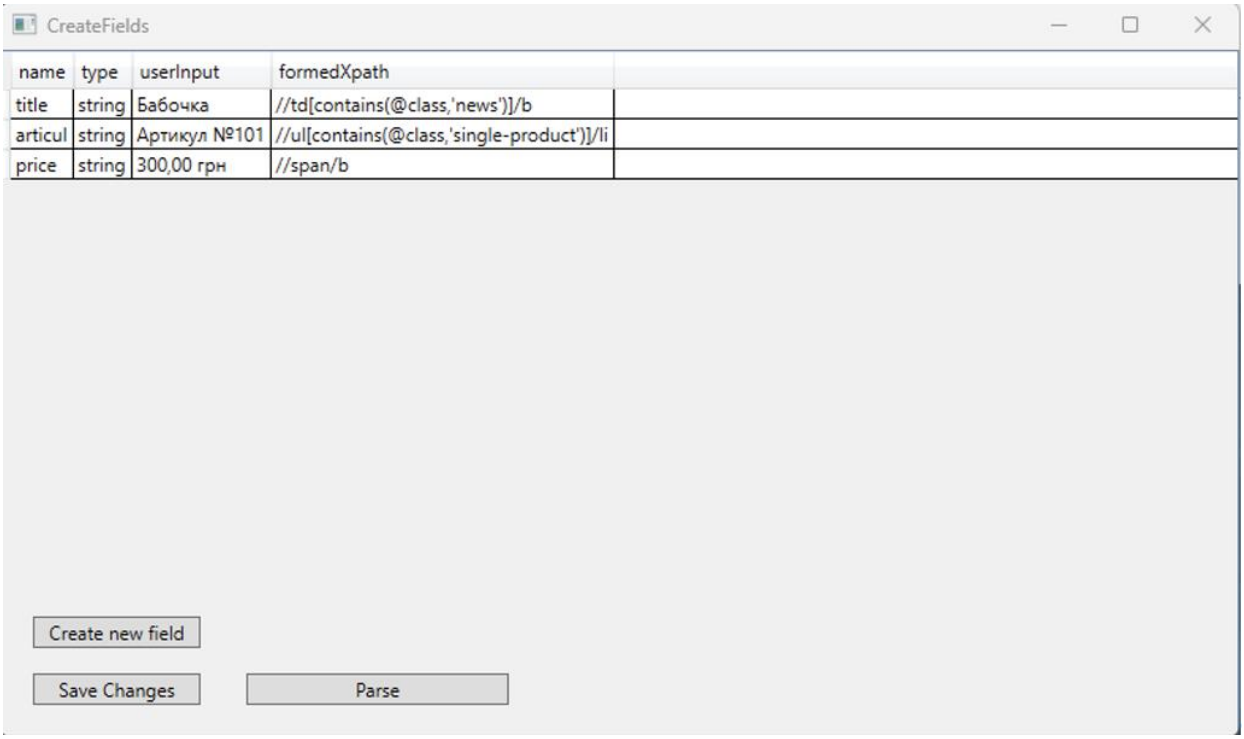


Рисунок 3.14 - Поля запитів

3.9 Висновки по розділу

У цьому розділі були реалізовані програмні застосунки як для статичного, так і для динамічного автоматизованого збору та аналізу даних з веб-сайтів. Проведено аналіз функціональних і нефункціональних вимог до програмних застосунків, а також побудовано діаграми класів, діаграми використання, діаграми послідовностей тощо. На основі виконаних дій можна зробити наступні висновки:

- реалізація програмних застосунків - успішно реалізовано програмні застосунки для статичного та динамічного автоматизованого збору і аналізу даних;

- діаграми класів - створено та описано діаграми класів для застосунків статичного і динамічного автоматизованого збору та аналізу інформації;

- використання HtmlAgilityPack - для побудови DOM-об'єкта та взаємодії з ним був застосований пакет HtmlAgilityPack;

- функціональність побудови запитів - програмний застосунок забезпечує можливість формування запитів;

- користувацький інтерфейс - інтерфейс програмного застосунку є простим та зрозумілим для користувачів;

- зберігання даних у форматі JSON - програмний застосунок використовує JSON-файл для зберігання даних.

4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ ФУНКЦІОНУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ ДЛЯ АВТОМАТИЗОВАНОГО ЗБОРУ І АНАЛІЗУ ІНФОРМАЦІЇ З ВЕБРЕСУРСІВ

4.1 Оцінка працездатності програмного застосунку для динамічного автоматизованого збору та аналізу інформації з веб-сайтів

Програмний застосунок для динамічного автоматизованого збору та аналізу інформації з веб-сайтів був протестований шляхом формування запитів до кількох веб-сайтів. На рис. 38 представлені результати роботи застосунку для створеного запиту. Було виявлено, що метод формування шляхів за допомогою Xpath має кілька недоліків, а саме:

- пошук за текстом - можливість здійснення пошуку за текстом існує лише тоді, коли текст на HTML-сторінці та у запиті повністю збігаються, включаючи регістр;

- кодування спеціальних символів - існують проблеми з кодуванням спеціальних символів, таких як нерозривний пробіл, у таких випадках користувачеві потрібно вказувати правильне кодування елемента, наприклад, « »;

- дублювання інформації на сторінці - оскільки запити формуються на основі текстових виразів, можуть виникати ситуації, коли Xpath створюється для іншого елемента сторінки, що призводить до дублювання даних;

- дублювання CSS-селекторів - у деяких веб-сайтах структура документа побудована таким чином, що однакові CSS-селектори можуть використовуватися для різних елементів (наприклад, технічні характеристики та детальний опис), що може призвести до того, що Xpath, сформований на основі селекторів, вказуватиме на кілька схожих елементів одночасно.

Також було виявлено вимоги до формату веб-сайтів, з яких ми формуємо запити. Інформація, що підлягає збору, повинна бути присутня на кожній сторінці (вакансії, товару, послуги тощо). Щоб поліпшити ці недоліки, реалізовано можливість внесення змін безпосередньо у сформовані Xpath. Зміни, які вносяться, активуються з моменту натискання кнопки «Save» користувачем. Програмний застосунок призначений для роботи з веб-застосунками типу WPA.

Варіанти покращення програмного застосунку.

На основі проведеного дослідження, аналізу взаємодії користувача з веб-сайтом та процесу аналізу результатів роботи програмного застосунку, можна запропонувати такі покращення:

- реалізація переходів між знайденими елементами - додати функціональність, що дозволяє користувачеві переходити між елементами, які були знайдені за текстовою інформацією, і побудувати шлях безпосередньо до них;

- реалізація переходу до батьківського елемента - забезпечити можливість користувачу повернутися до батьківського елемента, що дозволить легше навігувати по структурі даних;

- гнучкі налаштування URL адрес - впровадити більш гнучкі можливості налаштування URL-адрес та механізмів взаємодії з ними, що забезпечить краще управління запитами та збільшить ефективність збору даних.

4.2 Висновки до розділу

У цьому розділі було проведено дослідження різних факторів, що впливають на ефективність роботи програмного застосунку. Для покращення результатів його функціонування були запропоновані кілька варіантів подальшого розвитку.

ВИСНОВКИ

В ході виконання дипломної роботи був спроектований і реалізований програмний застосунок для автоматизованого збору та аналізу інформації з веб-сайтів. На основі проведених досліджень можна зробити такі висновки:

- досліджено проблеми, пов'язані з розробкою програмних застосунків для автоматизованого збору та аналізу інформації;

- проведено теоретичні дослідження в галузі систем автоматизованого збору та аналізу даних;

- вивчені засоби, бібліотеки та інструменти, що можуть бути використані для вирішення поставленої задачі;

- реалізовано та проведено перевірку працездатності програмного застосунку для автоматизованого збору та аналізу інформації з веб-сайтів;

- визначено переваги та недоліки розробленого рішення для універсального автоматизованого збору та аналізу даних;

- сформульовано варіанти покращення програмного застосунку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Management information systems for the information age Stephen Haag, Maeve Cummings, Amy Phillips, McGraw-Hill/Irwin, Boston, 2020 – 672 p.

2. Методи аналізу даних [Електронний ресурс]. – Режим доступу: http://web.kpi.kharkov.ua/auts/wpcontent/uploads/sites/67/2017/02/DAMAP_Ivashko_posobie2.pdf (дата звернення 13.09.2024).

3. The Universal Parser Compiler and Its Application to a Speech Translation System Masaru Tomita, Marion Kee, Hiroaki Saito, Teruko Mitamura and Hideto Tomabechi. URL: https://aclanthology.org/1988.tmi-1.9/ (Last accessed: 01.10.2024).

4. Parsing of HTML Document Pranit C. Patil , Pramila M. Chawan , Prithviraj M. Chauhan. URL: https://www.researchgate.net/profile/Pramila- Chawan/ publication/316510678_Parsing_of_HTML_Document/links/59019 1270f7e9bcf6546a9a4/Parsing-of-HTML-Document.pdf (Last accessed: 27.09.2024).

5. The Universal Parser Compiler and Its Application to a Speech Translation System Masaru Tomita, Marion Kee, Hiroaki Saito, Teruko Mitamura and Hideto Tomabechi. URL: https://aclanthology.org/1988.tmi-1.9/ (Last accessed: 03.10.2024).

6. Офіційний ресурс про HTML специфікацію WHATWG. URL: https://html.spec.whatwg.org/multipage/ (дата звернення 01.10.2024).

7. Інформаційний документ, що містить технічні специфікації Https. URL: https://www.rfc-editor.org/rfc/rfc2818 (дата звернення 18.09.2024).

8. E. Turban Information Technology for Management: Advancing Sustainable, Profitable Business Growth Paperback – Realtime Publishers, 2020 – 92p.

9. Що таке DOM або об’єктна модель документа https://foxminded.ua/dom-tse/ (дата звернення 21.09.2024).

10. Багато-сторінкові (MPA), односторінкові (SPA) та прогресивні (PWA). https://embo.com.ua/uk/blog/odnostorinkovi-spa-i-bagatostorinkovi-pwa/ (дата звернення 29.09.2024).

11. Вплив пагінацію на внутрішню оптимізацію сайту і просування https://cityhost.ua/uk/blog/chto-takoe-paginaciya-sayta-i-kak-ee-nastroit.html (дата звернення 11.10.2024).

12. The Future of Web Scraping Projects:. URL: https://astroproxy.com/ua/blog/shho-take-veb-skreiping-i-yak-vin-povyazanii-z-proksi (Last accessed: 27.09.2024).

13. Powerful Web Scraping Software. URL: https://www.parsehub.com/features (Last accessed: 17.10.2024).

14. Використання Puppeteer для автоматизації тестування веб-додатків. URL: https://it-rating.ua/vikoristannya-puppeteer-dlya-avtomatizatsii-testuvannya-veb-dodatkiv (дата звернення 11.10.2024).

15 Scrapy at a glance. URL: https://doc.scrapy.org/en/latest/intro/overview.html (Last accessed: 07.09.2024).

16. Hypertext Transfer Protocol (Http/1.1): Semantics and Content RFC 7231 . URL: https://datatracker.ietf.org/doc/rfc7231/ (Last accessed 07.11.2024).

17. О.В. Васильєв Програмування на мові Python . 2018 – 504 с.

18. The Universal Parser Compiler and Its Application to a Speech Translation System Masaru Tomita, Marion Kee, Hiroaki Saito, Teruko Mitamura and Hideto Tomabechi. URL: https://aclanthology.org/1988.tmi-1.9/ (Last accessed 01.11.2024).

19. Parsing of HTML Document Pranit C. Patil , Pramila M. Chawan , Prithviraj M. Chauhan . URL: https://www.researchgate.net/profile/Pramila-Chawan/publication/316510678_Parsing_of_HTML_Document/links/59019 1270f7e9bcf6546a9a4/Parsing-of-HTML-Document.pdf (Last accessed 27.10.2024).

20. Windows Presentation Foundation. URL: https://iteacorp.com/course/prilozheniya-windows-na-wpf/ (Last accessed 05.11.2024).

21. Об'єктно-орієнтоване програмування. Частина 1. Григорович В.Г., 2023 – 284 стор.

22. Eric Evans. Domain-Driven Design: Tackling Complexity in the Heart of Software. 2020 – 534 p.

23. Unified Modeling Language. UML: https://evergreens.com.ua/ua/articles/uml-diagrams.html (Last accessed 23.10.2024).

24. Проектування інформаційних систем: навчальний посібник / В.С. Авраменко, А.С. Авраменко. – Черкаси: Черкаський національний університет ім. Б. Хмельницького, 2017. – 434 с..