МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ/факультет	Інформаційних технологій
Кафедра	Інформатики і прикладного програмного забезпечення
Спеціальність	Інженерія програмного забезпечення
Форма навчання	Денна

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Волошина Данила Андрійовича
(прізвище, ім'я, по батькові здобувача)

на тему

«Розробка парсеру веб-сторінок»
(повна назва теми)

за матеріалами праць провідних спеціалістів з розробки ПЗ та проектування БД
(повна назва бази дослідження)

науковий керівник к.е.н., доц. _____ Баран С.В.
(наук. ступінь, вчене звання) *(підпис)* *(прізвище, ініціали)*

Робота допущена до захисту в ЕК
Протокол засідання кафедри
від 11.06.2025р. № 12
Завідувач кафедри _____
(підпис)

д.т.н., професор Зеленський О.С.
Наук. ступень, вчене звання *Ініціали, прізвище*

Кривий Ріг – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ/факультет	Інформаційних технологій
Кафедра	Інформатики і прикладного програмного забезпечення
Спеціальність	Інженерія програмного забезпечення
Форма навчання	Денна

«ЗАТВЕРДЖУЮ»
Завідувач кафедри _____ Зеленський О.С.
(підпис) (Прізвище, ініціали)
«11» червня 2025 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ

1. Тема роботи «Розробка парсеру веб-сторінок»

Керівник роботи к.е.н., доц. Баран С.В.
затверджені наказом закладу вищої освіти від «04» квітня 2025 р. № 222-ст

2. Строк подання здобувачем роботи до «09» червня 2025 р.

3. Зміст кваліфікаційної роботи, об’єкт, предмет та мета дослідження:

Розділ 1. Постановка задачі

Розділ 2. Розробка алгоритмів парсингу

Розділ 3. Розробка інформаційного забезпечення

Розділ 4. Розробка програмного забезпечення

Об’єкт дослідження: інформаційні ресурси Інтернету у вигляді веб-сторінок та процес автоматизованого збору та обробки їх вмісту за допомогою сучасних технологій
Предмет дослідження: методи й алгоритми парсингу веб-сторінок за сценаріями та тегами, реалізовані в архітектурі ASP.NET MVC
Мета кваліфікаційної роботи: *розробити програмне забезпечення для автоматичного парсингу інформації з заданого масиву посилань.*

5. Дата видачі завдання «04» квітня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів МДР	Строк виконання етапів роботи	Відмітка керівника про виконання етапів (дата, підпис)
1	Підготовка розділу 1	04.04.2025-13.04.2025	
2	Підготовка розділу 2	14.04.2025-26.04.2025	
3.	Підготовка розділу 3	27.04.2025-15.05.2025	
4	Підготовка розділу 4	16.05.2025-08.06.2025	
5	Реєстрація завершеної кваліфікаційної роботи	09.06.2025	Реєстраційний №___ «09»червня 2025 р.
6	Отримання відгуку від наукового керівника	10.06.2025	
7	Подання кваліфікаційної роботи на перегляд завідувачу кафедри	11.06.2025	
8	Отримання зовнішньої рецензії	12.06.2025	
9	Попередній захист кваліфікаційної роботи на кафедрі	13.06.2025	
10	Підготовка до захисту в ЕК	16.06.2025-21.06.2025	

Завдання підготував науковий керівник _____ Баран С. В.
(підпис) (прізвище та ініціали)

Завдання одержав _____ Волошин Д.А
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

на кваліфікаційну бакалаврську роботу

«Розробка парсеру веб-сторінок»

Волошина Данила Андрійовича

Кваліфікаційна бакалаврська робота на здобуття освітньо-кваліфікаційного рівня бакалавра зі спеціальності 121 «Інженерія програмного забезпечення» – Державний університет економіки і технологій – Кривий Ріг, 2025.

У бакалаврській дипломній роботі розроблено програмний комплекс для автоматизованого збору й обробки даних із веб-сторінок. Веб-додаток створено на базі ASP.NET MVC із використанням бібліотеки HTML Agility Pack для парсингу та технології ADO.NET для збереження сценаріїв парсингу у СУБД Microsoft Access. На виході формується файл у форматі CSV/JSON, який потім візуалізується в окремому десктоп-додатку, реалізованому на C++/MFC із використанням OpenGL для графічного представлення даних.

Ключові слова: ПАРСИНГ ВЕБ-СТОРІНОК, ASP.NET MVC, HTML AGILITY PACK, ADO.NET, MICROSOFT ACCESS, C++/MFC, OPENGL, CSV, JSON.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД	(база даних) Упорядкований набір логічно взаємопов'язаних даних, що використовуються спільно та призначені для задоволення інформаційних потреб користувачів
СУБД	(система управління базами даних) Програмний комплекс, призначений для створення, управління й обслуговування баз даних
ПЗ	(програмне забезпечення) Комплект програмних засобів та документації, необхідний для виконання певних функцій на комп'ютері
ADO .NET	(ActiveX Data Objects для .NET) Технологія доступу та управління базами даних для платформи .NET, що забезпечує уніфікований інтерфейс роботи з різними СУБД
ASP.NET MVC	(Active Server Pages .NET Model-View-Controller) Фреймворк від Microsoft для побудови веб-додатків згідно з архітектурною моделлю «модель–представлення–контролер»
HTML	(HyperText Markup Language) Мова розмітки гіпертексту, основа веб-сторінок у мережі Інтернет.
HAP	(HTML Agility Pack) Бібліотека для .NET, що дозволяє завантажувати, аналізувати та модифікувати HTML-документи.
MVC	(Model-View-Controller) Шаблон архітектури програмного забезпечення, який розділяє додаток на модель, представлення та контролер
MFC	(Microsoft Foundation Classes) Набір бібліотек для розробки десктоп-додатків під Windows із використанням C++.

ЗМІСТ

ВСТУП

У сучасному цифровому середовищі обсяг інформації, розміщеної на веб-сайтах, зростає експоненціально. Інтернет-ресурси охоплюють різноманітні галузі: від електронної комерції до наукових публікацій, від соціальних мереж до спеціалізованих аналітичних платформ. Разом з цим виникає необхідність автоматизованого збору, обробки та аналізу даних із веб-сторінок — завдання, яке здобуває все більшої актуальності як у бізнесі, так і в дослідницькій та освітній сферах. Ручний моніторинг сайтів і викачування даних стає неефективним через великий обсяг інформації, швидкі зміни структури сторінок та непостійність форматів подання контенту.

В Україні зростає потреба підприємств і дослідників у швидкому отриманні актуальних даних (наприклад, цінових пропозицій конкурентів, новин на тематичних порталах, інформації про вакансії тощо). При цьому більшість ресурсів не надають відкритих API або ж обмежують доступ до інформації. У таких умовах розробка гнучкого веб-парсеру, який дозволяє задати масив URL і набори селекторів (сценаріїв і тегів), стане важливим інструментом для автоматизації збору інформації. Збереження результатів у файл CSV/JSON забезпечить подальшу обробку й аналіз, а десктоп-візуалізатор на C++/MFC з OpenGL зробить дані наочними для користувача.

Актуальність роботи пояснюється такими факторами:

- широким використанням веб-даних у маркетингових дослідженнях, конкурентному аналізі та аналітичних звітах;
- необхідністю мінімізувати витрати часу та ресурсів на ручне копіювання інформації з веб-сторінок;
- потребою у візуалізації великих масивів даних для оперативного прийняття управлінських рішень;
- обмеженим доступом до API багатьох популярних українських і міжнародних сайтів, що вимагає використання кастомізованих рішень.

Метою кваліфікаційної роботи є розробка програмного комплексу, який складається з веб-додатку на базі ASP.NET MVC із використанням бібліотеки HTML Agility Pack та СУБД Microsoft Access (через ADO .NET) для парсингу веб-сторінок за сценаріями, а також десктоп-візуалізатора на C++/MFC із застосуванням OpenGL для графічного відображення зібраних даних у форматі CSV.

Ядром проєкту є:

Створення механізму зберігання сценаріїв і тегів у базі Access, що дає змогу гнучко налаштовувати правила парсингу без зміни коду.

Реалізація веб-інтерфейсу ASP.NET MVC, який приймає обрану конфігурацію, проводить асинхронний парсинг HTML-документів через HTML Agility Pack і формує фінальний CSV або JSON файл із зібраною інформацією.

Розробка десктоп-додатка на C++/MFC з OpenGL, який зчитує згенеровані CSV файли та відображає їх у вигляді таблиці для аналізу користувачем у реальному часі.

У роботі використано такі методи дослідження: аналіз технічної літератури та документації, моделювання архітектури програмного забезпечення, об’єктно-орієнтоване програмування на C# та C++, застосування OpenGL, застосування ADO .NET для роботи з Access, використання HTML Agility Pack для парсингу HTML.

Результатом кваліфікаційної роботи стане комплексне програмне рішення, яке зможе автоматизувати процес збору інформації з веб-сторінок, а також забезпечувати наочну візуалізацію зібраних даних, що сприятиме підвищенню ефективності прийняття рішень у різноманітних прикладних сферах.

РОЗДІЛ 1
ПОСТАНОВКА ЗАДАЧІ

1.1. Характеристика задачі

Парсинг веб-сторінок — це процес автоматичного збору інформації зі сторінок у мережі Інтернет у вигляді структурованих даних, які можна одразу використовувати для аналізу, звітів чи інших цілей. У різних галузях парсер стає незамінним інструментом: в електронній комерції він допомагає моніторити ціни конкурентів і оновлювати власний асортимент, у фінансових сервісах збирає дані про курси валют і котирування цінних паперів зі сторінок бірж, в медіа-аналітиці автоматизує збір заголовків новин і трендових статей для формування добірок чи тематичних оглядів. Наукові дослідники, особливо в соціальних та гуманітарних сферах, використовують парсери для збору даних із сайтів державних статистичних служб або соціальних мереж, аби опрацьовувати великі масиви інформації без ручного введення. [6]

У класичних випадках парсер звертається до веб-серверів за адресами, що зберігаються у налаштуваннях, надсилає HTTP-запити (зазвичай GET) і отримує візуальний HTML-код сторінки. Потім цей сирий HTML перетворюється всередині парсера на внутрішню структуру, яку часто називають DOM-деревом: кожен тег, кожен атрибут і кожен шматок тексту стають вузлами цього дерева, доступними для подальшого аналізу. Завдяки цьому можна точно вказати, де знаходиться інформація, яка нас цікавить: назва товару, ціна, опис, рейтинг, дата публікації статті або будь-які інші елементи. Завдяки виразам XPath чи CSS-селекторам ми “промовляємо” до парсера: знайди у DOM-дереві всі блоки певного класу або конкретні теги, інакше кажучи — отримаємо точні координати шуканого вмісту. [5]

Насправді парсинг вимагає враховувати специфіку кожного ресурсу. Одна й та сама інформація може бути реалізована зовсім різними способами, наприклад: кнопка “купити” може знаходитися в із певним класом на

одному сайті, а на іншому — в всередині списку. Для того щоб парсер не ламався після оновлення дизайну або структури сайту, правила парсингу сторінок зберігають за межами коду наприклад, у конфігураціях. З цього випливає ключовий принцип: якщо структура змінилася, замінюються лише налаштування (шляхи XPath, назви класів, атрибути), а не доводиться переписувати кожен фрагмент програми. Така гнучкість особливо важлива для українських компаній, у яких може бути обмежений технічний ресурс або відсутня можливість постійно доопрацьовувати власний інструмент під кожен новий сайт.

У сучасному Інтернеті чимало сторінок формуються динамічно за допомогою JavaScript і AJAX, тож простий HTTP GET вже не завжди повертає остаточний вміст, який бачить користувач у браузері. Наприклад, у випадку маркетплейсів або фреймворків типу React і Vue сторінки можуть завантажувати “порожню” оболонку, а всі дані підвантажуються через API на клієнті. У такому разі базовий HTML буде без потрібних блоків із цінами чи відгуками, і звичайний парсер нічого не знайде. Щоб вирішити цю проблему, дехто переходить до використання headless-браузерів (Selenium, Playwright) або спеціальних сервісів, які підтримують виконання JavaScript-коду, імітуючи роботу повноцінного браузера. Цей підхід суттєво ускладнює розгортання парсера та зростає витрата ресурсів.

Ще одна суттєва складність — це мінливість структури ресурсів. Власники сайтів часто змінюють розмітку або назви CSS-класів, а іноді додають нові блоки реклами, які зсувають позицію шуканих елементів. У таких умовах парсер має бути здатним працювати стабільно: наприклад, сканувати по кількох альтернативних XPath-шляхах, реагувати на відсутність очікуваних елементів або видавати пусті результати замість помилок. Оскільки в Україні багато малих та середніх компаній не витрачають кошти на надання відкритих API, часто доводиться “розбирати” сайти вручну й тримати в базі налаштування для десятків різних ресурсів — від досить простих інтернет-магазинів до великих агрегаторів вакансій чи оголошень.

Після того, як необхідні вузли знайдені, парсер бере з них текстову частину або значення атрибутів, очищує від зайвих символів та зайвих HTML-тегів, переформатовує дати й числа у єдиний формат (наприклад, українську дату “03.06.2025” перетворює у “2025-06-03” або число “1 234,56” у “1234.56”). Цей етап очищення й нормалізації дозволяє уникнути помилок під час аналізу зібраних даних у BI-інструментах або під час збереження в базу. Наприклад, якщо парсер одразу передає інформацію в CSV або JSON для подальшого імпорту в Excel чи Python, важливо, щоб формат числа був у прийнятному для цих систем вигляді, а не “345 грн.” чи “2 500 – 3 000”.

Збереження результатів відбувається у формі, придатній для подальшої обробки. Найчастіше це CSV-файл або JSON-об’єкт, які передаються користувачу або відразу заносяться до вбудованої СУБД. У бізнесі це дозволяє аналітикам за потреби швидко підвантажити дані в табличну програму чи в модуль звітності. У наукових проєктах результати парсингу можуть іти на вхід у статистичні обчислення, кластеризацію чи машинне навчання. Наприклад, лінгвісти можуть збирати тексти статей із сайтів новин, а соціологи — статистику вакансій регіонального рівня для дослідження ринку праці в окремих областях України.

Таким чином, у тих галузях, де інформація змінюється дуже часто або обсяги занадто великі для ручного опрацювання, парсинг стає незамінним. Інтернет-магазини оцінюють ціни конкурентів щохвилинно, торгівельні платформи збирають відгуки користувачів практично в реальному часі, а медіа-агрегатори формують новинні стрічки для своїх користувачів, базуючись на результатах парсерів. Використання веб-парсингу охоплює найрізноманітніші випадки: від моніторингу курсів валют та динаміки цін на криптовалюти до збору контактів потенційних клієнтів чи компіляції статистики публічних тендерів. Однак для кожного з цих випадків важливо враховувати та долати особливі виклики: різноманітність HTML-розміток, обмеження зі сторони серверів (CAPTCHA, блокування за IP, перевірка User-Agent) та нестабільність структури сайтів. Завдяки використанню шаблонів для конфігурації,

асинхронного завантаження сторінок та гнучких засобів обробки помилок, система парсингу здатна надійно служити у широкому спектрі завдань, забезпечуючи необхідний рівень автоматизації та оперативності.

1.2 Огляд існуючих рішень

У середовищі, де часом немає можливості витрачати тижні на розробку власного парсера, широко застосовуються готові рішення, які дозволяють швидко налаштувати збір даних із веб-сторінок без написання коду. Нижче наведені приклади найбільш поширених інструментів, кожен із яких має свої особливості, переваги та обмеження.

- Octoparse

Octoparse — це десктоп-додаток із візуальним конструктором парсинг-сценаріїв, який працює на Windows. Користувач відкриває вбудований браузер у межах програми, обирає елементи сторінки, і Octoparse підхоплює відповідні XPath або CSS-селектори. Після цього можна вказати, які дані зберігати (текст, атрибути, посилання) і як переходити між сторінками, якщо необхідна пагінація. Результати зберігаються у CSV, JSON, Excel. Серед мінусів — обмеження безкоштовної версії (кількість паралельних потоків, об’єм результатів на місяць) і необхідність запуску клієнтської програми, що вирішується тільки платною підпискою. Інтерфейс програми зображено на Рис. 1.1.

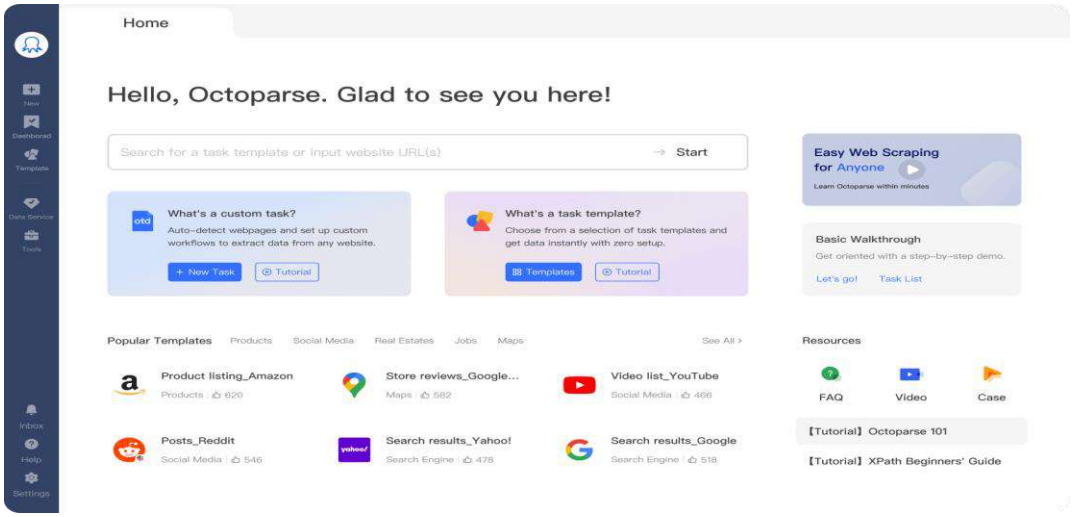


Рис. 1.1. Інтерфейс користувача Octoparse

- Import.io

Import.io — це хмарний SaaS-сервіс, який орієнтований на підприємства. Після реєстрації користувач завантажує URL сторінки у веб-інтерфейс, а Import.io інтелектуально намагається знайти повторювані патерни (наприклад, рядок товарів у каталозі). Якщо алгоритм помилково класифікує елементи, користувач може вручну позначити, які області містять потрібні дані (назва, ціна, зображення тощо). Після цього платформа сама генерує сценарій і починає збір. Import.io підтримує інтеграцію з Google Sheets, Zapier і навіть власний API для розробників, а також має вбудовані механізми обходу блокувань (ротація проксі, зміна User-Agent). Головний недолік — висока вартість корпоративних планів, яка може бути виправданою лише для великих компаній, що опрацьовують сотні тисяч URL щодня. Зовнішній вигляд програмного забезпечення показано на Рис. 1.2.

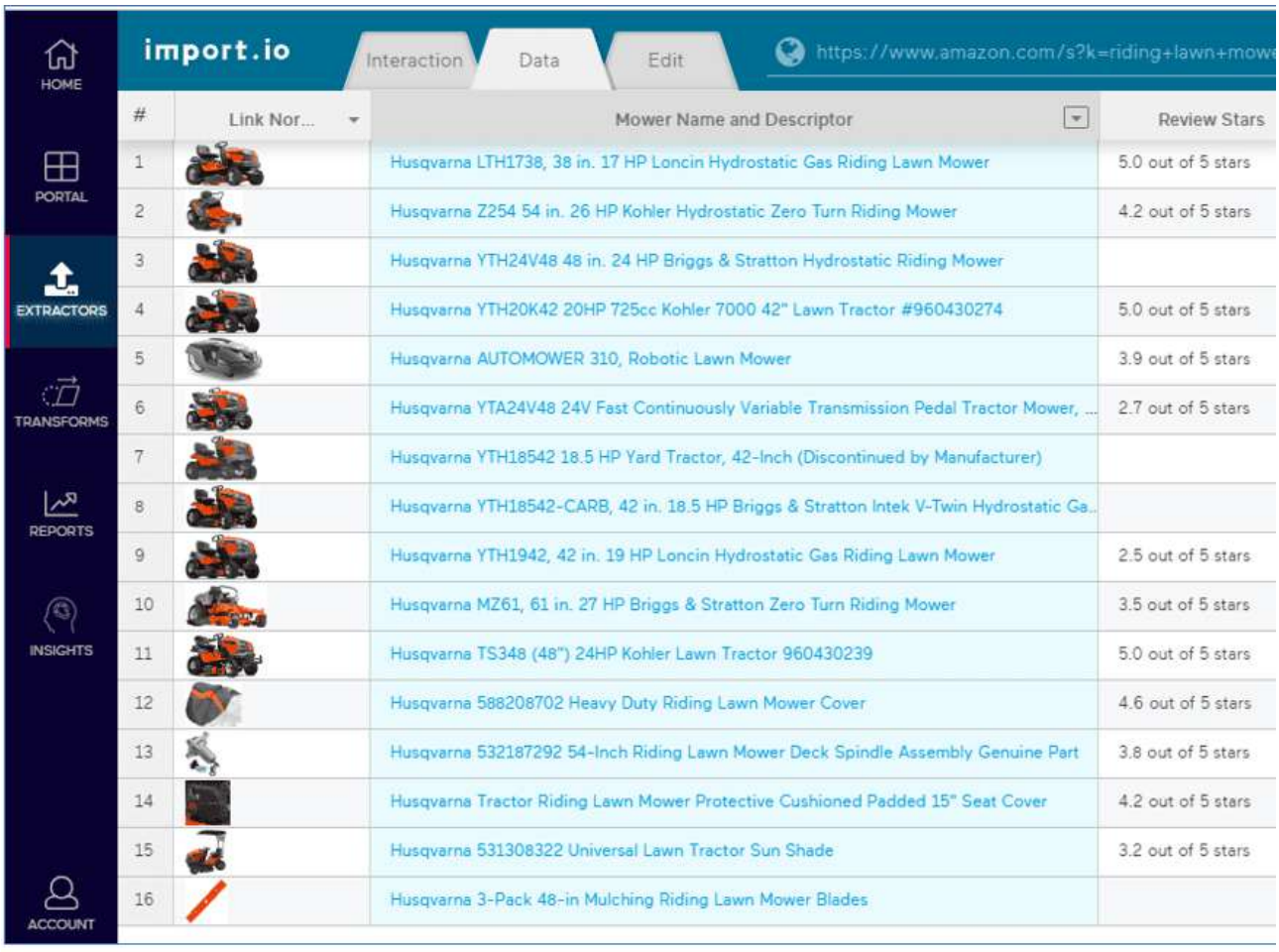


Рис. 1.2. Інтерфейс користувача Import.io

- ParseHub

ParseHub працює як у вигляді десктоп-програми (Windows, macOS, Linux), так і через власний хмарний сервіс. Він створює “віртуального користувача”: у графічному інтерфейсі користувач обирає елементи сторінки, а ParseHub генерує візуальну логіку переходів, умов і циклів. Крім витягання статичних елементів, ParseHub дозволяє взаємодіяти з динамічними компонентами — натискати кнопки “Загрузити ще”, авторизуватися тощо. Зібрані дані автоматично експортуються у CSV або JSON або можуть бути передані через API. У безкоштовного тарифу обмежена кількість одночасних завдань і сторінок на виконання, а в платного — доступні необмежена кількість проєктів і швидкість обробки. ParseHub особливо популярний серед маркетологів і аналітиків, які швидко налаштовують збір інформації з оголошень, каталогів товарів чи відгуків. Інтерфейс програми ParseHub зображено на Рис. 1.3.

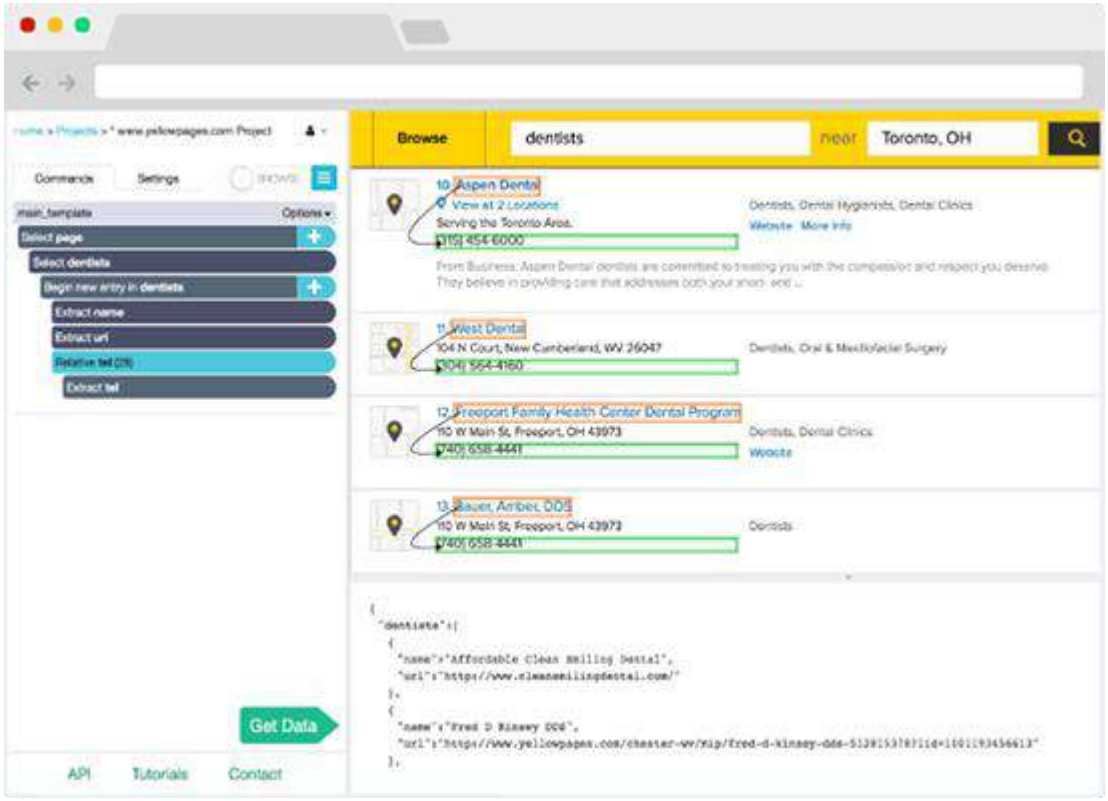
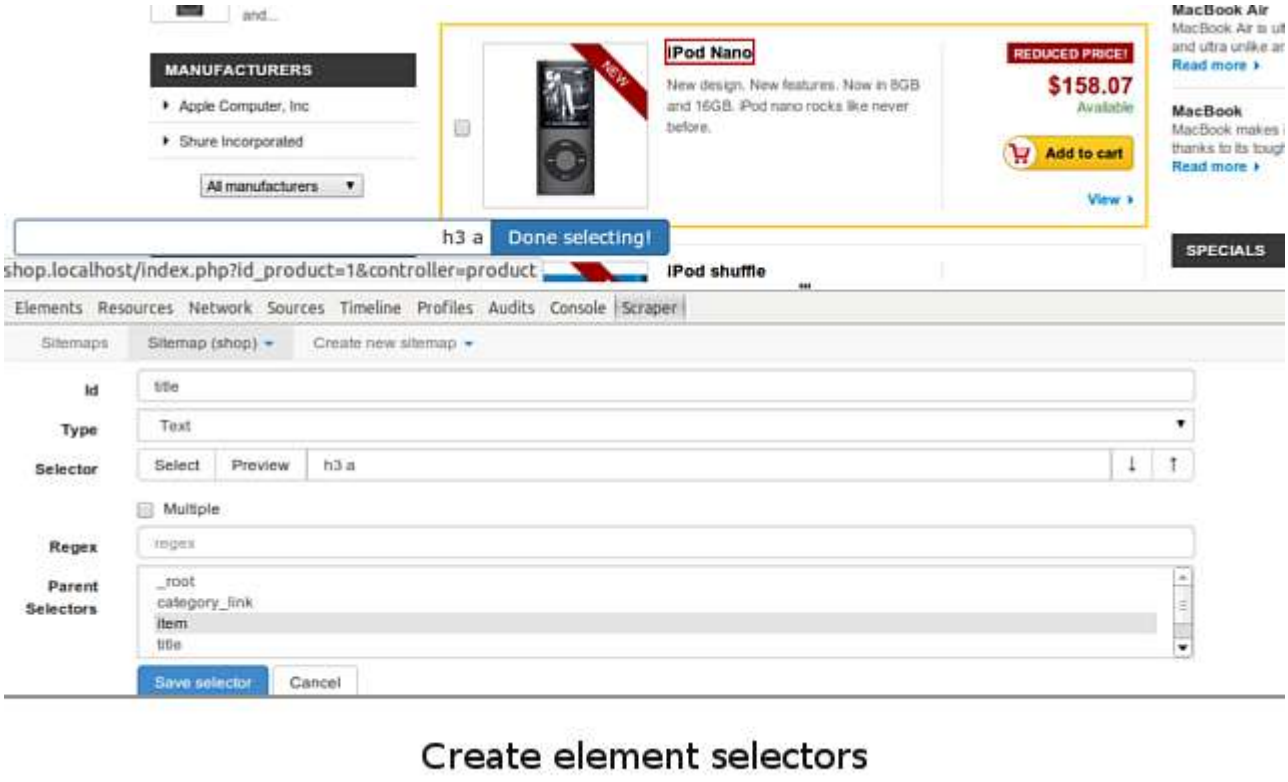


Рис. 1.3. Інтерфейс користувача програми ParseHub

Web Scraper (додаток для Chrome)

Web Scraper — це розширення для Google Chrome із візуальним інтерфейсом, яке дозволяє легко створювати карту сайту (sitemap) у вигляді набору правил (selectors). Після інсталяції розширення у вікні інструментів розробника можна клацати на будь-які елементи веб-сторінки, призначати їм імена і вказувати, чи потрібно збирати текст, атрибути чи кліки для навігації. Після налаштування sitemap скрипт автоматично переключається між сторінками, переходить за посиланнями, завантажує та форматує дані. Результат зберігається у CSV, JSON. Web Scraper підходить для швидкого прототипування, водночас може не впоратися з більш складними сайтами, та вимогами до збору даних. Інтерфейс користувача додатку Web Scraper зображено на Рис. 1.4.



Create element selectors

Рис. 1.4. Інтерфейс користувача додатку Web Scraper

1.3. Аналіз вимог до розробки

У цьому підрозділі описані ключові вимоги до розробки системи парсингу — зокрема, що саме надходить на вхід і який результат очікується на виході.

Спочатку визначається структура вхідних даних: сценарії парсингу з їхніми правилами та XPath-виразами, набір тегів, що задають поля для вибірки, а також перелік URL-адрес, які користувач хоче обробити. Далі описано формат вихідних даних — структуровані файли у форматах CSV та JSON, деталі їхньої організації та відповідності встановленим у сценарії ключам. Такий підхід дозволяє чітко окреслити, які саме дані потрібні для роботи програми, й яким чином він має повернути результат для подальшого аналізу. [5]

Вхідними даними для програмного комплексу є:

1. Сценарії парсингу (Parsing Scenario):
    - Кожен сценарій описує певний набір правил, XPath-виразів, які використовуються для витягнення тексту, або атрибутів з заданих елементів HTML – сторінки.
    - У базі Access міститься таблиця ParsingScenarios зі структурою:
        - Id – унікальний ідентифікатор сценарію (GUID)
        - Name – назва сценарію (текстова строка)
        - ContainerTag – тег контейнеру, з якого буде витягатися інформація (текстова строка)
        - ContainerClass – клас контейнеру, з якого буде витягатися інформація (текстова строка)
2. Набір тегів (Parsing Tag):
    - Для кожного сценарію у таблиці ParsingTags зберігаються записи:
        - Id – унікальний ідентифікатор тегу (GUID)
        - ScenarioId – зовнішній ключ на ParsingScenarios.Id
        - HtmlTag – тег елементу з якого буде взято текстову інформацію, або значення атрибуту. (текстова строка)
        - Class – клас елементу з якого буде взято текстову інформацію, або значення атрибуту. (текстова строка)

- Text – опціональний параметр для пошуку елементу за текстом (текстова строка)
- ParsingType – параметр для специфікації парсингу значення атрибуту/тексту з обраного елемента (текстова строка)
- AttributeName – опціональний параметр для специфікації назви атрибуту, значення якого користувач хоче отримати.
- Key – визначає до якого ключа буде записано отриману інформацію по цьому тегу (текстова строка)

3. Масив посилань (Scenario Urls):
   - Користувач у веб-формі передає перелік посилань.
   - Кожне посилання записується у таблицю ScenarioUrls.
   - Id – унікальний ідентифікатор – лічильник посилань (лічильник)
   - ScenarioId - зовнішній ключ на ParsingScenarios.Id
   - Url – зберігає посилання на сторінку, з якох буде діставатися інформація (текстова строка).

Результатом роботи системи є:

- Файли структурованих даних у форматі CSV.
  - Перша строка таблиці буде містити в собі назву сценарію
  - Друга строка буде мати в собі введен і користувачем ключі, до яких будуть записуватися витягнуті дані.
  - В таблицях кожен рядок відповідає витягнутому тексту з елементу.
  - Роздільник CSV – крапка з комою, щоб уникнути конфлікту з комами в текстових полях.
- Файли структурованих даних у форматі JSON.
  - Зберігаються як масив об’єктів, у кожному з елементів подібний шаблон:
    - ScenarioName1

- id 0:
- data:
    - UserSpecifiedKey1: value,
    - UserSpecifiedKey2: value,
    - UserSpecifiedKey3: value,
    - UserSpecifiedKey4: value
- id 1:
- data:
    - UserSpecifiedKey5: value,
    - UserSpecifiedKey6: value,
    - UserSpecifiedKey7: value,

У цьому випадку один JSON-об’єкт відповідає одному зібраному сценарію, де у полі data зберігаються усі зібрані дані, а у полі id номер зібраного елемента.

Висновки до розділу 1

У підсумку, розділ 1 деталізує суть веб-парсингу, його практичне значення та основні труднощі (різноманітність HTML-структур, динамічний контент, нестабільність шаблонів). Наведено огляд готових інструментів для швидкого запуску проєктів збору даних без кодування. Описані вхідні дані (сценарії з правилами, теги й перелік URL) та формат вихідних файлів (CSV і JSON із визначеною структурою), що встановлює чіткі умови взаємодії користувача й системи. Завдяки такому підходу проект отримає гнучку архітектуру, зрозумілі контракти та готову основу для подальшої реалізації.

РОЗДІЛ 2
РОЗРОБКА АЛГОРИТМІВ ПАРСИНГУ

Парсинг веб-сторінок (англ. web scraping) — це процес автоматизованого отримання й обробки структурованих даних із HTML-документів, призначений для витягнення певних елементів із веб-сторінок. З огляду на різноманітність будови сучасних сайтів, змішані підходи до генерації контенту (статичний HTML, динамічний контент через JavaScript/AJAX) та різноманітні формати розмітки, були розроблені кілька основних алгоритмічних концепцій парсингу. У цьому підрозділі розглянемо класифікацію й особливості найпоширеніших підходів, аналізуючи їх переваги, недоліки та області застосування.

1. Розбір HTML за допомогою регулярних виразів

Суть алгоритму:

Витягування інформації з HTML-документів, застосовуючи регулярні вирази (Regex). Підходи такого типу побудовані на пошуку шаблонів у сирому тексті HTML. Наприклад, щоб знайти всі рядки, які містять ціни у форматі “$123.45”, використовують патерн типу \$[0-9]+\.[0-9]{2}.

Переваги:

- Простота реалізації для елементарних випадків: якщо відомо, що потрібна інформація завжди записана в жорстко визначеному форматі (наприклад, сума лише з цифр і крапкою).
- Висока швидкість обробки (регулярні вирази вбудовані у більшість мов програмування, виконуються без посередників).

Недоліки:

- Нестійкість до змін структури HTML: навіть мінімальні зміни в розмітці (додавання нового атрибуту або перенос тега в іншу позицію) можуть порушити відповідність шаблону. [15], [16]
- Застосування регулярних виразів для складного HTML — погана практика: HTML не є регулярною мовою, тому аналіз за допомогою Regex

ускладнюється (наприклад, глибокі вкладення тегів) і легко призводить до помилок.

- Не відрізняє локальної структури DOM: регулярний вираз не бачить ієрархію елементів (родительські/дочірні зв’язки), тому важко отримати, наприклад, лише ті заголовки, які знаходяться всередині певного тегу.

Область застосування:

- Швидке “одноразове” витягнення елементарних патернів (конкретна цифра, регулярний шаблон без вкладень).
- Скрипти “для себе”, коли розмітка гарантовано не змінюється.

2. DOM-парсинг (Document Object Model)

Суть алгоритму:

DOM-парсинг передбачає побудову внутрішнього представлення HTML-документу у вигляді дерева, де кожна гілка відповідає тегу, а вузли містять інформацію про атрибути та текстові фрагменти. Після побудови DOM-дерева алгоритм переміщується у потрібні вузли за допомогою запитів XPath або CSS-селекторів. [17], [18]

Основні бібліотеки-представники:

- HTML Agility Pack (HAP) для .NET, який перетворює сирий HTML у HtmlDocument з підтримкою XPath.
- BeautifulSoup (Python) — створює дерево на базі lxml або html5lib.
- Jsoup (Java) — дозволяє завантажити HTML, побудувати дерево DOM, а потім вибрати елементи за CSS-селекторами.
- AngleSharp (C#) — потужний HTML/CSS-парсер, що підтримує DOM Level 2/3, CSSOM і роботу з атрибутами.

Переваги:

- Надійність: незалежно від невеликих змін у HTML, XPath/CSS запит знайде елемент, якщо тег і атрибути залишилися незмінними. [19]
- Чітка ієрархія: можна звернутися до елементів у контексті батьківських/дочірніх вузлів.

- Гнучкість запитів: підтримка складних виразів XPath (фільтрація за атрибутами, позиція у списку, порівняння даних).
- Підтримка неправильного чи неповного HTML (DOM-бібліотеки можуть автоматично “ремонтувати” некоректно закриті теги).

Недоліки:

- Витрати пам’яті: повне дерево DOM може займати значний обсяг оперативної пам’яті, особливо для великих сторінок з сотнями тисяч вузлів.
- Повільніша обробка порівняно з Regex, особливо при великій кількості запитів до вузлів.
- Потрібно забезпечити коректну контекстно-залежну роботу: іноді структури HTML бувають доволі хитромудрими, й XPath треба складати дуже точно, щоб не отримати порожні результати.

3. Використання headless-браузерів (Selenium, Puppeteer)

Суть алгоритму:

Деякі веб-сторінки активно генерують контент через JavaScript/AJAX [20], тому просте завантаження HTML не містить кінцевих даних (наприклад, списку товарів, який підтягується через REST API після завантаження сторінки). У таких випадках застосовують headless-браузери, здатні виконувати JavaScript, відтворюючи роботу “справжнього” браузера, але без графічного інтерфейсу.

Найпоширеніші інструменти:

- Selenium WebDriver (C#, Java, Python, JavaScript) — може автоматизувати браузери типу Chrome, Firefox, Edge у headless-режимі.
- Puppeteer (JavaScript/Node.js) — керування Chromium у headless-режимі.
- Playwright (C#, Node.js, Python) — новіший інструмент із подібними можливостями.

Переваги:

- Можливість обробляти сторінки з динамічним контентом (SPA — single-page applications).

- Забезпечення максимально наближеної імітації дій користувача (натискання кнопок, введення тексту, виконання форм).
- Підтримка роботи з елементами, які генеруються JavaScript (наприклад, вкладені списки, карти, динамічні таблиці).

Недоліки:

- Високі витрати ресурсів: запуск headless-браузера потребує значної кількості оперативної пам’яті та CPU.
- Повільніше в порівнянні зі “статичними” DOM-парсерами: очікування рендерингу JS обов’язково збільшує час обробки.
- Додаткові залежності (встановлений браузер, драйвер).
- Складність в інтеграції в середовищі .NET без додаткових налаштувань (наприклад, необхідність розгортання відповідних bin-файлів Selenium WebDriver).

4. Фреймворки та робочі середовища для парсингу

Поряд із “чистими” алгоритмічними підходами існують комплексні фреймворки, що забезпечують одночасно завантаження, чергування URL, розбір HTML і збереження даних у різні типи сховищ.

Scrapy (Python):

- Підтримує відкладення запитів, чергу активних та відкладених URL, кешування відповідей.
- Має вбудоване дзеркало обробки помилок, обмеження швидкості (throttling), обхід правил.
- Експортує результати у CSV, JSON, XML, бази даних SQL (SQLite, PostgreSQL).
- Дає змогу легко масштабуватися за допомогою інструментів типу Scrapyd.

Goutte (PHP):

- Використовує бібліотеку Symfony BrowserKit + Guzzle для HTTP-запитів і DomCrawler для розбору.

- Добре підходить для простих сценаріїв у веб-розробці на PHP.

Jsoup (Java):

- Працює лише зі статичним HTML (без рендерингу JS).
- Підтримує CSS-селектори, що робить створення запитів простим

AngleSharp (C#/.NET):

- Підтримує JavaScript-плагіни (через AngleSharp.Scripting), що дозволяє деякою мірою емулювати JS-механізми. [21]
- Вбудований CSS-парсер — наприклад, можна застосовувати document.QuerySelectorAll("a[href]").

Підсумковий висновок щодо фреймворків:

Незважаючи на велику різноманітність готових інструментів, для проєкту було обрано HTML Agility Pack у поєднанні з ASP.NET MVC [1] і власними алгоритмами ADO .NET. Причини:

- Повна інтеграція з .NET, відсутність потреби підключати стороннє Python/Java середовище.
- Легкість налаштування бібліотеки HAP через NuGet.
- Можливість гнучкого формування XPath-виразів, що зберігаються в Access, без потреби компілювати додатковий код.
- Мінімальні залежності (немає необхідності розгортати сторонні служби).
- Можливість по–справжньому контролювати процес паралельного завантаження через асинхронні методи .NET (async/await) та параметри HttpClient.

Нижче приведено порівняльну таблицю основних методів парсингу (Табл. 2.1).

Таблиця 2.1

Порівняльна таблиця основних методів парсингу HTML-сторінок

Підхід	Плюси	Мінуси	Рекомендована сфера застосування
Регулярні вирази	Швидкість. Простота для примітивних випадків.	Нестійкість до змін розмітки, немає підтримки вкладених структур.	Швидке витягнення простих патернів із гарантованим форматом.
DOM-парсинг	Гнучкість, ієрархічна навігація, стійкість до мінімальних змін структури.	Витрати пам'яті, повільніший за Regex.	Складніший парсинг із вкладеними тегами й нестабільною розміткою.
Headless-браузери	Підтримка JS/AJAX, імітація поведінки користувача.	Велика витрата ресурсів, повільніший за інші методи, додаткові залежності.	Парсинг динамічних SPA, коли потрібне виконання скриптів.
Фреймворки	Комплексні рішення «із коробки»: черги, кешування, throttling, мультипотоковість.	Додаткова складність розгортання, залежності, крива навчання.	Проекти з великими вимогами до масштабування й відмовостійкості.

Як видно з таблиці 2.1, наведено порівняння чотирьох основних підходів до парсингу та вичленення даних із веб-сторінок: регулярні вирази, DOM-парсинг, headless-браузери та спеціалізовані фреймворки. Кожен із цих методів має свої сильні і слабкі сторони, а також області, в яких він найбільш доцільний.

Далі описано алгоритм, що реалізує послідовність дій для обробки веб-сторінок згідно з обраним користувачем сценарієм. На його основі будуть діяти сервіси: ParsingService для безпосереднього витягнення даних із HTML [22] за допомогою HTML Agility Pack і ParsingScenarioRepository для

зчитування/збереження сценаріїв та тегів у Microsoft Access через ADO .NET.

Алгоритм складається з таких етапів:

- Зчитування та збереження сценарію
- Формування XPath-виразу контейнера
- Парсинг окремого URL за допомогою HTML Agility Pack
- Обробка кожного тегу (ParsingTag) у межах контейнера
- Постобробка витягнутих значень і формування результату (ParsingResult)
- Оновлення станів і повернення колекції результатів

Нижче на Рис. 2.1. наведено блок-схему алгоритму.

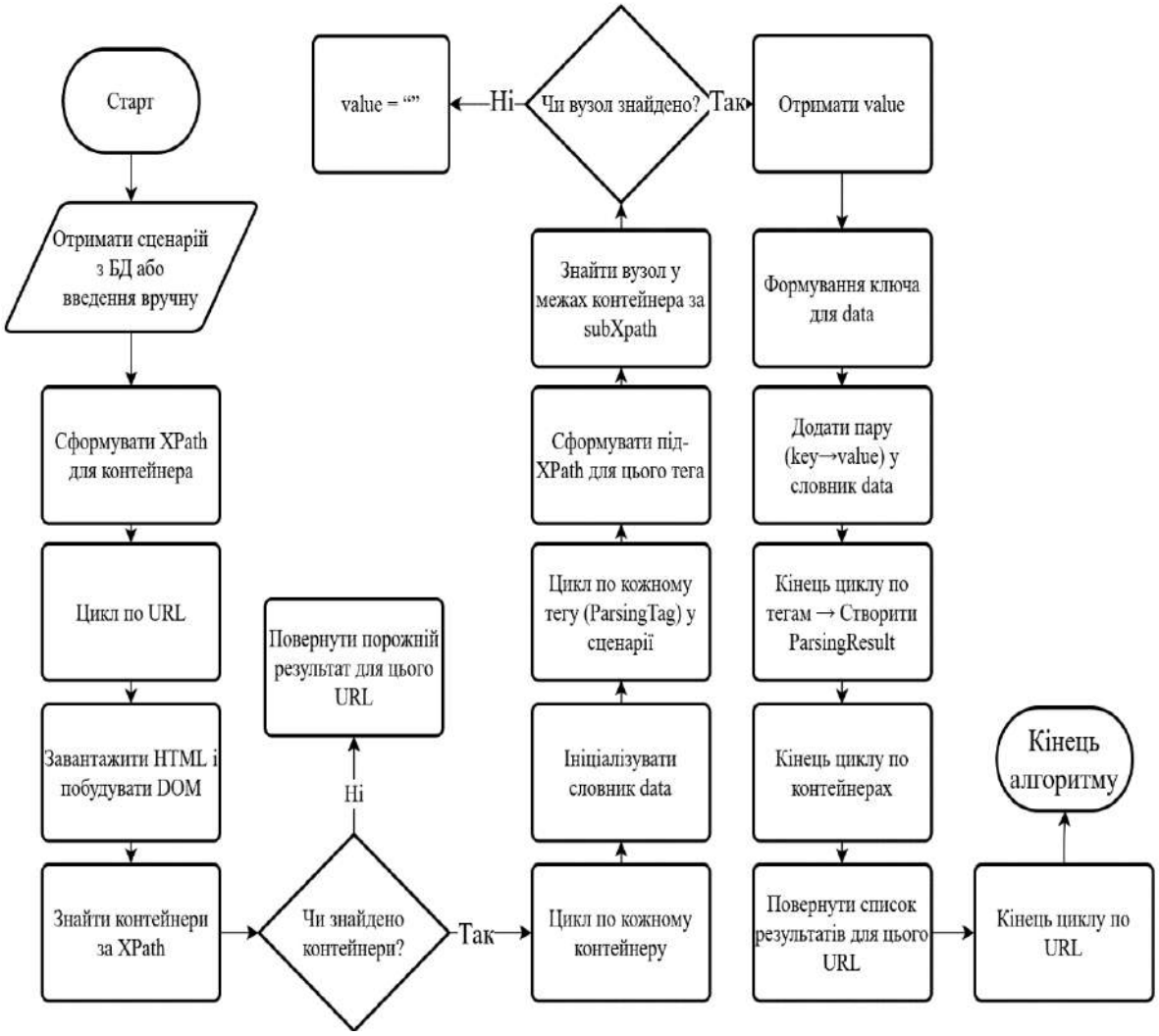


Рис. 2.1. Блок – схема алгоритму для обробки веб-сторінок за сценаріями та тегами

Цей алгоритм забезпечує чітку послідовність дій, чіткі умови обробки відсутніх елементів і мінімальний вплив на загальну роботу, коли якийсь окремий тег чи контейнер несподівано не знайдено.

Висновки до розділу 2

У цьому розділі детально порівняно основні підходи до парсингу веб-сторінок — від простих регулярних виразів до DOM-парсерів і headless-браузерів — із урахуванням їхніх переваг, недоліків та практичних сфер застосування. Регулярні вирази показують себе як швидке та просте рішення для елементарних задач із чітко визначеним форматом даних, проте вразливі до змін структури HTML і не здатні працювати зі вкладеними елементами. DOM-парсинг із використанням XPath або CSS-селекторами забезпечує стабільність у разі незначних змін розмітки, чітку ієрархічну навігацію й обробку помилок, хоча й потребує більших ресурсів пам’яті. Headless-браузери, наприклад Selenium або Puppeteer, дозволяють коректно обробляти динамічний JavaScript/AJAX-контент [23], імітуючи дії користувача, але вимагають значних витрат CPU й оперативної пам’яті, а також додаткових залежностей. Комплексні фреймворки (Scrapy, Goutte, Jsoup) пропонують «з коробки» механізми чергування запитів, кешування й масштабування, однак часто надмірні для невеликих .NET-проєктів.

Виходячи з орієнтації на ASP.NET MVC [2], інтеграцію з Microsoft Access через ADO .NET та прагнення звести до мінімуму зовнішні залежності, було обрано HTML Agility Pack. У підсумку, цей інструмент дозволяє зберегти максимальну гнучкість при формуванні XPath-виразів. Зберігати дані у базі Access і забезпечити стабільну обробку веб-сторінок навіть за незначних змін у розмітці. Завдяки цьому адміністратор чи користувач може змінювати сценарії й XPath-вирази без потреби перекомпільовувати код, а сервіс залишається стійким у разі відсутності очікуваних елементів у DOM. Такий підхід поєднує в собі ідеї порівняння різних методів парсингу з практичною реалізацією цих методів.

РОЗДІЛ 3
РОЗРОБКА ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Структура та опис таблиць бази даних

Опис таблиці ParsingScenarios:

Таблиця ParsingScenarios зберігає основні налаштування для кожного сценарію парсингу. Кожен запис цієї таблиці відповідає одному «сценарію» — тобто набору параметрів, за якими система відбиратиме блоки-контейнери з веб-сторінок.

Id – унікальний ідентифікатор сценарію. Генерується автоматично у форматі GUID. Цей ключ використовується в інших таблицях (ScenarioUrls та ParsingTags) для зв’язування даних із конкретним сценарієм.

Name – зрозуміла людині назва сценарію. На англійській мові, без пробілів. Наприклад, “ExampleScenario”, “UnivNews ” тощо. Саме з цього поля користувач бачить ім’я сценарію в інтерфейсі.

ContainerTag – назва HTML-тега, який система вважатиме «контейнером» для подальшого пошуку тегів-правил. Наприклад, якщо на сторінці всі потрібні елементи укладені всередині тегу , значення цього поля має бути “div”.

ContainerClass – (за потреби) CSS-клас контейнера. Якщо користувач хоче точніше обмежити область пошуку, можна вказати, скажімо, “product-item”. Тоді система шукатиме лише ті елементи, що мають тег ContainerTag і містять атрибут class, який включає “product-item”. Якщо це поле порожнє, система ігнорує фільтр за класом і брати до уваги лише тег із ContainerTag.

Опис таблиці ScenarioUrls:

Таблиця ScenarioUrls містить перелік веб-адрес (URL), які потрібно обробити в межах конкретного сценарію. Кожний запис відображає зв’язок “один сценарій → багато URL”.

Id – Лічильник, який містить у собі унікальний числовий номер URL.

ScenarioId – зовнішній ключ на поле ParsingScenarios.Id. Завдяки цьому полю можна визначити, до якого саме сценарію належить кожна URL-адреса.

Url – безпосередньо рядок із веб-адресою. Наприклад, “https://example.com/page1” або “https://shop.ua/item/12345”.

При збереженні сценарію кожне URL, введене користувачем у веб-формі, записується в цю таблицю. Під час виконання алгоритму парсингу система перебирає усі значення Url, що мають відповідний ScenarioId, і робить HTTP-запит до кожної адреси, щоб отримати HTML-документ для подальшого аналізу.

Опис таблиці ParsingTags:

Таблиця ParsingTags визначає набір «правил» (тегів) для кожного сценарію. Кожний запис описує один тег-умову, за якою система витягуватиме інформацію з усіх знайдених контейнерів у HTML.

Id – унікальний ідентифікатор тегу GUID. Використовується лише для внутрішнього зв’язку, але може знадобитися, якщо треба редагувати чи видаляти конкретний тег.

ScenarioId – зовнішній ключ на випадаюче ParsingScenarios.Id. Вказує, до якого сценарію належить даний тег.

HtmlTag – назва HTML-тега, який потрібно знайти у межах контейнера. Наприклад, “h1”, “span”, “img”, “a” тощо.

Class – (опційно) CSS-клас HTML-тега. Якщо поле непорожнє, при формуванні під-XPath додається умова contains(@class, 'Class'). Це дозволяє точніше відбирати елементи, що мають заданий клас.

Text – (опційно) частковий текст, що повинен входити до вмісту тегу. Якщо це поле непорожнє, система додає умову contains(text(), 'Text') до під-XPath.

ParsingType – визначає, яке значення ми хочемо отримати з знайденого вузла. Два можливих варіанти:

Text – система бере з елемента InnerText.Trim().

Attribute – система підставляє у поле AttributeName і отримує GetAttributeValue(AttributeName, "").

AttributeName – (якщо ParsingType = Attribute) назва атрибута, з якого буде зчитано значення (наприклад, src, href, alt). Якщо ж ParsingType = Text, це поле ігнорується.

Key – умовна назва поля, під якою результат зберігатиметься у словнику ParsingResult.Data[key]. Наприклад, якщо тег відповідає заголовку товару, можна написати “Title”; якщо тег – ціна, Key = “Price”. Якщо Key порожнє, система спочатку спробує використати значення HtmlTag як ім’я поля; а якщо й воно пусте, згенерує випадковий GUID.

Взаємозв’язки між таблицями:

ParsingScenarios.Id → ScenarioUrls.ScenarioId

– Один сценарій може мати кілька URL, тому в таблиці ScenarioUrls для кожного запису вказується той самий ScenarioId.

ParsingScenarios.Id → ParsingTags.ScenarioId

– Кожен сценарій може мати багато тегів-умов, які описують, що потрібно витягти. Тому в таблиці ParsingTags кожний запис також містить ScenarioId.

Нижче наведено описи таблиць бази даних із зазначенням їхніх назв, типів даних та призначення (Табл. 3.1- Табл. 3.3).

Таблиця 3.1

Таблиця ParsingScenarios

Поле	Тип даних	Опис
Id	Короткий текст	Унікальний ідентифікатор сценарію (GUID)
Name	Короткий текст	Назва сценарію (наприклад: “UnivNews”)
ContainerTag	Короткий текст	HTML-тег-контейнер (наприклад: div, a, p)
ContainerClass	Короткий текст	CSS-клас контейнера (наприклад: product-item; якщо не використовується – порожній)

Як видно з таблиці 3.1, наведено перелік полів, які описують конфігурацію сценарію для веб-парсера. Кожне поле має тип даних «Короткий текст» і призначене для зберігання певного атрибута сценарію:

Таблиця 3.2

Таблиця ScenarioUrls

Поле	Тип даних	Опис
Id	Лічильник	Унікальний числовий ідентифікатор
ScenarioId	Короткий текст	Зовнішній ключ → ParsingScenarios.Id; вказує, до якого сценарію належить цей URL
Url	Довгий текст	Один URL-адрес (рядок), який потрібно обробити

Як видно з таблиці 3.2, наведено перелік полів для збереження URL-адрес, які належать до певного сценарію парсингу. Кожне поле відповідає за зберігання унікального ідентифікатора запису, зв’язку з конкретним сценарієм і самої адреси сторінки.

Таблиця 3.3

Таблиця ParsingTags

Поле	Тип даних	Опис
Id	Короткий текст	Унікальний ідентифікатор тега (GUID)
ScenarioId	Короткий текст	Зовнішній ключ → ParsingScenarios.Id; вказує, до якого сценарію належить цей тег
HtmlTag	Короткий текст	Назва HTML-тега для пошуку всередині контейнера (наприклад: span, h1, img)
Class	Короткий текст	CSS-клас для уточнення вибірки (якщо порожній – відбираються всі відповідні теги без урахування класу)
Text	Короткий текст	Частковий текст, що має міститися всередині знайденого вузла (якщо не потрібен – порожній)
ParsingType	Короткий текст	Тип парсингу: Text (витягти InnerText) або Attribute (витягти значення атрибута)
AttributeName	Короткий текст	Назва атрибута (наприклад: src, href), якщо ParsingType = Attribute (інакше – порожній)
Key	Короткий текст	Умовна назва поля, під яким зберігатиметься отримане значення у результаті парсингу

Як видно з таблиці 3.3, наведено перелік полів для конфігурації окремих тегів, які потрібно шукати та обробляти всередині контейнера в межах певного сценарію парсингу. Кожне поле відповідає за налаштування умов обрання тегу, способу вилучення даних і зберігання результату.

3.2. Моделі доменних об’єктів та відповідність таблицям.

У цій частині наведено опис класів-доменних об’єктів у проекті та показано.

Клас ParsingScenario

Призначення:

Модель описує один сценарій парсингу – набір параметрів (контейнер, URL, теги), які система використовує для вибірки елементів із веб-сторінок. Нижче, у таблиці 3.4 наведені атрибути класу ParsingScenario.

Таблиця 3.4

Таблиця атрибутів класу ParsingScenario

Атрибут	Тип	Опис
Id	string	Унікальний ідентифікатор сценарію (GUID)
Name	string	Назва сценарію
ContainerTag	string	HTML-тег, який обмежує область пошуку (наприклад, div)
ContainerClass	string	CSS-клас контейнера (якщо потрібне звуження області)
Urls	List<string>	Перелік URL для обробки
Tags	List<ParsingTag>	Колекція правил (тегів) для вибірки даних
Results	List<ParsingResult>	Список результатів парсингу для кожного знайденого контейнера

Як видно з таблиці 3.4, наведено перелік атрибутів, які описують головну модель сценарію парсингу у вигляді об’єкта з відповідними властивостями. Кожне поле відповідає за окремий аспект конфігурації та результатів виконання:

- Поля Id, Name, ContainerTag та ContainerClass зчитуються та зберігаються безпосередньо в таблиці ParsingScenarios.
- Властивість Urls відображає всі записи з таблиці ScenarioUrls, де ScenarioId = ParsingScenario.Id.
- Властивість Tags накопичує всі записи з таблиці ParsingTags, де ScenarioId = ParsingScenario.Id.
- Results формується динамічно під час виконання парсингу й існує тільки в пам’яті: жодне поле цієї властивості не зберігається в Access.

Клас ParsingTag

Призначення:

Модель описує одне правило (один тег) для сценарію — тобто, який HTML-тег потрібно шукати, за яких умов і як витягувати значення.

Нижче, у таблиці 3.5 наведені властивості класу ParsingTag.

Таблиця 3.5

Таблиця атрибутів класу ParsingTag

Атрибут	Тип	Опис
Id	string	Унікальний ідентифікатор тегу (GUID)
ScenarioId	string	Ідентифікатор сценарію, до якого належить тег
HtmlTag	string	Назва HTML-тега для пошуку (наприклад, span, h1, img)
Class	string	CSS-клас для уточнення вибірки (якщо порожній, умову ігнорують)
Text	string	Частковий текст, який повинен входити в потрібний вузол (якщо порожній, умову ігнорують)
ParsingType	ParsingType (enum)	Text — витягнути InnerText, Attribute — взяти значення атрибута
AttributeName	string	Назва атрибута для витягання (наприклад, src або href), якщо ParsingType = Attribute
Key	string	Умовна назва поля, за яким зберігатиметься результат (наприклад, “Title”, “Price”)

Як видно з таблиці 3.5, наведено перелік атрибутів класу ParsingTag, який задає правила пошуку та обробки окремих HTML-елементів у межах певного сценарію парсингу:

- Поля Id та ScenarioId зв’язують тег із конкретним сценарієм.
- HtmlTag, Class та Text потрібні для побудови під-XPath, за яким вибирається потрібний вузол у межах контейнера.
- ParsingType визначає, чи потрібно брати текст вузла (InnerText), або значення атрибута.
- AttributeName використовується лише коли ParsingType = Attribute; інакше ігнорується.
- Key задає, під яким іменем ця інформація потрапить у словник результату.

Клас ParsingResult

Призначення:

Модель представляє один результат парсингу для певного контейнера на сторінці. Містить ідентифікатор контейнера (його індекс у списку знайдених блоків) та словник ключ → значення, де зібрані всі пари, що відповідають правилам у ParsingTag.

Нижче, у таблиці 3.6 наведені властивості класу ParsingResult.

Таблиця 3.6

Таблиця атрибутів класу ParsingResult

Атрибут	Тип	Опис
Id	int	Порядковий номер контейнера (0, 1, 2, …), який відповідає знайденому елементу-контейнеру
Data	Dictionary<string, string>	Словник, де ключ = ParsingTag.Key, а значення = витягнуте з HTML (текст або атрибут)

Як видно з таблиці 3.6, наведено перелік атрибутів класу ParsingResult, який служить для зберігання результатів парсингу одного знайденого контейнера:

- Id використовується лише в рамках однієї сесії парсингу, вказує, в якому порядку зустрічаються контейнери. Це допомагає ідентифікувати кожну окрему групу даних, що витяглися з одного контейнера.
- Data заповнюється під час обходу всіх правил (ParsingTag) і містить набір пар (ключ → значення).
- Жодне поле ParsingResult не відображається у базі Access, оскільки результати парсингу не зберігаються там. Після завершення парсингу список ParsingResult лише повертається клієнту у форматі JSON.

3.3. Зв’язки моделей і таблиць у проекті

1. ParsingScenario ↔ ParsingScenarios
- Ключове поле ParsingScenario.Id відповідає ParsingScenarios.Id.
- На екрані користувач обирає або створює новий сценарій, заповнює Name, ContainerTag, ContainerClass. Після збереження ці значення потрапляють у відповідні колонки таблиці ParsingScenarios.

2. ParsingScenario.Urls ↔ ScenarioUrls
   - Для кожного рядка в ParsingScenario.Urls створюється окремий запис у ScenarioUrls з полями:
     - ScenarioUrls.ScenarioId = ParsingScenario.Id
     - ScenarioUrls.Url = <значення URL>
3. ParsingScenario.Tags ↔ ParsingTags
   - Кожен об’єкт ParsingTag у колекції ParsingScenario.Tags відповідає одному рядку в таблиці ParsingTags, де:
     - ParsingTags.Id = ParsingTag.Id
     - ParsingTags.ScenarioId = ParsingScenario.Id
     - ParsingTags.HtmlTag = ParsingTag.HtmlTag
     - ParsingTags.Class = ParsingTag.Class
     - ParsingTags.Text = ParsingTag.Text

- ParsingTags.ParsingType = ParsingTag.ParsingType.ToString()
- ParsingTags.AttributeName = ParsingTag.AttributeName
- ParsingTags.Key = ParsingTag.Key

4. ParsingResult

- Модель не зберігається в жодній таблиці. Клас ParsingResult існує тільки під час виконання методу ParseUrl і потім передається клієнту JSON-об’єктом.

Таким чином, кожна модель-доменний об’єкт у коді чітко відображає відповідну структуру в базі даних (за винятком ParsingResult, який призначений лише для роботи в оперативній пам’яті під час парсингу).

Висновки до розділу 3

У цьому розділі описано, як організовано інформаційне забезпечення системи парсингу. Таблиця ParsingScenarios зберігає налаштування кожного сценарію: унікальний GUID (Id), назву (Name) та параметри контейнера (ContainerTag, ContainerClass). Для кожного сценарію таблиця ScenarioUrls містить список URL (із зовнішнім ключем ScenarioId), які система обробляє, роблячи HTTP-запити та отримуючи HTML-документи.

Таблиця ParsingTags визначає правила витягування даних із знайдених контейнерів. Кожне правило має GUID (Id), посилання на ScenarioId, назву HTML-тега (HtmlTag), опційний CSS-клас (Class) чи фрагмент тексту (Text), а також способи отримання даних: через ParsingType = Text (InnerText) або ParsingType = Attribute (із вказівкою AttributeName). Поле Key задає ім’я для збереження значення в результаті.

У коді клас ParsingScenario відповідає таблиці ParsingScenarios: властивості Id, Name, ContainerTag, ContainerClass, а також колекції Urls (List<string>) і Tags (List<ParsingTag>), що завантажуються з таблиць ScenarioUrls і ParsingTags за ScenarioId. Клас ParsingTag містить поля, які дають змогу побудувати XPath-умови для витягування даних. ParsingResult існує лише

під час виконання парсингу: кожен екземпляр містить порядковий номер контейнера (Id) та словник Data із парами «Key → значення». Після завершення результати передаються клієнту в JSON, не зберігаючись у базі. Така структура забезпечує гнучке налаштування сценаріїв, чітке відображення даних у коді й високу продуктивність системи.

РОЗДІЛ 4
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Інтерфейс користувача додатку

При формулюванні вимог до програмного комплексу враховувалося, що рішення має бути:

- Простим у розгортанні: мінімізувати потребу в налаштуванні додаткових серверів чи складних середовищ. [13]
- Масштабованим та гнучким: можливість додавати нові сценарії парсингу без правки коду. [14]
- Надійним та продуктивним: стійко переносити певне навантаження (до кількох десятків URL за один сеанс) та швидко обробляти сторінки.
- Зручним у подальшому обслуговуванні: чітка архітектура з розділенням шарів, зручний інтерфейс для налаштування сценаріїв у БД.

На підставі цих критеріїв було ухвалено такі рішення:

Платформа ASP.NET MVC (на .NET 7.0) [3].

- Призначена для швидкої розробки веб-рішень із чітким поділом на Model, View, Controller.
- Забезпечує впорядкований роутинг, можливість використання Razor-шаблонів.

HTML Agility Pack

- .NET-пакет для роботи з HTML: здатний сприймати навіть «битий» HTML і перетворювати його на DOM-дерево.
- Підтримує XPath-вирази для вибірки вузлів, що спрощує реалізацію тих сценаріїв, які зберігаються в базі.

ADO .NET + Microsoft Access [11], [12]

- Дозволяє зберігати сценарії і теги парсингу у файл-базі (.accdb), без необхідності налаштування окремого сервера БД.

- Через OleDbConnection та OleDbCommand можна реалізувати швидкий доступ CRUD (Create, Read, Update, Delete) до таблиць: ParsingScenarios, ParsingTags, ScenarioUrls.
- Access має обмеження на обсяг (до 2 ГБ) і кількість одночасних підключень (~255), але для задач дипломного рівня (обробка десятків URL за сеанс) цього достатньо. [7]

C++/MFC + OpenGL для десктоп-візуалізатора [8]

- Використовувати MFC (Microsoft Foundation Classes) як інструментарій для створення Windows-додатків із графічним інтерфейсом.
- OpenGL обрано для побудови 2D-графіки: дозволяє швидко рендерити графічні таблиці на основі CSV-даних. [9]

Таким чином, поєднання ASP.NET MVC [4] (C#), HTML Agility Pack, ADO .NET + Access і десктопного компонента на C++/MFC з OpenGL відповідає поставленим вимогам простоти, гнучкості й продуктивності.

Головна сторінка веб-додатку побудована як Razor-вид, що відображає перелік сценаріїв у вигляді карток і містить необхідні модальні вікна для створення, редагування сценарію та тегів. [15]

На головній сторінці веб-додатку користувач бачить інтерфейс, який складається з таких логічних блоків:

1. Заголовок і панель основних дій

У верхній частині розміщено великий заголовок “Web Parser”. Під ним розташована панель із низкою кнопок:

- “Створити новий сценарій” — відкриває форму для введення назви нового сценарію, URL, тег-контейнера та його класу.
- “Запустити всі сценарії” — викликає виконання парсингу для всіх сценаріїв, що зараз завантажені в додатку.
- “Завантажити результати (JSON)” — експортує всі результати парсингу у форматі JSON.

- “Завантажити результати (CSV)” — експортує всі результати у форматі CSV.
- “Завантажити сценарії з БД” — робить запит до серверу, завантажує наявні у базі сценарії та відображає їх у модальному вікні, після чого користувач може їх завантажити, редагувати, або видалити.

2. Форма створення нового сценарію (модальне вікно)

- При натисканні “Створити новий сценарій” відкривається модальне вікно, у якому можна задати:
- Назву сценарію (латиницею, без пробілів).
- Список URL — набір полів, кожне з яких призначене для окремої адреси. Можна додавати або видаляти поля для URL, і всі введені адреси збираються у колекцію.
- Тег контейнера — одиничне текстове поле для введення назви HTML-тега, який обмежуватиме область пошуку.
- Клас контейнера — необов’язкове текстове поле, що уточнює пошук лише до тих тегів, які мають певний CSS-клас.
- По завершенню користувач натискає “Створити”, і сценарій додається в пам’ять, після чого модальне вікно закривається.

Вигляд модального вікна створення нового сценарію зображено на Рис. 4.1.

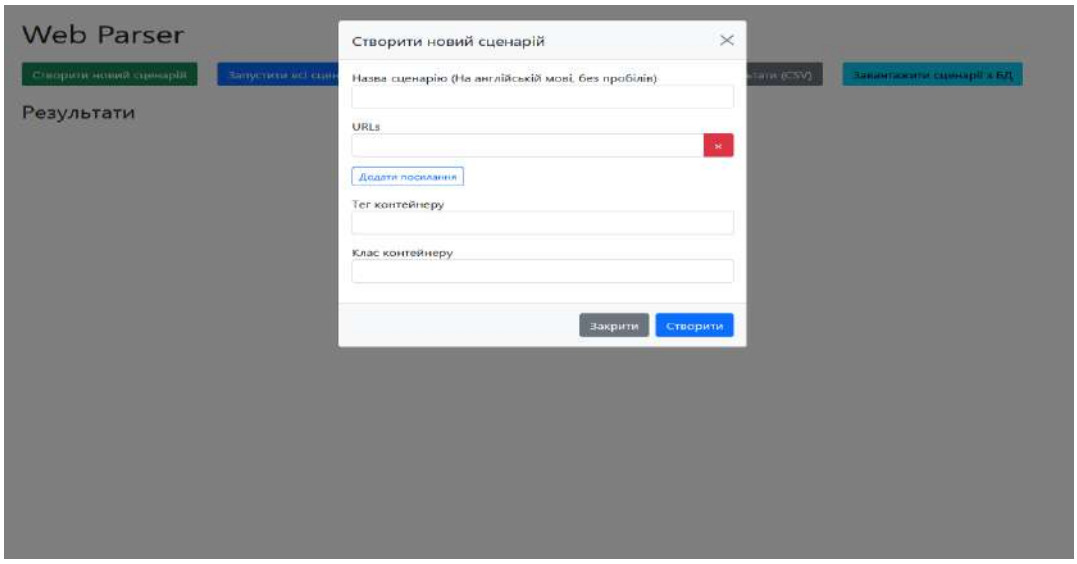


Рис. 4.1. Модальне вікно створення сценарію

3. Список завантажених/створених сценаріїв

Під панеллю дій розташований блок, в якому динамічно виводятся картки кожного сценарію (якщо список не порожній). Картка сценарію має таку структуру:

- Заголовок картки — назва сценарію.
- Кнопки дій у заголовку:
- “Редагувати сценарій” — відкриває модальне вікно з формою, заповненою поточними параметрами (назва, URL, тег-контейнер, класи та перелік тегів).
- “Видалити сценарій” — видаляє сценарій із пам’яті.
- “Запустити сценарій” — запускає тестовий парсинг обраного сценарію.
- “Додати тег” — відкриває форму для створення нового правила (ParsingTag) у межах цього сценарію.
- “Зберегти до БД” — записує сценарій у базу Access.

- Основний вміст картки:
- Відображається перша URL-адреса із списку (як ілюстрація того, які посилання додавав користувач), а також тег-контейнер і його CSS-клас.
- Нижче розташована область, де відображуються всі існуючі теги парсингу в рамках цього сценарію. Для кожного тегу показано:
- Його ключ (Key).
- HTML-тег, у якому шукатимуться дані.
- CSS-клас, за яким фільтруватимуть елемент (якщо вказано).
- Текстова умова (Text), яка може бути використана для додаткової фільтрації за частковим вмістом.
- Тип парсингу (Text або Attribute). Якщо тип – Attribute, також виводиться назва атрибута (AttributeName).
- Кнопки “Редагувати” та “Видалити” поряд із кожним тегом, що дозволяють змінювати параметри вже існуючого правила чи видаляти його.

На Рис. 4.2. зображено зовнішній вигляд сторінки із створеним сценарієм.

На Рис. 4.3. зображено зовнішній вигляд сторінки із сценарієм до якого додано пару тегів.

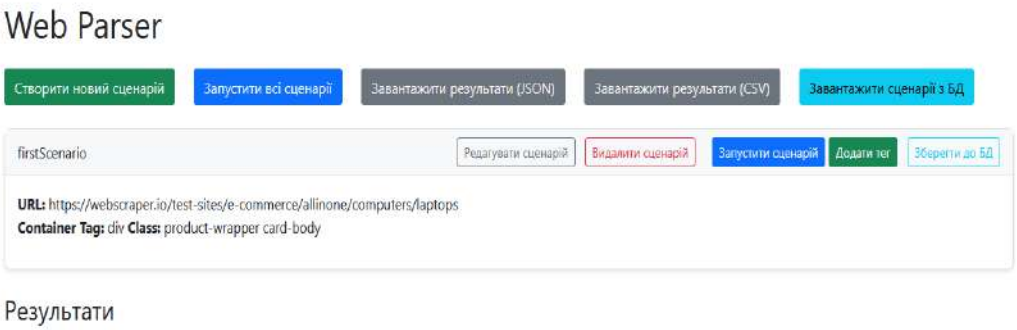


Рис. 4.2. Вигляд сторінки із створеним сценарієм

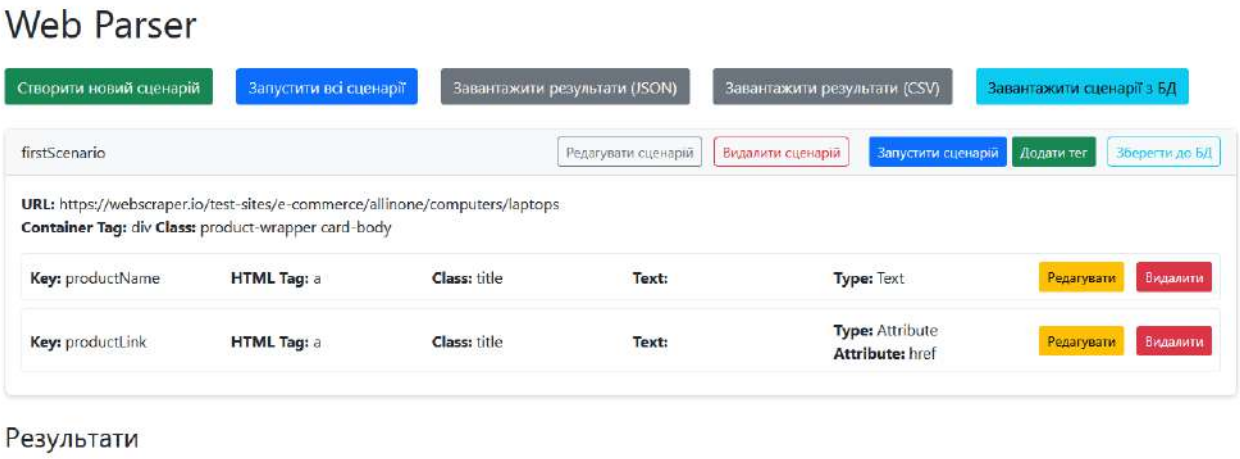


Рис. 4.3. Вигляд сторінки сценаріїв із доданими тегами

4. Результати парсингу

Під списком сценаріїв розташована секція “Результати”. Спочатку вона порожня. Коли користувач натискає “Запустити сценарій” або “Запустити всі сценарії”, у цю область динамічно виводяться дані у JSON форматі.

Приклад із отриманими результатами показано на Рис. 4.4.

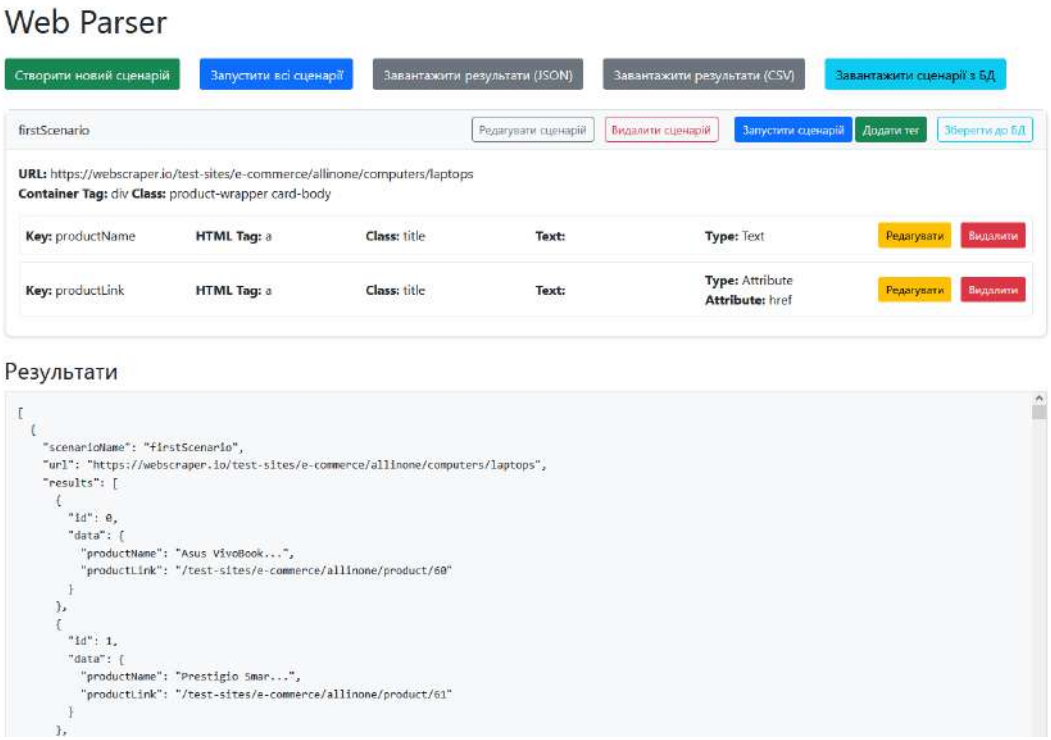


Рис. 4.4. Результати парсингу після запуску сценарію

5. Модальні вікна для тегів і редагування сценарію

У картці кожного сценарію є три додаткові модальні форми:

- “Додати тег” — дозволяє створити нове правило для парсингу в межах цього сценарію. Користувач вводить HTML-тег, опційний CSS-клас, опційний текст-фільтр, обирає тип парсингу (Text або Attribute) і, якщо Attribute, задає назву атрибута. Також вводить ключ (Key), під яким результат відображатиметься у вихідних даних.
- “Редагувати тег” — аналогічна форма, але вже з попередньо заповненими полями поточного тегу. Дозволяє змінити будь-яку з умов, після чого оновлені дані відображаються в картці.
- “Редагувати сценарій” — відкриває форму, де можна змінити назву сценарію, тег-контейнер, CSS-клас та список URL (додавати/видаляти поля). Поле для тегів у цій формі не відображається, оскільки теги редагуються окремо через “Редагувати тег”.

Вигляд модального вікна “Додати тег” зображено на Рис. 4.5.

Вигляд модального вікна “Редагувати тег” зображено на Рис. 4.6.

Вигляд модального вікна “Редагувати сценарій” зображено на Рис. 4.7.

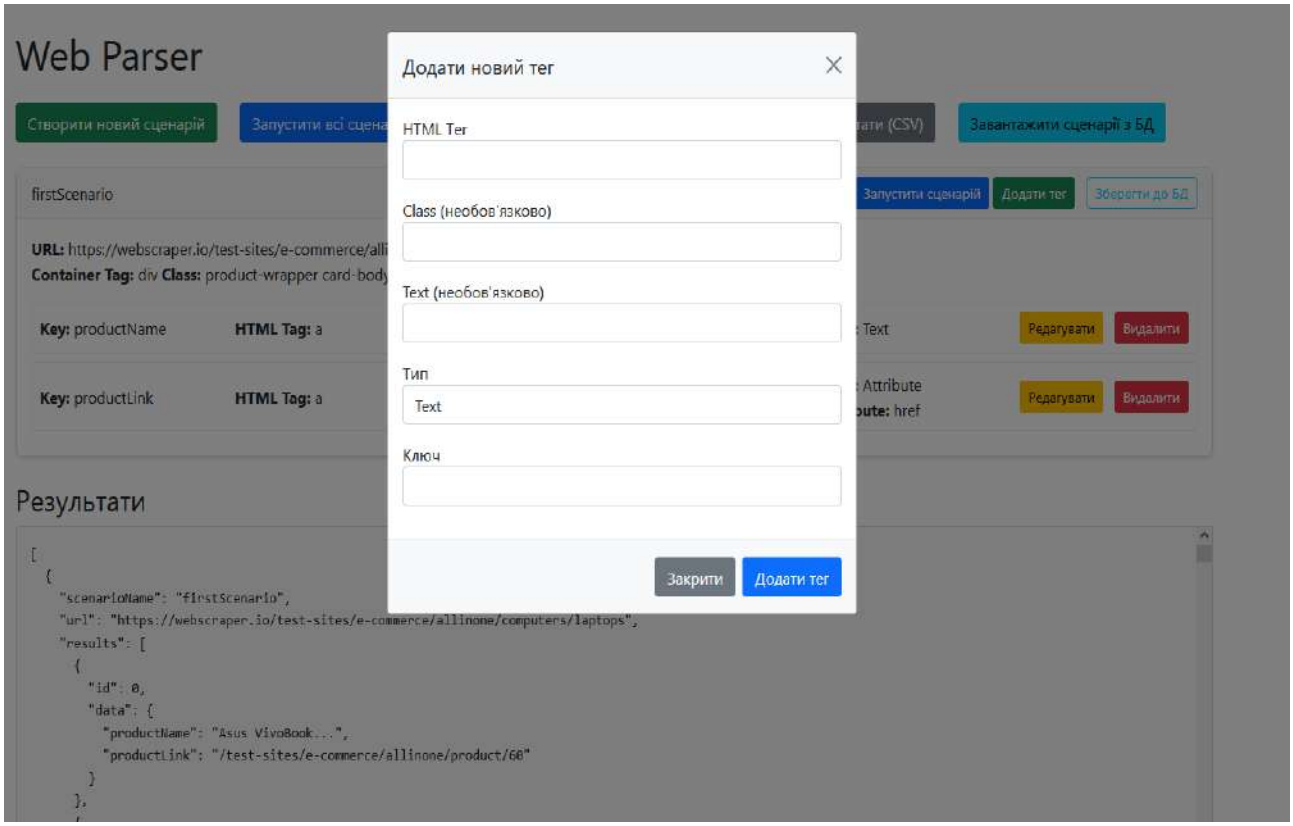


Рис. 4.5. Модальне вікно “Додати тег”

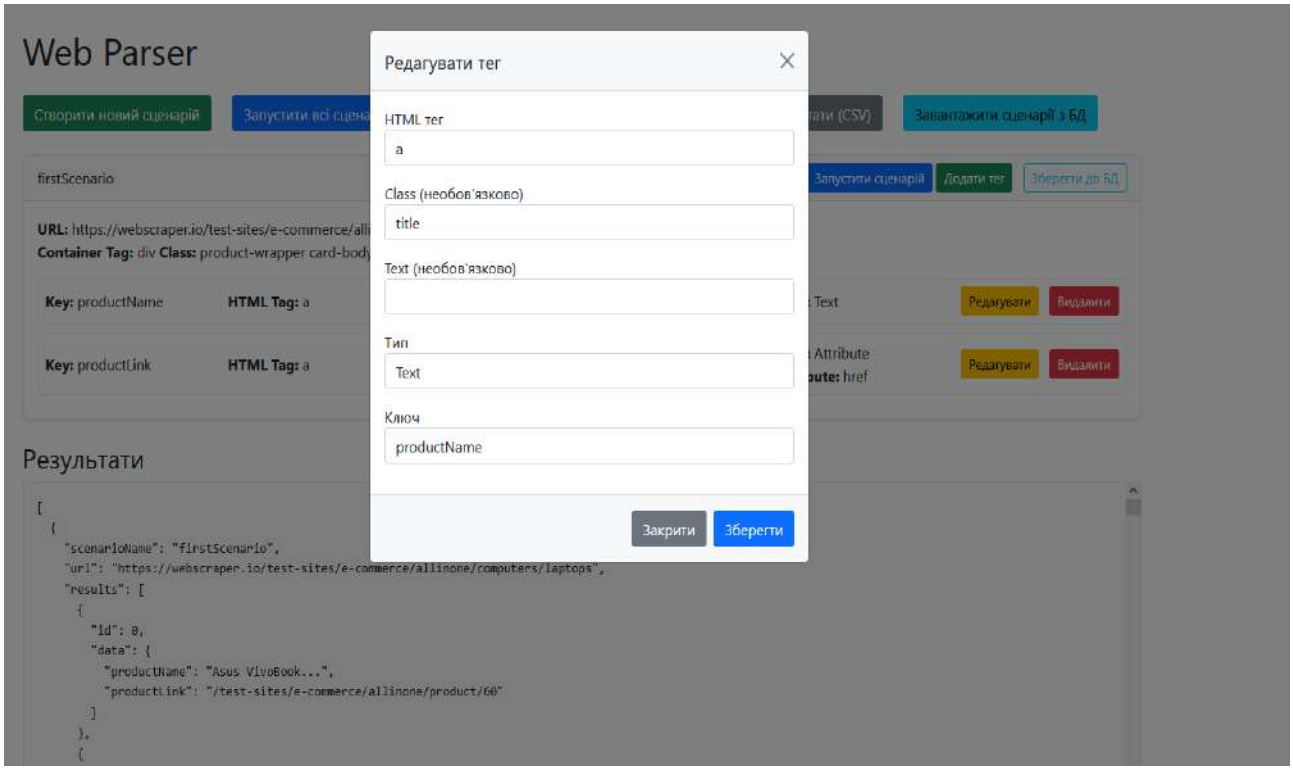


Рис. 4.6. Модальне вікно “Редагувати тег”

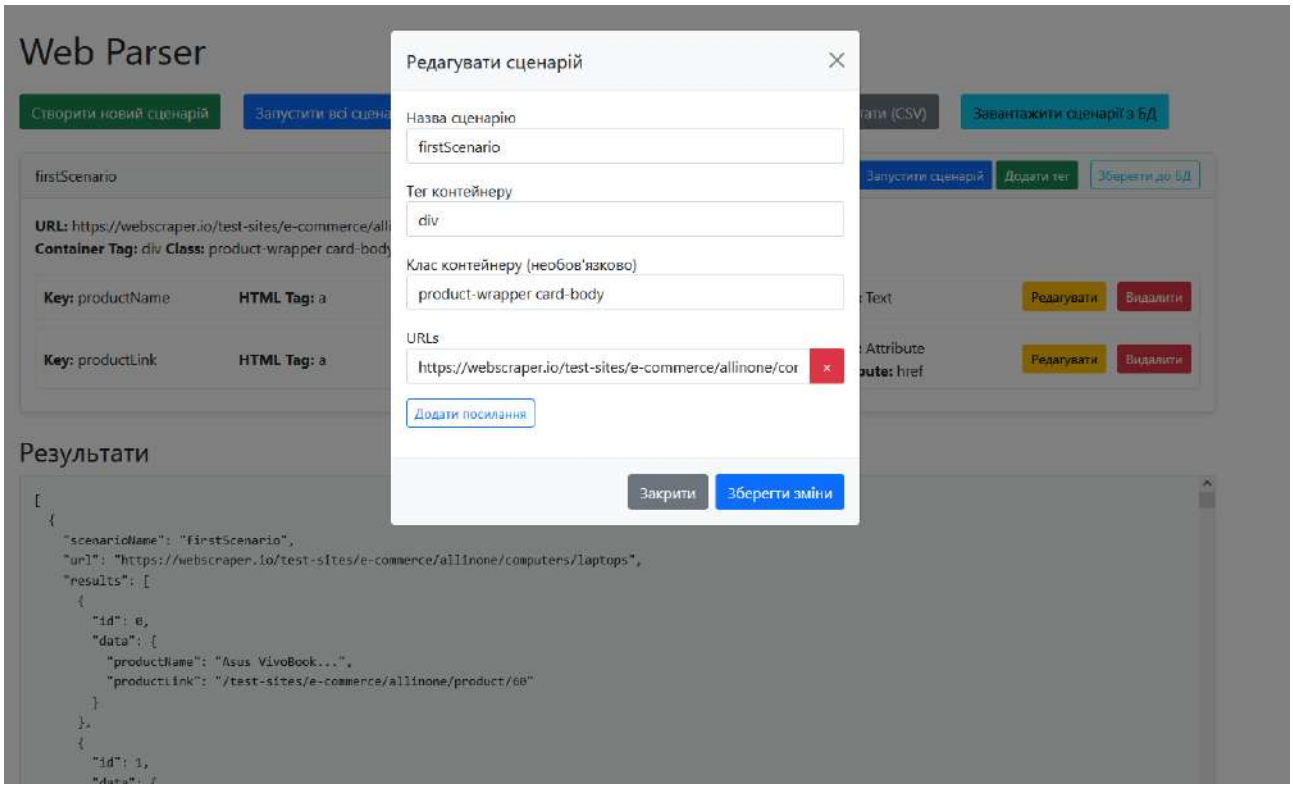


Рис. 4.7. Модальне вікно “Редагувати сценарій”

6. Механізм завантаження/збереження сценаріїв з/до бази даних.

- Механізм завантаження/збереження у базу

Кнопка “Зберегти до БД” на картці сценарію дозволяє відправити обраний об’єкт сценарію на сервер у метод SaveScenarioToDb, в якому репозиторій зберігає всі зміни у таблиці ParsingScenarios, ScenarioUrls і ParsingTags.

Результат натискання кнопки “Зберегти до БД” є “alert” повідомлення. На Рис. 4.8. можна побачити результат натискання на кнопку “Зберегти до БД”.

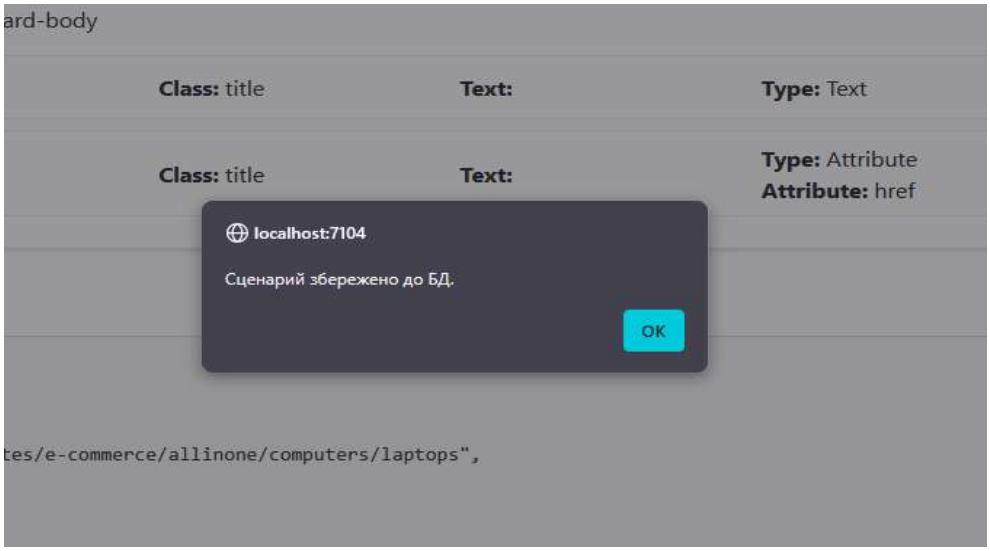


Рис. 4.8. Збереження сценарію до бази даних

Кнопка “Завантажити сценарії з БД” звертається до методу LoadAllScenariosFromDb, який повертає усі сценарії із бази даних, і потім ці сценарії відображаються користувачеві для вибору, редагування або видалення.

Результат натискання кнопки “Завантажити сценарії з БД” зображено на Рис. 4.9.

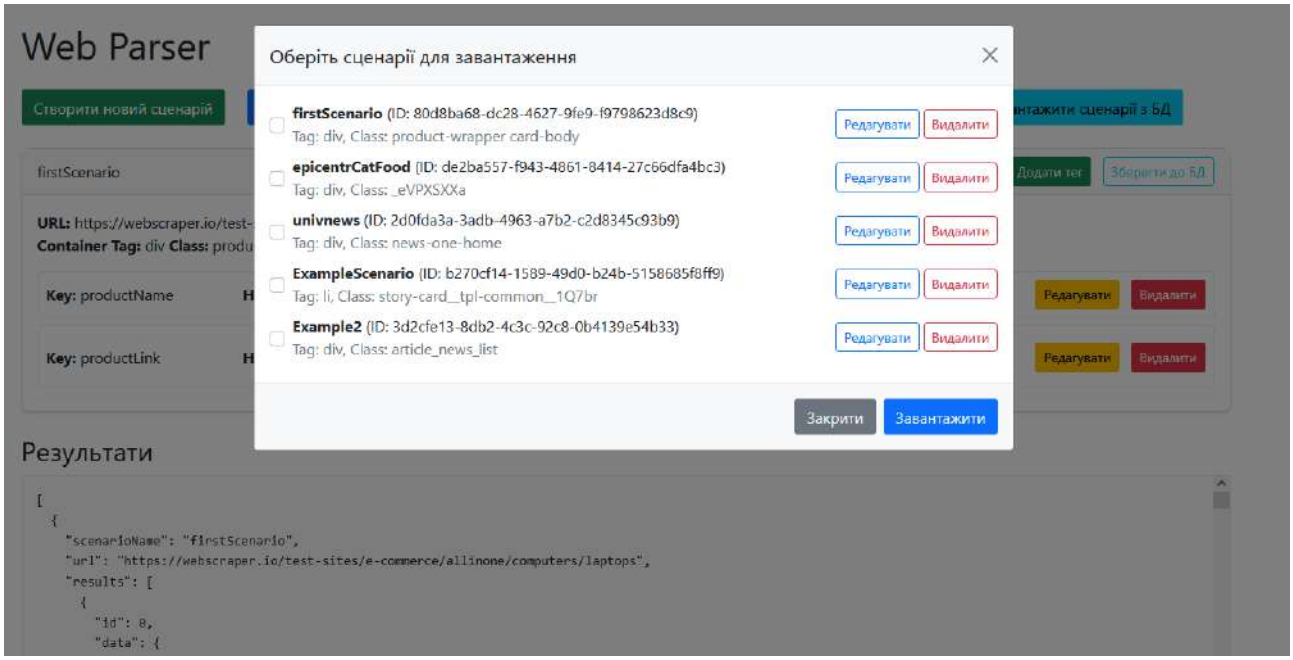


Рис. 4.9. Результат натискання кнопки “Завантажити сценарії з БД”

Вибір сценаріїв для завантаження здійснюється шляхом відмічання потрібних сценаріїв у чек-боксах. Редагування – шляхом натиску на кнопку “Редагувати”. Результатом натискання на кнопку “Редагувати” є модальне вікно, в якому користувач знову може змінити параметри сценарію. Вигляд модального вікна зображено на Рис. 4.10.

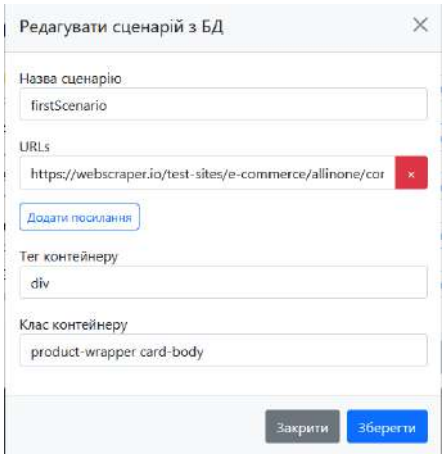


Рис. 4.10. Модальне вікно редагування сценарію

Видалення сценаріїв з бази данних здійснюється за допомогою натискання кнопки “Видалити”. При натисканні на цю кнопку, система додатково спитає користувача, чи дійсно він хоче видалити обраний сценарій, після підтвердження дії, сценарій видаляється з бази даних і сценарій зникає з модального вікна для обирання сценаріїв для завантаження.

4.2. Опис розроблених класів та функцій

У підрозділі представлено детальний опис основних класів і функцій, реалізованих у рамках WebParserApp. Архітектура додатку побудована за принципом поділу відповідальностей (Separation of Concerns), що забезпечує гнучкість і розширюваність функціоналу. Нижче наведено опис кожного компонента, їхніх властивостей та ролі в загальному процесі парсингу веб-сторінок і збереження/завантаження сценаріїв.

1. Моделі даних (простір імен WebParserApp.Models)

1.1.Клас ParsingScenario

- Призначення: Представляє собою «сценарій» парсингу, тобто набір налаштувань, необхідних для витягування інформації з веб-сторінок. Сценарій містить перелік URL-адрес, у яких треба здійснювати парсинг, та набір тегів/елементів, які слід шукати у документах.
- Властивості:
    - string Id – унікальний ідентифікатор сценарію. Автоматично ініціалізується за допомогою Guid.NewGuid().ToString().
    - string Name –назва сценарію (наприклад, «UniveNews» або «PriceParsing»).
    - List<string> Urls – масив рядків, кожен з яких містить URL, що підлягає парсингу.
    - string ContainerTag – назва HTML-тегу, який є «контейнером» для групи елементів, що мають бути опрацьовані (наприклад, div, article, section).

- string ContainerClass – (необов’язкова) назва CSS-класу контейнера. Якщо заповнена, пошук контейнерів виконується за XPath з урахуванням цього класу.
- List<ParsingTag> Tags – перелік об’єктів типу ParsingTag, кожен з яких описує конкретний елемент/піделемент, що слід витягти з контейнера.
- List<ParsingResult> Results – список результатів парсингу, у який записуються отримані дані у структурованому вигляді.

1.2 Клас ParsingTag

- Призначення: Зберігає інформацію про окремий HTML-елемент, який потрібно шукати в межах кожного контейнера.
- Властивості:
  - string Id – унікальний ідентифікатор тегу (ініціалізується через Guid.NewGuid().ToString()).
  - string HtmlTag – назва HTML-тегу, наприклад span, h2, a тощо.
  - [JsonPropertyName("class")] public string Class – назва CSS-класу елемента (атрибуту class). Атрибут перейменовано через JsonPropertyName("class"), щоб уникнути конфлікту зі словом-ключем class.
  - string Text – текстовий фрагмент, який має міститися всередині тегу (наприклад, ключове слово у тексті).
  - ParsingType ParsingType – тип парсингу: або Text (витягування внутрішнього тексту елемента), або Attribute (витягування значення певного атрибуту).
  - string AttributeName – назва атрибуту (наприклад, href, src), якщо ParsingType == ParsingType.Attribute, і потрібно отримати атрибут замість тексту.

- string Key – ключ, під яким витягнуте значення зберігатиметься у словнику Data результату (наприклад, "Title", "Link", "Price"). Якщо Key не задано, використовується назва тегу (HtmlTag) або згенерований GUID.

1.3 Перечислення ParsingType

- Варіанти:
  - Text – парсинг внутрішнього тексту елемента (властивість InnerText).
  - Attribute – парсинг значення вказаного атрибуту (властивість GetAttributeValue(...)).

1.4. Клас ParsingResult

- Призначення: Містить результат одного «проходу» по контейнеру на сторінці. По суті, це словник, у якому ключі – це значення властивостей Key з ParsingTag, а значення – відповідні витягнуті рядки.
- Властивості:
  - int Id – порядковий номер запису в межах одного URL (індекс контейнера у списку HtmlNodeCollection).
  - Dictionary<string, string> Data – безпосередньо дані парсингу. Наприклад:
    { "Title": "Новина 1", "Link": "https://example.com/article" }.

- Сервіс парсингу (простір імен WebParserApp.Services)

1. Інтерфейс IParsingService
   - Опис: Визначає таск для служби парсингу. Містить єдиний метод: Task<List<ParsingResult>> ParseUrl(string url, ParsingScenario scenario); який повинен отримати сторінку за вказаним url, застосувати налаштування із scenario та повернути список об’єктів ParsingResult.

2. Клас ParsingService
    - Реалізація: Наслідує IParsingService і використовує бібліотеку HtmlAgilityPack для роботи з HTML.
    - Метод ParseUrl: Завантаження HTML-документа
        - Завантажує вміст сторінки асинхронно.
    - Формування XPath для контейнерів:
        - Початковий вираз: //{scenario.ContainerTag}.
        - Якщо scenario.ContainerClass не порожній, додається умова contains(@class, '{ContainerClass}').
        - Таким чином знаходяться всі вузли, які відповідають критеріям контейнера.

- Ітерація по кожному контейнеру: Для кожного знайденого HTML-вузла:
    - Створюється порожній словник Dictionary<string, string> data.
    - Для кожного ParsingTag у scenario.Tags:
        - Формується підшлях (відносний XPath), наприклад ".//span[contains(@class, 'price')]" або ".//a[contains(text(), 'Докладніше')]".
        - Виконується SelectSingleNode(subXpath) для пошуку елемента.
        - Якщо вузол знайдено:
            - Якщо ParsingType == ParsingType.Text, у змінну value записується node.InnerText.Trim().
            - Якщо ParsingType == ParsingType.Attribute та AttributeName не порожній, отримується значення атрибуту через node.GetAttributeValue(tag.AttributeName, "").
        - Формується ключ: якщо tag.Key не пустий, використовується він, інакше – tag.HtmlTag або згенерований GUID.
        - Очищається рядок від HTML-сутностей: " замінюється на ", всі ; на пробіл.
        - Записується пара key -> value у data.

- Після обробки всіх тегів створюється об’єкт ParsingResult { Id = i, Data = data }, який додається до списку results.

- Повернення результатів: Список List<ParsingResult> містить результати для кожного контейнеру на сторінці. Якщо контейнерів не знайдено (тобто containers == null), повертається порожній список.

Нижче на Рис. 4.11. приведено код реалізації функції, public async Task<List<ParsingResult>> ParseUrl(string url, ParsingScenario scenario).

```
HtmlWeb web = new HtmlWeb();

HtmlDocument doc = await web.LoadFromWebAsync(url);
List<ParsingResult> results = new List<ParsingResult>();
string containerXpath = $"//{scenario.ContainerTag}";
if (!string.IsNullOrEmpty(scenario.ContainerClass))
    containerXpath += $"[contains(@class, '{scenario.ContainerClass}')]";

HtmlNodeCollection containers = doc.DocumentNode.SelectNodes(containerXpath);
if (containers == null)
    return results;
string subXpath, value, key;
HtmlNode node;

for (int i = 0; i < containers.Count; i++)
{
    HtmlNode container = containers[i];
    Dictionary<string, string> data = new Dictionary<string, string>();
    foreach (ParsingTag tag in scenario.Tags ?? new List<ParsingTag>())
    {
        if (string.IsNullOrWhiteSpace(tag.HtmlTag))
            continue;
        subXpath = $".//{tag.HtmlTag}";
        if (!string.IsNullOrEmpty(tag.Class))
            subXpath += $"[contains(@class, '{tag.Class}')]";
        if (!string.IsNullOrEmpty(tag.Text))
            subXpath += $"[contains(text(), '{tag.Text}')]";
        node = container.SelectSingleNode(subXpath);
        value = string.Empty;
        if (node != null)
        {
            if (tag.ParsingType == ParsingType.Text)
                value = node.InnerText.Trim();
            else if (!string.IsNullOrWhiteSpace(tag.AttributeName))
                value = node.GetAttributeValue(tag.AttributeName, "");
        }
        key = !string.IsNullOrWhiteSpace(tag.Key) ? tag.Key : (!string.IsNullOrWhiteSpace(tag.HtmlTag) ? tag.HtmlTag : Guid.NewGuid().ToString());
        value = value.Replace("&quot;", "\"");
        value = value.Replace(";", " ");
        data[key] = value;
    }
    results.Add(new ParsingResult { Id = i, Data = data });
}
return results;
```

Рис. 4.11. Реалізація функції ParseUrl

3. Репозиторій сценаріїв (простір імен WebParserApp.Services)

Клас ParsingScenarioRepository реалізує роботу з базою даних Access (.accdb) через провайдер OleDb. Його основна задача – зберігати, завантажувати та видаляти збережені сценарії парсингу.

3.1.Конструктор ParsingScenarioRepository(string connectionString)

- Призначення: Отримує рядок підключення до бази даних і зберігає його в приватному полі _connectionString.

3.2.Метод SaveScenario(ParsingScenario scenario)

- Логіка збереження:
  - Відкриває з’єднання з базою OleDbConnection.
  - Виконує SELECT COUNT(*) FROM ParsingScenarios WHERE Id = ? для перевірки, чи існує вже запис з таким Id.
  - Якщо запис є (тобто count > 0):
    - Виконується UPDATE ParsingScenarios SET Name = ?, ContainerTag = ?, ContainerClass = ? WHERE Id = ?.
    - Потім проводиться видалення старих URL: DELETE FROM ScenarioUrls WHERE ScenarioId = ?.
    - Аналогічне видалення старих тегів: DELETE FROM ParsingTags WHERE ScenarioId = ?.
  - Якщо запису немає:
    - Виконується INSERT INTO ParsingScenarios (Id, Name, ContainerTag, ContainerClass) VALUES (?, ?, ?, ?).
  - Далі у циклі по scenario.Urls здійснюється INSERT INTO ScenarioUrls (ScenarioId, Url) VALUES (?, ?) для кожного URL.
  - І нарешті у циклі по scenario.Tags виконується INSERT INTO ParsingTags ([Id], [ScenarioId], [HtmlTag], [Class], [Text], [ParsingType], [AttributeName], [Key]) VALUES (?, ?, ?, ?, ?, ?, ?, ?) для кожного тегу.
  - Після завершення всіх команд метод закриває з’єднання (через блок using).

3.3.Метод ParsingScenario LoadScenario(string scenarioId)

- Логіка завантаження одного сценарію:
  - Створюється порожній об’єкт ParsingScenario scenario = new ParsingScenario().
  - Відкривається з’єднання й виконується SELECT * FROM ParsingScenarios WHERE Id = ?. Якщо запис знайдено, заповнюються поля Id, Name, ContainerTag, ContainerClass.

- Виконується SELECT Url FROM ScenarioUrls WHERE ScenarioId = ? і для кожного запису додається рядок у scenario.Urls.
- Виконується SELECT * FROM ParsingTags WHERE ScenarioId = ? і для кожного запису створюється об’єкт ParsingTag, зчитуються поля (Id, HtmlTag, Class, Text, ParsingType, AttributeName, Key), а потім додається до списку scenario.Tags.

3.4.Метод List<ParsingScenario> LoadAllScenarios()

- Логіка завантаження всіх сценаріїв:
  - Створюється порожній список scenarios.
  - Виконується SELECT * FROM ParsingScenarios і для кожного запису створюється об’єкт ParsingScenario з мінімальним набором полів (Id, Name, ContainerTag, ContainerClass). На цьому етапі Urls та Tags ще порожні.
  - Виконується SELECT ScenarioId, Url FROM ScenarioUrls. Ітерація по результатах: для кожного запису шукається відповідний ParsingScenario у списку scenarios за Id та додається Url до його колекції Urls.
  - Виконується SELECT * FROM ParsingTags. Для кожного запису читаються поля, формується ParsingTag, після чого шукається батьківський ParsingScenario (за ScenarioId) і тег додається до його Tags.
  - Повертається список усіх сценаріїв, кожен з яких вже містить колекції URL-ів і тегів.

3.5.Метод DeleteScenario(string scenarioId)

- Логіка видалення:
  - Виконується DELETE FROM ScenarioUrls WHERE ScenarioId = ? для видалення всіх рядків у таблиці URL-ів.
  - Виконується DELETE FROM ParsingTags WHERE ScenarioId = ? для видалення усіх тегів.

- Виконується DELETE FROM ParsingScenarios WHERE Id = ? для видалення самого запису сценарію.

4. Контролер ASP.NET Core MVC (WebParserApp.Controllers.ParsingController)

Контролер ParsingController відповідає за обробку HTTP-запитів, пов’язаних зі сценаріями парсингу: створення нових, запуск, отримання результатів, редагування та видалення. Використовує інжекцію залежності для IParsingService і самостійно створює екземпляр ParsingScenarioRepository для роботи із БД.

4.1.Поля та конструктор

- private static readonly List<ParsingScenario> _scenarios = new();
  - Список, що утримує у пам’яті всі сценарії, створені протягом роботи програми.
- private readonly IParsingService _parsingService;
  - Служба парсингу, реалізована через HtmlAgilityPack.
- private readonly ParsingScenarioRepository _scenarioRepository = new ParsingScenarioRepository(...)
  - Репозиторій для збереження/завантаження сценаріїв із бази Access. Рядок підключення будується динамічно, враховуючи фізичний шлях до файлу бази.

4.2.Дії (Actions)

1. Home:
   - Відображає головну сторінку, передаючи у подання список усіх сценаріїв, що зберігаються в пам’яті. Якщо список порожній, користувач побачить пустий інтерфейс для створення нового сценарію.
2. Error:
   - Стандартна сторінка помилки.

3. CreateScenario:
    - Приймає об’єкт CreateScenarioRequest (містить Name, Urls, ContainerTag, ContainerClass), створює новий ParsingScenario у пам’яті (_scenarios) і повертає його у форматі JSON. Поле Id генерується всередині конструктора ParsingScenario.
4. AddTag:
    - Додає новий тег (ParsingTag) до конкретного сценарію в пам’яті. Перевіряє коректність request і наявність ScenarioId. Якщо сценарію з таким Id немає, повертає 404 Not Found.
5. RunScenario:
    - Приймає об’єкт ParsingScenario з фронтенду, у якому задані базові параметри парсингу. Для кожного непорожнього URL у scenario.Urls викликає ParseUrl сервісу парсингу (_parsingService.ParseUrl(...)) і формує список анонімних об’єктів, що містять:
        - ScenarioName – назву сценарію;
        - Url – оброблювану адресу;
        - Results – список об’єктів ParsingResult, отриманих із сервісу.
        - Повертає JSON, який фронтенд може використати для відображення результатів.
6. RunAllScenarios:
    - Приймає список усіх сценаріїв (наприклад, завантажених із бази). Для кожного сценарію:
        - Береться перший URL (якщо є);
        - Викликається _parsingService.ParseUrl(url, scenario);
        - Додається до словника allResults під ключем scenario.Name список результатів.

- Також оновлюється поле Results у пам’яті (_scenarios) для кожного сценарію, якщо він був знайдений. Після завершення повертається JSON-словник, де ключі – назви сценаріїв, а значення – їхні результати.

7. GetResults:
    - Повертає у форматі JSON список результатів (List<ParsingResult>) для конкретного сценарію в пам’яті за його Id. Якщо сценарію нема – 404 Not Found.
8. GetAllResults:
    - Повертає словник, де ключі – назви сценаріїв, а значення – їхні списки результатів. Якщо в пам’яті немає жодного сценарію, повертається порожній словник.
9. DeleteTag:
    - Видаляє тег за ключем (Key) із заданого сценарію. Перевіряє наявність сценарію та тегу. У разі успіху повертає 200 OK, інакше – 404 Not Found.
10. EditTag:
    - Редагує властивості певного ParsingTag у сценарії. Повертає відредагований об’єкт у форматі JSON.
11. EditScenario:
    - Редагує базові властивості сценарію (назву, список URL-ів, контейнер). Повертає 200 OK у випадку успіху.
12. DeleteScenario:
    - Видаляє сценарій з пам’яті за його Id.
13. SaveScenarioToDb:
    - Зберігає сценарій із пам’яті в базу даних, використовуючи метод SaveScenario репозиторію.

14.LoadScenariosFromDb:

- Завантажує всі сценарії з бази даних у пам’ять. Очистка _scenarios перед додаванням, щоб уникнути дублювання. Повертає список сценаріїв у JSON-форматі.

15.LoadSelectedScenariosFromDb:

- Завантажує з бази лише ті сценарії, ідентифікатори яких передані у списку scenarioIds. Додає їх у пам’ять та повертає результат.

16.DeleteScenarioFromDb:

- Видаляє сценарій із бази даних за переданим scenarioId. Повертає 200 OK.

17.GetScenarioFromDb:

- Завантажує з бази один сценарій за id і повертає його у форматі JSON.

18.EditScenarioInDb:

- Оновлює (або додає, якщо ще не існує) сценарій у базі через виклик SaveScenario(scenario). Повертає 200 OK.

4.3 Опис інтерфейсу користувача для перегляду результатів імпорту з використанням програми розробленої на C++ MFC та OpenGL.

У цьому підпункті представлено опис інтерфейсу користувача десктоп-програми на C++ MFC з використанням OpenGL для візуалізації результатів парсингу у вигляді таблиці. [9], [10]

Головне вікно програми при запуску можна побачити на Рис. 4.12.

Рис. 4.12. Головне вікно програми

Після натискання на кнопку “Завантажити дані”, користувачу надається діалогове вікно для вибору CSV-файлу для завантаження. Результат натискання на кнопку “Завантажити дані” зображено на Рис. 4.13.

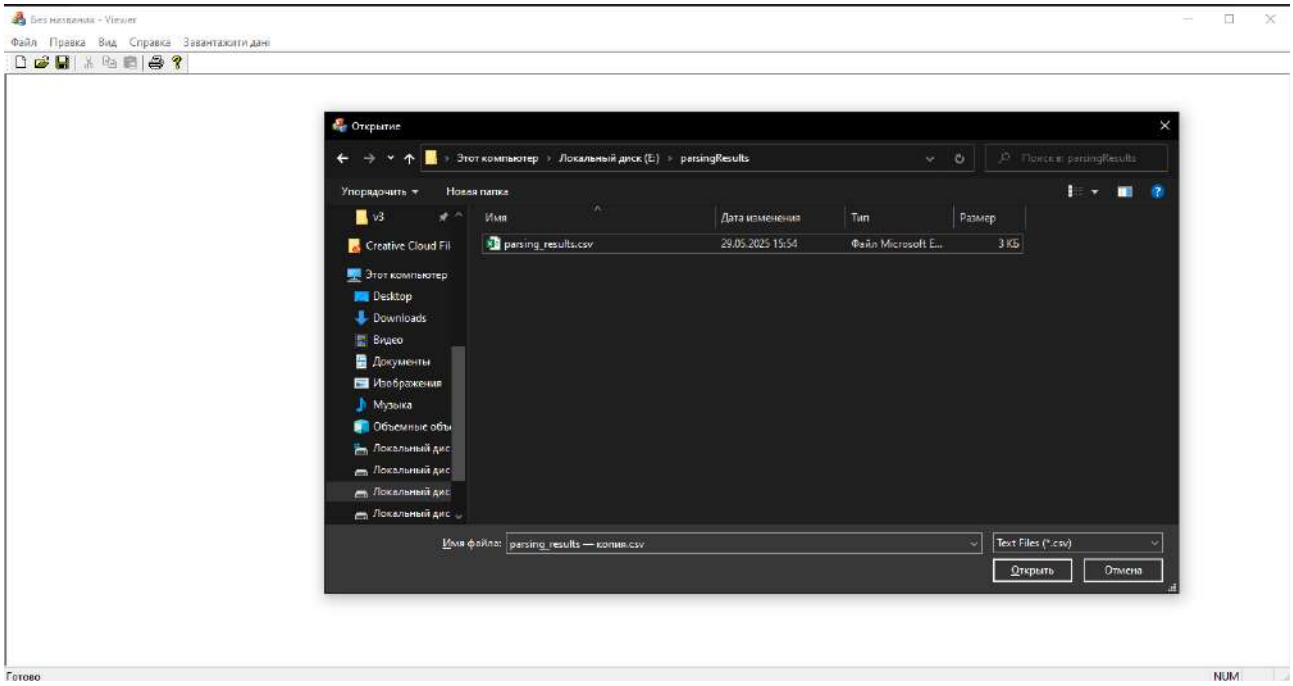


Рис. 4.13. Діалогове вікно вибору CSV файлу для завантаження

Після того як користувач обрав файл, та натиснув на кнопку “Відкрити”, програма оповістить його про кількість завантажених для перегляду сценаріїв. Результат вибору валідного файлу для перегляду показано на Рис. 4.14.

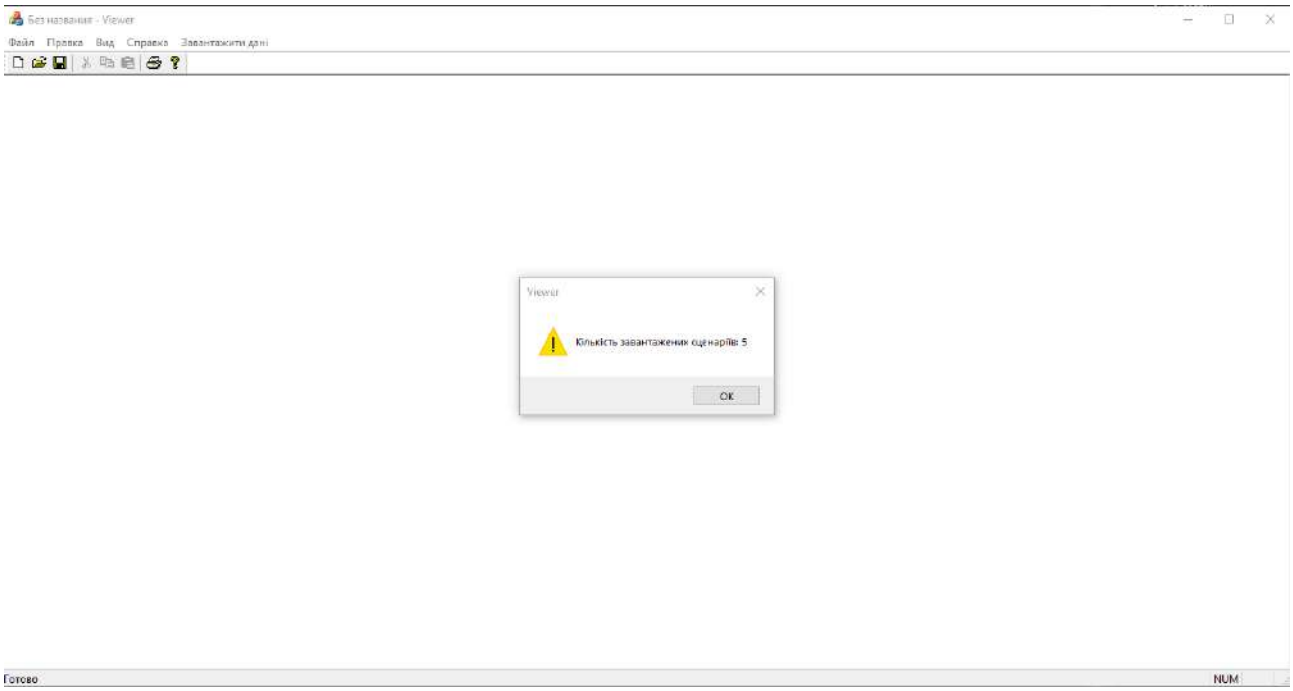


Рис. 4.14. Сповіщення користувача про кількість завантажених сценаріїв

Після натиснення кнопки “ОК”, користувач може натиснути правою кнопкою миші на будь-яке пусте місце у клієнтському вікні програми, після чого відкриється спливаюче меню з назвами сценаріїв, які були завантажені. Результат відкривання спливаючого меню відображено на Рис. 4.15.

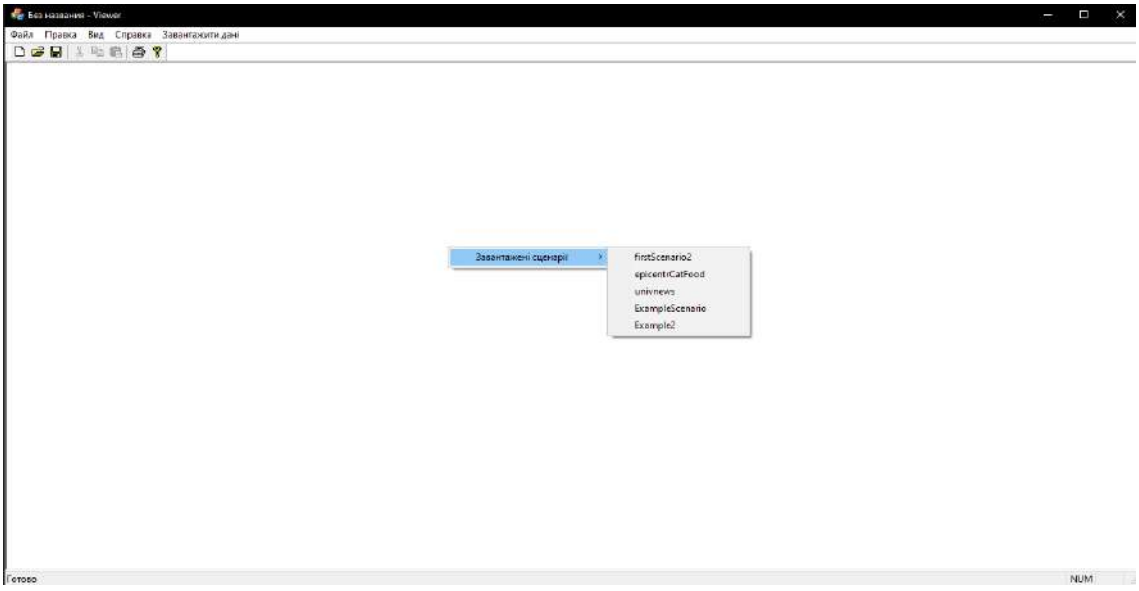
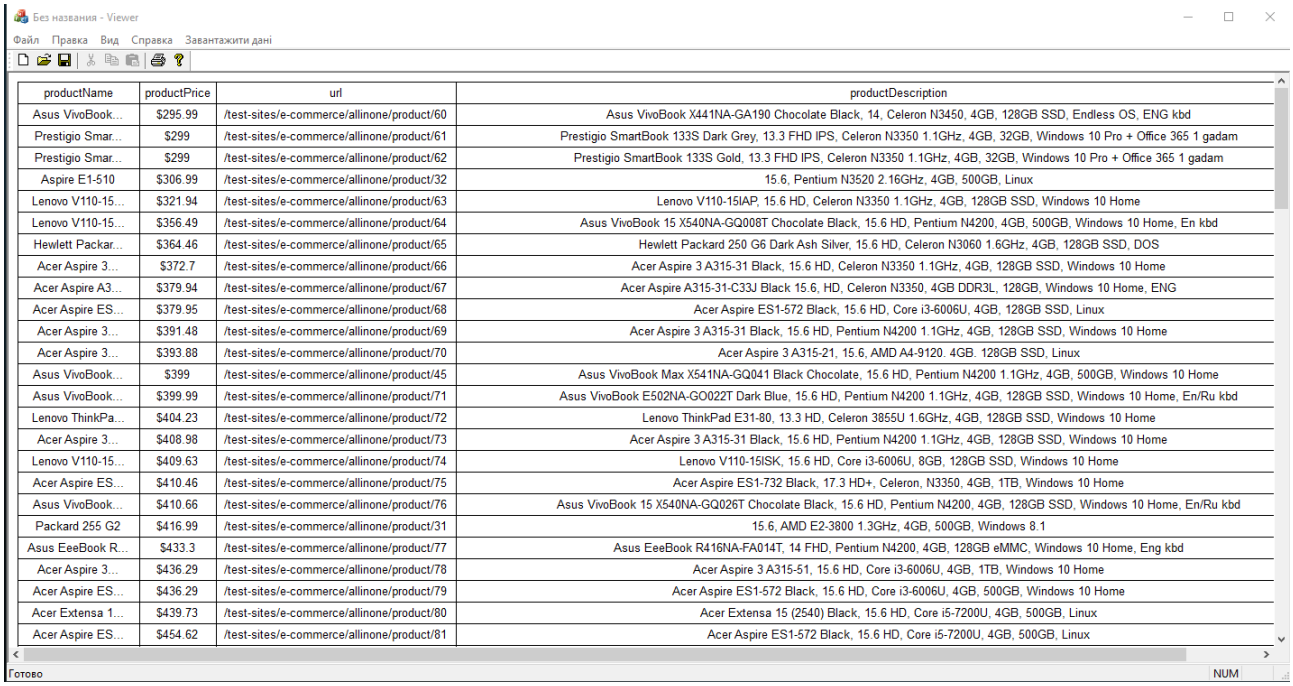


Рис. 4.15. Спливаюче меню з вибором завантажених для відображення сценаріїв

Після того, як користувач обирає пункт із назвою сценарію у спливаючому меню, вони відображаються в зручному для перегляду табличному форматі.

Результат обрання пункту із назвою сценарію у спливаючому меню зображено на Рис. 4.16.



productName	productPrice	url	productDescription
Asus VivoBook...	$295.99	/test-sites/e-commerce/allinone/product/60	Asus VivoBook X441NA-GA190 Chocolate Black, 14, Celeron N3450, 4GB, 128GB SSD, Endless OS, ENG kbd
Prestigio Smar...	$299	/test-sites/e-commerce/allinone/product/61	Prestigio SmartBook 133S Dark Grey, 13.3 FHD IPS, Celeron N3350 1.1GHz, 4GB, 32GB, Windows 10 Pro + Office 365 1 gadam
Prestigio Smar...	$299	/test-sites/e-commerce/allinone/product/62	Prestigio SmartBook 133S Gold, 13.3 FHD IPS, Celeron N3350 1.1GHz, 4GB, 32GB, Windows 10 Pro + Office 365 1 gadam
Aspire E1-510	$306.99	/test-sites/e-commerce/allinone/product/32	15.6, Pentium N3520 2.16GHz, 4GB, 500GB, Linux
Lenovo V110-15...	$321.94	/test-sites/e-commerce/allinone/product/63	Lenovo V110-15IAP, 15.6 HD, Celeron N3350 1.1GHz, 4GB, 128GB SSD, Windows 10 Home
Lenovo V110-15...	$356.49	/test-sites/e-commerce/allinone/product/64	Asus VivoBook 15 X540NA-GQ008T Chocolate Black, 15.6 HD, Pentium N4200, 4GB, 500GB, Windows 10 Home, En kbd
Hewlett Packar...	$364.46	/test-sites/e-commerce/allinone/product/65	Hewlett Packard 250 G6 Dark Ash Silver, 15.6 HD, Celeron N3060 1.6GHz, 4GB, 128GB SSD, DOS
Acer Aspire 3...	$372.7	/test-sites/e-commerce/allinone/product/66	Acer Aspire 3 A315-31 Black, 15.6 HD, Celeron N3350 1.1GHz, 4GB, 128GB SSD, Windows 10 Home
Acer Aspire A3...	$379.94	/test-sites/e-commerce/allinone/product/67	Acer Aspire A315-31-C33J Black 15.6, HD, Celeron N3350, 4GB DDR3L, 128GB, Windows 10 Home, ENG
Acer Aspire ES...	$379.95	/test-sites/e-commerce/allinone/product/68	Acer Aspire ES1-572 Black, 15.6 HD, Core i3-6006U, 4GB, 128GB SSD, Linux
Acer Aspire 3...	$391.48	/test-sites/e-commerce/allinone/product/69	Acer Aspire 3 A315-31 Black, 15.6 HD, Pentium N4200 1.1GHz, 4GB, 128GB SSD, Windows 10 Home
Acer Aspire 3...	$393.88	/test-sites/e-commerce/allinone/product/70	Acer Aspire 3 A315-21, 15.6, AMD A4-9120, 4GB, 128GB SSD, Linux
Asus VivoBook...	$399	/test-sites/e-commerce/allinone/product/45	Asus VivoBook Max X541NA-GQ041 Black Chocolate, 15.6 HD, Pentium N4200 1.1GHz, 4GB, 500GB, Windows 10 Home
Asus VivoBook...	$399.99	/test-sites/e-commerce/allinone/product/71	Asus VivoBook E502NA-GO022T Dark Blue, 15.6 HD, Pentium N4200 1.1GHz, 4GB, 128GB SSD, Windows 10 Home, En/Ru kbd
Lenovo ThinkPa...	$404.23	/test-sites/e-commerce/allinone/product/72	Lenovo ThinkPad E31-80, 13.3 HD, Celeron 3855U 1.6GHz, 4GB, 128GB SSD, Windows 10 Home
Acer Aspire 3...	$408.98	/test-sites/e-commerce/allinone/product/73	Acer Aspire 3 A315-31 Black, 15.6 HD, Pentium N4200 1.1GHz, 4GB, 128GB SSD, Windows 10 Home
Lenovo V110-15...	$409.63	/test-sites/e-commerce/allinone/product/74	Lenovo V110-15ISK, 15.6 HD, Core i3-6006U, 8GB, 128GB SSD, Windows 10 Home
Acer Aspire ES...	$410.46	/test-sites/e-commerce/allinone/product/75	Acer Aspire ES1-732 Black, 17.3 HD+, Celeron, N3350, 4GB, 1TB, Windows 10 Home
Asus VivoBook...	$410.66	/test-sites/e-commerce/allinone/product/76	Asus VivoBook 15 X540NA-GQ026T Chocolate Black, 15.6 HD, Pentium N4200, 4GB, 128GB SSD, Windows 10 Home, En/Ru kbd
Packard 255 G2	$416.99	/test-sites/e-commerce/allinone/product/31	15.6, AMD E2-3800 1.3GHz, 4GB, 500GB, Windows 8.1
Asus EeeBook R...	$433.3	/test-sites/e-commerce/allinone/product/77	Asus EeeBook R416NA-FA014T, 14 FHD, Pentium N4200, 4GB, 128GB eMMC, Windows 10 Home, Eng kbd
Acer Aspire 3...	$436.29	/test-sites/e-commerce/allinone/product/78	Acer Aspire 3 A315-51, 15.6 HD, Core i3-6006U, 4GB, 1TB, Windows 10 Home
Acer Aspire ES...	$436.29	/test-sites/e-commerce/allinone/product/79	Acer Aspire ES1-572 Black, 15.6 HD, Core i3-6006U, 4GB, 500GB, Windows 10 Home
Acer Extensa 1...	$439.73	/test-sites/e-commerce/allinone/product/80	Acer Extensa 15 (2540) Black, 15.6 HD, Core i5-7200U, 4GB, 500GB, Linux
Acer Aspire ES...	$454.62	/test-sites/e-commerce/allinone/product/81	Acer Aspire ES1-572 Black, 15.6 HD, Core i5-7200U, 4GB, 500GB, Linux

Рис. 4.16. Табличне представлення даних із завантаженого сценарію

Для зручного перегляду табличних даних, у програмі використовуються поля для скролінгу, завдяки яким, користувач може переглянути усі записи таблиці. Результат скролінгу зображено на Рис. 4.17.

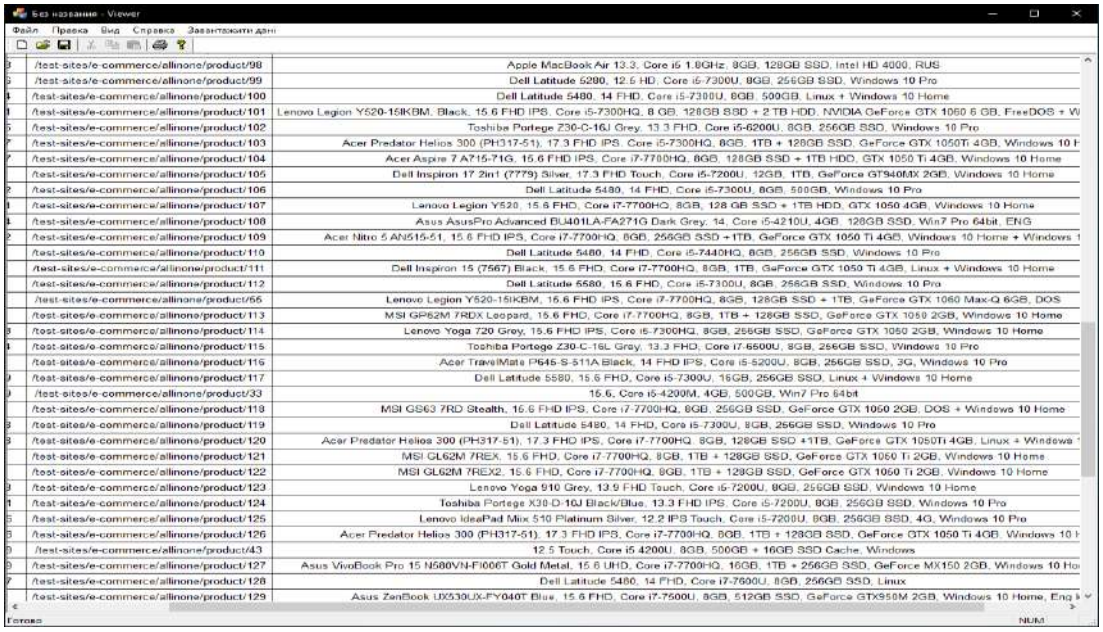


Рис. 4.17. Результат скролінгу по таблиці

Висновок до розділу 4

Представлена система складається з трьох основних шарів:

1. Моделі даних (WebParserApp.Models), які описують структуру сценарію парсингу, окремих тегів і результатів.
2. Сервіс парсингу (IParsingService та його реалізація ParsingService), що відповідає за безпосереднє завантаження HTML й обробку елементів за заданими правилами.
3. Репозиторій (ParsingScenarioRepository), який забезпечує збереження і відновлення сценаріїв у базі Access.
4. Контролер (ParsingController), що реалізує REST-інтерфейс для взаємодії з фронтендом: створення, редагування, запуск сценаріїв, отримання й показ результатів.

Така структура дозволяє:

- Легко додавати або модифікувати алгоритми парсингу, реалізувавши нові класи, що наслідують IParsingService.
- Розширювати базу даних, наприклад, додавши нові таблиці або поля, без зміни логіки контролера.
- Змінювати поведінку збереження (перехід на іншу СУБД) за мінімального впливу на сервіси та контролери.

Завдяки чіткому поділу відповідальностей код є підтримуваним, тестованим і масштабованим для подальшого розвитку дипломного проєкту як повноцінного інструменту веб-парсингу.

Також розроблено зручний додаток для табличного перегляду файлів CSV формату, що дозволяє користувачу переглядати зібрані парсером дані у сгрупованому, гарному вигляді.

ВИСНОВКИ

Результатом дипломної роботи на тему «Розробка парсеру веб-сторінок» стало:

Обґрунтування необхідності автоматизованого збору даних із динамічних веб-ресурсів в Україні через відсутність відкритих API.

Постановка завдання: створити веб-додаток, який дозволяє налаштовувати сценарії парсингу без зміни коду й експортувати результати у CSV та JSON для подальшого аналізу.

Технології та архітектура. Серверна частина реалізована на ASP.NET MVC (.NET 7.0) із Razor-шаблонами, що полегшує розгортання та зменшує залежності. Для парсингу HTML використано HTML Agility Pack з XPath, а дані зберігаються у Microsoft Access (ACCDB) через ADO.NET — достатньо продуктивне рішення для обробки десятків URL за сеанс. Моделі використані у програмі відповідають таблицям Access, а репозиторій забезпечує CRUD-операції через OleDbConnection.

Алгоритм роботи. Користувач у веб-інтерфейсі задає список URL та налаштовує теги. Сторінки завантажуються асинхронно, HAP будує DOM, знаходить контейнери за XPath і витягує текст чи значення атрибутів у словники ключ→значення. Дані групуються в об’єкти і повертаються клієнту як JSON. Контролери надають ендпоінти для створення/редагування/видалення сценаріїв і запуску парсингу. На фронтенді Razor+JavaScript [24] відображаються картки сценаріїв із кнопками запуску та завантаження результатів.

Десктоп-візуалізатор. Написаний на C++/MFC із OpenGL. Програмне забезпечення читає CSV-файл певного формату, з роздільником у вигляді “;” після чого показує результати їх зчитування у таблиці зі скролінгом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Walther, Stephen. ASP.NET MVC Framework Unleashed. Sams Publishing, 2009. – 744 с.

2. Freeman, Adam. Pro ASP.NET MVC 5 (Expert's Voice in .NET). Apress, 2013. – 832 с.

3. Galloway, Jon; Wilson, Brad; Allen, K. Scott; Matson, David. Professional ASP.NET MVC 5. Wrox (John Wiley & Sons), 2014. – 624 с.

4. Palermo, Jeffrey; Bogard, Jimmy; Hexter, Eric; Hinze, Matthew; Skinner, Jeremy. ASP.NET MVC 4 in Action. Manning, 2012. – 440 с.

5. Saeed, Mariwan H. "Real-Time Stock Market Trend Window Using XPath and Html Agility Pack." Basrah Journal of Science, vol. 40, no. 3, 2022. с. 615–629.

6. Ferrara, Emilio; De Meo, Pasquale; Fiumara, Giacomo; Baumgartner, Robert. "Web Data Extraction, Applications and Techniques: A Survey." Knowledge-Based Systems, vol. 70, 2014. с. 301–323.

7. Boehm, Anne; Mead, Ged. Murach's ADO.NET 4 Database Programming with C# 2010. Mike Murach & Associates Inc, 2011. – 712 с.

8. Prosise, Jeff. Programming Windows with MFC (2nd Edition). Microsoft Press, 1999. – 1200 с.

9. Shreiner, Dave; Sellers, Graham; Kessenich, John; Licea-Kane, Bill. OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 4.3 (8th Edition). Addison-Wesley Professional, 2013. – 935 с.

10. Sellers, Graham; Wright, Richard S.; Haemel, Nicholas. OpenGL SuperBible (6th Edition). Addison-Wesley, 2013. – 796 с.

11. Connolly, Thomas M.; Begg, Carolyn E. Database Systems: A Practical Approach to Design, Implementation, and Management (6th Edition). Pearson, 2014.– 1440 с.

12. Elmasri, Ramez; Navathe, Shamkant B. Fundamentals of Database Systems (7th Edition). Pearson, 2015. – 1280 с.

13. Pressman, Roger S.; Maxim, Bruce R. Software Engineering: A Practitioner’s Approach (8th Edition). McGraw-Hill, 2014. – 976 c.

14. Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994. – 416 c.

15. Robbins, Jennifer. Learning Web Design: A Beginner’s Guide to HTML, CSS, JavaScript, and Web Graphics (5th Edition). O’Reilly Media, 2018. – 808 c.

16. Duckett, Jon. HTML and CSS: Design and Build Websites. John Wiley & Sons, 2011. – 490 c.

17. Duckett, Jon. JavaScript and jQuery: Interactive Front-End Web Development. Wiley, 2014. – 640 c.

18. Meyer, Eric. Cascading Style Sheets: The Definitive Guide (4th Edition). O’Reilly Media, 2011. – 256 c.

19. Flanagan, David. JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language O’Reilly Media, 2020. – 706 c.

20. Zeldman, Jeffrey, Ethan Marcotte. Designing with Web Standards (3rd Edition). New Riders, 2009. – 411 c.

21. Marcotte, Ethan. Responsive Web Design. (Brief Books for People Who Make Websites, No. 4), 2011. – 150 c.

22. Krug, Steve. Don’t Make Me Think, Revisited: A Common Sense Approach to Web Usability (3rd Edition) (Voices That Matter). New Riders, 2013. – 216 c.

23. Simpson, Kyle. You Don’t Know JS: Scope & Closures. O’Reilly Media, 2014 – 96 c.

24. Freeman, Eric; Robson, Elisabeth. Head First HTML and CSS (2nd Edition). O’Reilly Media, 2015. – 762 c.

ДОДАТКИ

Додаток А

Опис основних класів веб-парсеру

```
using System.Data.OleDb;
using WebParserApp.Models;

namespace WebParserApp.Services
{
   public class ParsingScenarioRepository
   {
      private readonly string _connectionString;

      public ParsingScenarioRepository(string connectionString)
      {
         _connectionString = connectionString;
      }

      public void SaveScenario(ParsingScenario scenario)
      {
         using (OleDbConnection connection = new
OleDbConnection(_connectionString))
         {
            connection.Open();

            OleDbCommand checkCmd = new OleDbCommand("SELECT
COUNT(*) FROM ParsingScenarios WHERE Id = ?", connection);
            checkCmd.Parameters.AddWithValue("?", scenario.Id);
            int count = (int)checkCmd.ExecuteScalar();

            if (count > 0)
            {
```

Продовження Додатку А

```
                OleDbCommand updateCmd = new OleDbCommand($"UPDATE
ParsingScenarios SET Name = ?, ContainerTag = ?, ContainerClass = ? WHERE Id =
?", connection);
                updateCmd.Parameters.AddWithValue("?", scenario.Name);
                updateCmd.Parameters.AddWithValue("?",
scenario.ContainerTag);
                updateCmd.Parameters.AddWithValue("?",
scenario.ContainerClass);
                updateCmd.Parameters.AddWithValue("?", scenario.Id);
                updateCmd.ExecuteNonQuery();

                OleDbCommand delUrlsCmd = new OleDbCommand("DELETE
FROM ScenarioUrls WHERE ScenarioId = ?", connection);
                delUrlsCmd.Parameters.AddWithValue("?", scenario.Id);
                delUrlsCmd.ExecuteNonQuery();

                OleDbCommand delTagsCmd = new OleDbCommand("DELETE
FROM ParsingTags WHERE ScenarioId = ?", connection);
                delTagsCmd.Parameters.AddWithValue("?", scenario.Id);
                delTagsCmd.ExecuteNonQuery();
            }
            else
            {
                OleDbCommand cmd = new OleDbCommand("INSERT INTO
ParsingScenarios (Id, Name, ContainerTag, ContainerClass) VALUES (?, ?, ?, ?)",
connection);
                cmd.Parameters.AddWithValue("?", scenario.Id);
                cmd.Parameters.AddWithValue("?", scenario.Name);
                cmd.Parameters.AddWithValue("?", scenario.ContainerTag);
```

```
                cmd.Parameters.AddWithValue("?", scenario.ContainerClass);
                cmd.ExecuteNonQuery();
            }

            foreach (string url in scenario.Urls)
            {
                OleDbCommand urlCmd = new OleDbCommand("INSERT INTO
ScenarioUrls (ScenarioId, Url) VALUES (?, ?)", connection);
                urlCmd.Parameters.AddWithValue("?", scenario.Id);
                urlCmd.Parameters.AddWithValue("?", url);
                urlCmd.ExecuteNonQuery();
            }

            foreach (ParsingTag tag in scenario.Tags)
            {
                OleDbCommand tagCmd = new OleDbCommand(
                    "INSERT INTO ParsingTags ([Id], [ScenarioId], [HtmlTag],
[Class], [Text], [ParsingType], [AttributeName], [Key]) VALUES (?, ?, ?, ?, ?, ?, ?,
?)", connection);
                tagCmd.Parameters.AddWithValue("?", tag.Id);
                tagCmd.Parameters.AddWithValue("?", scenario.Id);
                tagCmd.Parameters.AddWithValue("?", tag.HtmlTag);
                tagCmd.Parameters.AddWithValue("?", tag.Class);
                tagCmd.Parameters.AddWithValue("?", tag.Text);
                tagCmd.Parameters.AddWithValue("?",
tag.ParsingType.ToString());
                tagCmd.Parameters.AddWithValue("?", tag.AttributeName);
                tagCmd.Parameters.AddWithValue("?", tag.Key);
                tagCmd.ExecuteNonQuery();
```

```
            }
        }
    }

    public ParsingScenario LoadScenario(string scenarioId)
    {
        ParsingScenario scenario = new ParsingScenario();

        using (OleDbConnection connection = new OleDbConnection(_connectionString))
        {
            connection.Open();

            OleDbCommand cmd = new OleDbCommand("SELECT * FROM ParsingScenarios WHERE Id = ?", connection);
            cmd.Parameters.AddWithValue("?", scenarioId);
            using (OleDbDataReader reader = cmd.ExecuteReader())
            {
                if (reader.Read())
                {
                    scenario.Id = reader["Id"].ToString();
                    scenario.Name = reader["Name"].ToString();
                    scenario.ContainerTag = reader["ContainerTag"].ToString();
                    scenario.ContainerClass = reader["ContainerClass"].ToString();
                }
            }

            OleDbCommand urlCmd = new OleDbCommand("SELECT Url FROM ScenarioUrls WHERE ScenarioId = ?", connection);
```

```
            urlCmd.Parameters.AddWithValue("?", scenarioId);
            using (OleDbDataReader reader = urlCmd.ExecuteReader())
            {
                while (reader.Read())
                {
                    scenario.Urls.Add(reader["Url"].ToString());
                }
            }

            OleDbCommand tagCmd = new OleDbCommand("SELECT * FROM ParsingTags WHERE ScenarioId = ?", connection);
            tagCmd.Parameters.AddWithValue("?", scenarioId);
            using (OleDbDataReader reader = tagCmd.ExecuteReader())
            {
                while (reader.Read())
                {
                    ParsingTag tag = new ParsingTag
                    {
                        Id = reader["Id"].ToString(),
                        HtmlTag = reader["HtmlTag"].ToString(),
                        Class = reader["Class"].ToString(),
                        Text = reader["Text"].ToString(),
                        ParsingType = Enum.TryParse<ParsingType>(reader["ParsingType"].ToString(), out ParsingType pt) ? pt : ParsingType.Text,
                        AttributeName = reader["AttributeName"].ToString(),
                        Key = reader["Key"].ToString()
                    };
                    scenario.Tags.Add(tag);
```

```
                }
            }
        }

        return scenario;
    }

    public List<ParsingScenario> LoadAllScenarios()
    {
        List<ParsingScenario> scenarios = new List<ParsingScenario>();
        using (OleDbConnection connection = new
OleDbConnection(_connectionString))
        {
            connection.Open();
            OleDbCommand cmd = new OleDbCommand("SELECT * FROM
ParsingScenarios", connection);
            using (OleDbDataReader reader = cmd.ExecuteReader())
            {
                while (reader.Read())
                {
                    ParsingScenario scenario = new ParsingScenario
                    {
                        Id = reader["Id"].ToString(),
                        Name = reader["Name"].ToString(),
                        ContainerTag = reader["ContainerTag"].ToString(),
                        ContainerClass = reader["ContainerClass"].ToString(),
                        Urls = new List<string>(),
                        Tags = new List<ParsingTag>()
                    };
```

```
                scenarios.Add(scenario);
            }
        }
        OleDbCommand urlCmd = new OleDbCommand("SELECT ScenarioId, Url FROM ScenarioUrls", connection);
        using (OleDbDataReader reader = urlCmd.ExecuteReader())
        {
            string scenarioId,url;
            ParsingScenario scenario;
            while (reader.Read())
            {
                scenarioId = reader["ScenarioId"].ToString();
                url = reader["Url"].ToString();
                scenario = scenarios.Find(s => s.Id == scenarioId);
                if (scenario != null)
                {
                    scenario.Urls.Add(url);
                }
            }
        }
        OleDbCommand tagCmd = new OleDbCommand("SELECT * FROM ParsingTags", connection);
        using (OleDbDataReader reader = tagCmd.ExecuteReader())
        {
            string scenarioId, url;
            ParsingScenario scenario;
            while (reader.Read())
            {
                scenarioId = reader["ScenarioId"].ToString();
```

```
                    scenario = scenarios.Find(s => s.Id == scenarioId);
                    if (scenario != null)
                    {
                        ParsingTag tag = new ParsingTag
                        {
                            Id = reader["Id"].ToString(),
                            HtmlTag = reader["HtmlTag"].ToString(),
                            Class = reader["Class"].ToString(),
                            Text = reader["Text"].ToString(),
                            ParsingType = Enum.TryParse<ParsingType>(reader["ParsingType"].ToString(), out ParsingType pt) ? pt : ParsingType.Text,
                            AttributeName = reader["AttributeName"].ToString(),
                            Key = reader["Key"].ToString()
                        };
                        scenario.Tags.Add(tag);
                    }
                }
            }
        }
        return scenarios;
    }

    public void DeleteScenario(string scenarioId)
    {
        using (OleDbConnection connection = new OleDbConnection(_connectionString))
        {
            connection.Open();
```

```
                OleDbCommand delUrlsCmd = new OleDbCommand("DELETE
FROM ScenarioUrls WHERE ScenarioId = ?", connection);
                delUrlsCmd.Parameters.AddWithValue("?", scenarioId);
                delUrlsCmd.ExecuteNonQuery();

                OleDbCommand delTagsCmd = new OleDbCommand("DELETE
FROM ParsingTags WHERE ScenarioId = ?", connection);
                delTagsCmd.Parameters.AddWithValue("?", scenarioId);
                delTagsCmd.ExecuteNonQuery();

                OleDbCommand delScenarioCmd = new OleDbCommand("DELETE
FROM ParsingScenarios WHERE Id = ?", connection);
                delScenarioCmd.Parameters.AddWithValue("?", scenarioId);
                delScenarioCmd.ExecuteNonQuery();

            }
        }
    }
}

    amespace WebParserApp.Services
    {
        public interface IParsingService
        {
            Task<List<ParsingResult>> ParseUrl(string url, ParsingScenario
scenario);
        }

        public class ParsingService : IParsingService
```

Продовження Додатку А

```
        {
            public async Task<List<ParsingResult>> ParseUrl(string url, ParsingScenario scenario)
            {
                HtmlWeb web = new HtmlWeb();

                HtmlDocument doc = await web.LoadFromWebAsync(url);
                List<ParsingResult> results = new List<ParsingResult>();
                string containerXpath = $"//{scenario.ContainerTag}";
                if (!string.IsNullOrEmpty(scenario.ContainerClass))
                    containerXpath += $"[contains(@class, '{scenario.ContainerClass}')]";

                HtmlNodeCollection containers = doc.DocumentNode.SelectNodes(containerXpath);
                if (containers == null)
                    return results;
                string subXpath, value, key;
                HtmlNode node;

                for (int i = 0; i < containers.Count; i++)
                {
                    HtmlNode container = containers[i];
                    Dictionary<string, string> data = new Dictionary<string, string>();
                    foreach (ParsingTag tag in scenario.Tags ?? new List<ParsingTag>())
                    {
                        if (string.IsNullOrWhiteSpace(tag.HtmlTag))
                            continue;
                        subXpath = $".//{tag.HtmlTag}";
```

Продовження Додатку А

```
                    if (!string.IsNullOrEmpty(tag.Class))
                        subXpath += $"[contains(@class, '{tag.Class}')]";
                    if (!string.IsNullOrEmpty(tag.Text))
                        subXpath += $"[contains(text(), '{tag.Text}')]";
                    node = container.SelectSingleNode(subXpath);
                    value = string.Empty;
                    if (node != null)
                    {
                        if (tag.ParsingType == ParsingType.Text)
                            value = node.InnerText.Trim();
                        else if (!string.IsNullOrWhiteSpace(tag.AttributeName))
                            value = node.GetAttributeValue(tag.AttributeName, "");
                    }
                    key = !string.IsNullOrWhiteSpace(tag.Key) ? tag.Key : (!string.IsNullOrWhiteSpace(tag.HtmlTag) ? tag.HtmlTag : Guid.NewGuid().ToString());
                    value = value.Replace("&quot;", "\"");
                    value = value.Replace(";", " ");
                    data[key] = value;
                }
                results.Add(new ParsingResult { Id = i, Data = data });
            }
            return results;
        }

    }
}
```

Додаток Б

Опис функцій фронт-енд складової веб-інтерфейсу

```
function runScenario(scenarioId) {
    const scenario = getScenarioFromDOM(scenarioId);
    $.ajax({
        url: `/Parsing/RunScenario`,
        type: 'POST',
        contentType: 'application/json',
        data: JSON.stringify(scenario),
        success: function(results) {
            displayResults(results);
        },
        error: function() {
            alert('Помилка при запуску сценарію');
        }
    });
}

function runAllScenarios() {
    const scenarios = [];
    $('[data-scenario-id]').each(function() {
        const scenarioId = $(this).data('scenario-id');
        scenarios.push(getScenarioFromDOM(scenarioId));
    });
    $.ajax({
        url: '/Parsing/RunAllScenarios',
        type: 'POST',
        contentType: 'application/json',
        data: JSON.stringify(scenarios),
        success: function(results) {
```

```
            displayResults(results);
        },
        error: function() {
            alert('Помилка при запуску всіх сценаріїв');
        }
    });
}
function downloadResultsJSON() {
    $.ajax({
        url: '/Parsing/GetAllResults',
        type: 'GET',
        success: function(results) {
            const dataStr = JSON.stringify(results, null, 2);
            const dataUri = 'data:application/json;charset=utf-8,'+
encodeURIComponent(dataStr);
            const exportFileDefaultName = 'parsing_results.json';

            const linkElement = document.createElement('a');
            linkElement.setAttribute('href', dataUri);
            linkElement.setAttribute('download', exportFileDefaultName);
            linkElement.click();
        },
        error: function() {
            alert('Помилка при завантаженні результатів');
        }
    });
}

function downloadResultsCSV() {
```

Продовження Додатку Б

```
        $.ajax({
            url: '/Parsing/GetAllResults',
            type: 'GET',
            success: function (results) {
                let csv = '';
                for (const scenarioName in results) {
                    const scenarioResults = results[scenarioName];
                    if (!scenarioResults || scenarioResults.length === 0) continue;
                    const headers = Object.keys(scenarioResults[0].data ||
scenarioResults[0].Data || {});
                    csv += `Scenario: ${scenarioName}\n`;
                    csv += headers.join(';') + '\n';
                    scenarioResults.forEach(res => {
                        const row = headers.map(h => {
                            const raw = (res.data || res.Data || {})[h] ?? '';
                            return String(raw).replace(/"/g, '');
                        });
                        csv += row.join(';') + '\n';
                    });

                    csv += '\n';
                }

                const blob = new Blob([csv], { type: 'text/csv;charset=utf-8;' });
                const link = document.createElement('a');
                link.href = URL.createObjectURL(blob);
                link.setAttribute('download', 'parsing_results.csv');
                document.body.appendChild(link);
                link.click();
```

```
            document.body.removeChild(link);
        },
        error: function () {
            alert('Помилка при збереженні у CSV');
        }
    });

function deleteScenario(scenarioId) {
    if (!confirm('Ви впевнені, що хочете видалити цей сценарій?')) return;
    $.ajax({
        url: '/Parsing/DeleteScenario',
        type: 'POST',
        contentType: 'application/json',
        data: JSON.stringify({ id: scenarioId }),
        success: function() {
            location.reload();
        },
        error: function(xhr, status, error) {
            alert('Помилка при видаленні сценарію');
            console.error('Delete scenario error:', xhr, status, error);
        }
    });
}
```

Додаток В

Опис функцій використаних у программі для візуалізування даних

```
void CViewerDoc::SaveScenariosAsBitmaps()
{
	int margin = 10;
	int paddingX = 6;
	int paddingY = 4;
	int maxLen = 255;
	auto Trunc = [&](const CString& src)->CString {
		if (src.GetLength() <= maxLen) return src;
		return src.Left(maxLen - 3) + L"...";
		};
	CClientDC screenDC(AfxGetMainWnd());
	CDC	memDC;
	memDC.CreateCompatibleDC(&screenDC);
	LOGFONT lf = {};
	lf.lfHeight = 16;
	lf.lfWeight = FW_NORMAL;
	_tcscpy_s(lf.lfFaceName, _T("Arial"));
	CFont font;
	font.CreateFontIndirect(&lf);
	CFont* oldFont = memDC.SelectObject(&font);
	TEXTMETRIC tm;
	memDC.GetTextMetrics(&tm);
	int rowHeight = tm.tmHeight + tm.tmExternalLeading + paddingY * 2;
	for (int idx = 0; idx < scenarios.size(); ++idx)
	{
		Scenario& sc = scenarios[idx];
		int	cols = sc.columnsAmount;
```

```
int      dataRows = sc.rows.size();
std::vector<int> colWidths(cols, 0);
CSize sz;
for (int c = 0; c < cols; ++c)
{
        sz = memDC.GetTextExtent(sc.headers[c]);
        colWidths[c] = sz.cx;
}
CString cellText;

for (CString& row : sc.rows)
{
        std::vector<CString> cells = Split(row, _T(';'));
        for (int c = 0; c < cols; ++c)
        {
                cellText = (c < (int)cells.size()) ? cells[c] : L"";
                CSize sz = memDC.GetTextExtent(cellText);
                colWidths[c] = max(colWidths[c], sz.cx);
        }
}
for (int c = 0; c < cols; ++c)
        colWidths[c] += paddingX * 2;
int bmpW = margin * 2;
for (int w : colWidths) bmpW += w;
int bmpH = margin * 2 + rowHeight * (1 + dataRows);
CBitmap bmp;
bmp.CreateCompatibleBitmap(&screenDC, bmpW, bmpH);
CBitmap* oldBmp = memDC.SelectObject(&bmp);
CBrush white(RGB(255, 255, 255));
```

Продовження Додатку В

```
memDC.FillRect(CRect(0, 0, bmpW, bmpH), &white);
memDC.SetTextColor(RGB(0, 0, 0));
memDC.SetBkMode(TRANSPARENT);
int x = margin;
int y = margin;

for (int c = 0; c < cols; ++c)
{
	CRect cellRect(x, y, x + colWidths[c], y + rowHeight);
	DrawOneCell(&memDC, sc.headers[c], cellRect,
		1 | 2 | 4 | 8,
		DT_VCENTER | DT_CENTER | DT_SINGLELINE,
		1);
	x += colWidths[c];
}
std::vector<CString> cells;
CString txt;
for (int r = 0; r < dataRows; ++r)
{
	x = margin;
	y = margin + (r + 1) * rowHeight;
	cells = Split(sc.rows[r], _T(';'));
	for (int c = 0; c < cols; ++c)
	{
		txt = (c < (int)cells.size()) ? cells[c] : L"";
		CRect cellRect(x, y, x + colWidths[c], y + rowHeight);
		DrawOneCell(&memDC, txt, cellRect,
			1 | 2 | 4 | 8,
```

Продовження Додатку В

```
                                        DT_VCENTER | DT_CENTER | DT_SINGLELINE,
                                        1);
                                x += colWidths[c];
                        }
                }
                CImage img;
                img.Attach((HBITMAP)bmp.GetSafeHandle());
                CString fileName;

                fileName.Format(_T("%s.bmp"), sc.name);
                img.Save(fileName);
                img.Detach();

                memDC.SelectObject(oldBmp);
        }

        memDC.SelectObject(oldFont);
}

void CViewerDoc::OnLoadData()
{
        CFileDialog dlg(
                TRUE,
                _T("csv"),
                nullptr,
                OFN_FILEMUSTEXIST | OFN_HIDEREADONLY,
                _T("Text Files (*.csv)|*.csv|All Files (*.*)|*.*||")
        );
```

```
if (dlg.DoModal() != IDOK)
    return;
CString path = dlg.GetPathName();
std::ifstream ifs;
ifs.open(path, std::ios::binary);
if (!ifs.is_open()) {
    AfxMessageBox(_T("Не вдалося відкрити файл"), MB_ICONERROR);
    return;
}
scenarios.clear();
CString currentName;
std::vector<CString> currentHeaders, currentRows;
bool nextIsHeader = false;
std::string rawLine;
while (std::getline(ifs, rawLine))
{
    if (!rawLine.empty() && rawLine.back() == '\r')
        rawLine.pop_back();
    int wlen = ::MultiByteToWideChar(
        CP_UTF8, 0,
        rawLine.c_str(), (int)rawLine.size(),
        nullptr, 0
    );
    std::wstring wline;
    if (wlen > 0) {
        wline.resize(wlen);
        ::MultiByteToWideChar(
            CP_UTF8, 0,
```

Продовження Додатку В

```
                    rawLine.c_str(), (int)rawLine.size(),
                    &wline[0], wlen
                );
            }
            CString line(wline.c_str());
            line.Trim();
            if (line.IsEmpty())
                continue;
            if (line.Left(9).CompareNoCase(_T("Scenario:")) == 0)
            {
                if (!currentName.IsEmpty())
                    scenarios.emplace_back(currentName,
currentHeaders, currentRows);

                currentName = line.Mid(9);
                currentName.Trim();
                currentHeaders.clear();
                currentRows.clear();
                nextIsHeader = true;}
            else if (nextIsHeader){
                currentHeaders = Split(line, _T(';'));
                nextIsHeader = false;
            }
            else
            {
                currentRows.push_back(line);}}
        if (!currentName.IsEmpty())
            scenarios.emplace_back(currentName,          currentHeaders,
currentRows);
```

Продовження Додатку В

```
        CString str;
        str.Format(_T("Кількість завантажених сценаріїв: %d"), scenarios.size());
        AfxMessageBox(str);
        SaveScenariosAsBitmaps();
    }
```