МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

МАЛКО Владислав Володимирович

Програмний модуль збору даних про довгострокову оренду житла з використанням фреймворків Scrapy та Playwright / Software module for collecting long-term housing rental data using Scrapy and Playwright frameworks

Спеціальність 122 – Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

Кваліфікаційна робота

Виконав студент групи КН-42
В.В. Малко

Науковий керівник:
к.т.н., доцент П.Є. Биковий

Кваліфікаційну роботу допущено до захисту
«___»_____ 2025 р.

В.о. завідувача кафедри
_____ Н.М. Васильків

Тернопіль – 2025

Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність 122 – Комп'ютерні науки
освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Н.М. Васильків
«____» _____2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
МАЛКО Владислав Володимирович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: Програмний модуль збору даних про довгострокову оренду житла з використанням фреймворків Scrapy та Playwright / Software module for collecting long-term housing rental data using Scrapy and Playwright frameworks
керівник роботи Биковий П.Є. к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 20 грудня 2024 р. № 938.

2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.

4. Основні питання, які потрібно розробити:
- проаналізувати предметну область збору даних про нерухомість та існуючі рішення, визначити особливості та виклики веб-скрапінгу даних про нерухомість;
- сформувати концепцію та спроєктувати архітектуру системи збору даних про нерухомість, що забезпечує автоматизований збір, обробку та зберігання інформації;
- розробити основні алгоритми збору та обробки даних, включаючи універсальний ItemLoader, системи автоматичної пагінації, аналізу табличних структур та аналізу адрес;
- реалізувати механізми роботи з динамічним контентом, захист від блокувань, автоматизацію життєвого циклу скраперів та механізми високочастотного збору даних.

5. Перелік графічного матеріалу в роботі:
- структура проекту;
- схема бази даних.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 20 грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи	до 01.01. 2025 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.02. 2025 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 01.04.2025 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 01.05. 2025 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2025 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2025 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2025 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 10.06.2025 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 10.06. 2025 р.	

Студент_____ Малко В.В.
(підпис) (прізвище та ініціали)

Керівник кваліфікаційної роботи_____ Биковий П.Є.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Програмний модуль збору даних про довгострокову оренду житла з використанням фреймворків Scrapy та Playwright» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки» написана обсягом в 70 сторінок і містить 8 ілюстрацій, 1 таблиця, 4 додатки та 37 використаних джерел.

Метою кваліфікаційної роботи є розробка та впровадження модулів котрі складають частину системи високочастотного збору даних про нерухомість з веб-ресурсів, що забезпечує високу продуктивність, точність та масштабованість.

Об'єктом дослідження є процеси автоматизованого збору та обробки даних про нерухомість з веб-ресурсів.

Предмет дослідження – модулі високочастотного збору даних про нерухомість з веб-ресурсів.

Розроблені модулі демонструють високу продуктивність (обробка понад 100 000 сторінок на годину), точність вилучення даних (понад 98% для структурованих даних) та масштабованість (підтримка 17 країн та понад 10 000 доменів). Вони забезпечують універсальний ItemLoader, системи автоматичної пагінації, аналізу табличних структур та аналізу адрес, механізми роботи з динамічним контентом, захисту від блокувань, автоматизації життєвого циклу скраперів та високочастотного збору даних. Практичне значення полягає у створенні повноцінної екосистеми для автоматизованого збору даних про нерухомість, що може бути використана компаніями-агрегаторами, інвестиційними фондами, аналітичними агентствами та науково-дослідними установами.

Ключові слова: ВЕБ-СКРАПІНГ, ЗБІР ДАНИХ, НЕРУХОМІСТЬ, АВТОМАТИЗАЦІЯ, МАСШТАБОВАНІСТЬ.

ANNOTATION

The qualification work on the topic "Software module for collecting long-term housing rental data using Scrapy and Playwright frameworks" for obtaining the degree of "bachelor" in specialty 122 "Computer Science" of the educational program "Computer Science" is written in a volume of 70 pages and contains 8 illustrations, 1 tables, 4 appendices and 37 sources used.

The purpose of the qualification paper is to develop and implement a high-frequency real estate data collection system from web resources, ensuring high performance, accuracy, and scalability.

The object of research is the processes of automated collection and processing of real estate data from web resources.

The subject of research is a high-frequency real estate data collection system from web resources.

The developed system demonstrates high performance (processing over 100,000 pages per hour), data extraction accuracy (over 98% for structured data), and scalability (support for 17 countries and over 10,000 domains). It includes a universal ItemLoader, automatic pagination systems, table structure analysis and address analysis, mechanisms for working with dynamic content, blocking protection, automation of the scraper life cycle, and high-frequency data collection. The practical significance lies in creating a complete ecosystem for automated real estate data collection, which can be used by aggregator companies, investment funds, analytical agencies, and research institutions.

Key words: WEB SCRAPING, DATA COLLECTION, REAL ESTATE, AUTOMATION, SCALABILITY.

ЗМІСТ

ВСТУП

Актуальність теми дослідження зумовлена стрімким розвитком ринку нерухомості у світі та необхідністю автоматизованого збору й аналізу даних з великої кількості веб-ресурсів. Сучасний ринок нерухомості характеризується високою динамічністю, різноманітністю пропозицій та географічною розпорошеністю даних, що створює значні виклики для аналітиків та компаній, які працюють у цій галузі. Традиційні методи збору інформації часто виявляються неефективними, ресурсозатратними та не дозволяють отримувати дані в режимі реального часу.

Наявність структурованих та актуальних даних про нерухомість є критично важливою для різних категорій користувачів: інвесторів, ріелторів, девелоперів, аналітичних агентств та потенційних покупців. Однак, збір таких даних ускладнюється через відсутність єдиних стандартів подання інформації, різноманітність форматів та структур веб-сайтів, а також захисні механізми від автоматизованого сканування.

Розробка модулів та спеціалізованої системи, здатної ефективно збирати, структурувати та аналізувати дані з різноманітних джерел інформації про нерухомість, є актуальним завданням, що має суттєве практичне значення для галузі.

Метою роботи є розробка модулів що є частиною системи автоматизованого збору та структуризації даних про нерухомість з великої кількості веб-ресурсів у різних країнах світу.

Для досягнення мети поставлено такі завдання:

1. Провести аналіз існуючих методів та інструментів веб-скрапінгу, визначити їх переваги та обмеження.
2. Розробити архітектуру модулів та системи, що забезпечує масштабованість, стійкість до відмов та гнучкість щодо обробки різних типів веб-ресурсів.
3. Спроектувати та реалізувати модулі для обробки динамічного контенту, захисту від блокувань та валідації отриманих даних.

4. Створити механізми автоматизації життєвого циклу скраперів, включаючи розгортання, моніторинг та контроль якості.
5. Розробити систему зберігання та управління отриманими даними.
6. Провести тестування та оцінку ефективності розробленої системи.

Об'єктом дослідження є процес автоматизованого збору та обробки даних з веб-ресурсів, що містять інформацію про нерухомість.

Предметом дослідження є методи та засоби розробки масштабованої системи веб-скрапінгу для отримання структурованих даних про нерухомість з різноманітних джерел.

Методи дослідження базуються на використанні принципів системного аналізу, об'єктно-орієнтованого програмування, асинхронного програмування та паралельних обчислень. У роботі застосовуються методи веб-скрапінгу, аналізу DOM-структури [21], обробки природної мови для вилучення релевантної інформації, а також статистичні методи для оцінки ефективності розробленої системи.

Практичне значення одержаних результатів полягає в створенні модулів та повноцінної екосистеми для автоматизованого збору даних про нерухомість, яка може бути використана компаніями-агрегаторами нерухомості, інвестиційними фондами, аналітичними агентствами та науково-дослідними установами для отримання актуальної інформації про стан ринку нерухомості в різних країнах світу. Розроблені в рамках роботи методи та інструменти можуть бути адаптовані для збору даних в інших предметних областях, що підвищує їх практичну цінність.

Результати дослідження опубліковано в матеріалах науково-практичної конференції «Інтелектуальні інформаційні технології в прикладних дослідженнях» (IITAR – 2025), м. Тернопіль, 27–29 травня 2025 р. (додаток Б).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

Предметна область дослідження охоплює ринок нерухомості та пов'язані з ним інформаційні ресурси в мережі Інтернет. Ринок нерухомості є одним із найбільших сегментів світової економіки, що характеризується високою вартістю активів, значним обсягом транзакцій та різноманітністю об'єктів. Для ефективного функціонування цього ринку критично важливим є доступ до актуальної та структурованої інформації.

Основними джерелами даних про нерухомість в Інтернеті є:

1. Спеціалізовані сайти оголошень про нерухомість (Zillow [11], Rightmove [12], ImmoWeb [13]).
2. Веб-сайти агентств нерухомості та девелоперських компаній.
3. Платформи для оренди житла (Airbnb, Booking.com та ін.).
4. Національні та регіональні класифайди (сайти оголошень загального призначення).
5. Спеціалізовані бази даних та реєстри нерухомості.

Кожне з цих джерел має свої особливості щодо структури даних, частоти оновлення, обсягу інформації та механізмів захисту від автоматизованого сканування. Це створює значні технічні виклики для розробки універсальної системи збору даних.

Типова структура даних про об'єкт нерухомості включає наступні категорії інформації:

- базові характеристики (тип нерухомості, площа, кількість кімнат, поверх);
- географічні дані (адреса, координати, район);
- цінові показники (вартість продажу або оренди, історія змін ціни);
- візуальні матеріали (фотографії, відео, плани);
- контактна інформація (телефон, email);

- додаткові параметри (наявність меблів, ремонт, інфраструктура);
- метадані оголошення (дата публікації, ідентифікатор, статус).

Важливою особливістю даної предметної області є значна географічна диверсифікація, що проявляється у відмінностях термінології, одиниць виміру, форматів представлення дат та чисел, а також законодавчих вимог до опису нерухомості в різних країнах. Це вимагає створення гнучких механізмів нормалізації та стандартизації даних для їх подальшого використання.

1.2 Огляд існуючих рішень

У сучасному цифровому середовищі існує широкий спектр рішень для автоматизованого збору даних з веб-ресурсів, які можна умовно поділити на кілька основних категорій: спеціалізовані фреймворки, хмарні платформи, а також комерційні агрегатори даних. Кожен з підходів має свої переваги, недоліки та сфери застосування, що зумовлює необхідність їх критичного аналізу при побудові власної системи збору інформації про нерухомість.

Серед спеціалізованих фреймворків найбільш відомим і поширеним є Scrapy [1], що написаний мовою програмування Python. Його сильними сторонами є висока продуктивність, досягнута за рахунок асинхронного підходу до обробки запитів, модульна архітектура, яка дозволяє гнучке розширення, а також підтримка популярних мов розмітки (XPath, CSS-селектори). Крім того, Scrapy має вбудовані механізми автоматичного слідування за посиланнями та можливості керування сесіями через cookies.

Однак у роботі з сучасними динамічними сайтами цей фреймворк стикається з рядом обмежень. Зокрема, йому бракує повноцінної підтримки JavaScript-рендерингу, він не має вбудованих механізмів для обходу захисту від ботів [25], вимагає ручного налаштування для кожного окремого сайту, а також обмежений у можливостях масштабованого запуску. У цьому ж контексті варто

згадати альтернативні інструменти, такі як Beautiful Soup, Selenium та Puppeteer, кожен з яких має свої специфічні переваги залежно від типу задачі.

Інший клас рішень представлений хмарними платформами для веб-скрапінгу, серед яких можна виокремити Apify [16], Scrapy Cloud, ScrapingBee [18] та інші подібні сервіси. Їх основною перевагою є надання готової інфраструктури, що дозволяє запускати скрапер без необхідності налаштовувати сервери та середовище виконання. Такі платформи забезпечують автоматичне масштабування, включають проксі-сервери, інструменти моніторингу та планування задач, а також підтримують API для інтеграції з іншими системами. Водночас хмарні рішення мають і недоліки, серед яких варто зазначити високу вартість при роботі з великими обсягами даних, обмежену гнучкість у конфігурації скраперів, залежність від зовнішнього постачальника послуг, а також ризики, пов’язані з конфіденційністю оброблюваної інформації.

Третю категорію складають комерційні агрегатори даних про нерухомість, до яких належать такі компанії, як Zillow Research, PropertyData [14], CoreLogic [15] тощо. Вони надають своїм клієнтам уже підготовлені структуровані набори даних, зручні аналітичні інструменти, історичну інформацію та прогнози, а також доступ до API. Ці сервіси є корисними для користувачів, які прагнуть швидко отримати доступ до якісних даних без необхідності розробки власних скраперів.

Утім, за зручністю приховуються і суттєві обмеження: висока вартість підписки, обмежене географічне покриття, низька частота оновлення деяких даних та обмеженість у вибору джерел і налаштування параметрів збору інформації.

Для системного аналізу зазначених рішень було визначено ключові критерії оцінки, що є критично важливими при побудові ефективної системи збору даних про нерухомість. До них належать: масштабованість, гнучкість, стійкість до блокувань, якість отриманих даних, вартість експлуатації та рівень автоматизації. Порівняння цих характеристик представлено в таблиці 1.1. Це дозволяє обґрунтувати необхідність створення спеціалізованого рішення, яке враховує специфіку збору даних про

нерухомість з широкого спектра джерел і забезпечує кращий баланс між функціональністю, продуктивністю та витратами.

Таблиця 1.1 – Порівняння існуючих рішень

Категорія рішень	**Масштабованість**	**Гнучкість**	**Стійкість до блокувань**	**Якість даних**	**Вартість експлуатації**	**Автоматизація**
Scrapy	Висока	Висока	Низька	Висока	Низька	Середня
Beautiful Soup	Низька	Висока	Низька	Висока	Низька	Низька
Selenium / Puppeteer	Середня	Висока	Висока	Висока	Висока	Низька
Хмарні платформи (Apify, ScrapingBee тощо)	Висока	Середня	Висока	Висока	Висока	Висока
Комерційні агрегатори (Zillow, CoreLogic)	Висока	Низька	Висока	Висока	Дуже висока	Висока

Аналіз існуючих рішень показав, що жодне з них не забезпечує оптимального балансу між масштабованістю, гнучкістю, якістю даних та вартістю експлуатації. Це зумовлює необхідність розробки спеціалізованого рішення, яке враховуватиме специфіку збору даних про нерухомість з різноманітних джерел.

1.3 Особливості та виклики веб-скрапінгу даних про нерухомість

Процес збору даних про нерухомість із веб-ресурсів пов'язаний із низкою складних викликів, які охоплюють як технологічні, так і структурні, юридичні й етичні аспекти. Ці чинники мають бути враховані при проектуванні та реалізації ефективної системи веб-скрапінгу.

До ключових технологічних викликів насамперед належить різноманіття сучасних підходів до веб-розробки. Багато сайтів використовують не лише статичні HTML-сторінки, а й більш складні механізми, зокрема динамічне формування контенту за допомогою JavaScript, архітектури Single Page Application (SPA) на базі таких фреймворків, як React, Angular або Vue.js, а також прогресивні веб-додатки (PWA) і WebSocket-з’єднання для динамічного оновлення даних. Така різноманітність ускладнює процес збору інформації, оскільки вимагає інструментів, здатних не лише обробляти статичний HTML, а й здійснювати повноцінний рендеринг сторінок у браузерному середовищі.

Іншим технологічним бар'єром є захист від автоматизованого сканування, який активно застосовують більшість комерційних веб-ресурсів. Серед типових механізмів можна відзначити CAPTCHA [28] і reCAPTCHA [29], обмеження частоти запитів (rate limiting) [30], блокування за IP-адресою [31], аналіз поведінкових патернів, технології браузерного "відбитка" (fingerprinting), TLS-фінгерпринтинг [19] та перевірки JavaScript-середовища. Подолання цих бар'єрів вимагає впровадження комплексних технічних рішень: ротації IP-адрес, емуляції дій реального користувача та використання сервісів обходу CAPTCHA.

Крім того, викликом є і необхідність забезпечення масштабованості та високої продуктивності системи. Ринок нерухомості постійно генерує значні обсяги нових даних, що потребує здатності системи обробляти велику кількість запитів паралельно, забезпечувати розподіл навантаження та мати механізми відновлення після збоїв.

До структурних викликів належить, зокрема, різноманіття форматів даних, у яких може бути представлена інформація на сайтах. Це можуть бути як структуровані

HTML-фрагменти із семантичною розміткою, так і неструктурований текст, таблиці, списки, об’єкти JavaScript, відповіді у форматі JSON або XML. Це зумовлює необхідність створення гнучких алгоритмів вилучення, розпізнавання і нормалізації даних, адаптованих до різних типів вхідних форматів.

Ще однією значною проблемою є мовна та регіональна варіативність. На глобальному ринку нерухомості існують суттєві відмінності у мові подання інформації, термінології, одиницях виміру (наприклад, квадратні метри, квадратні фути, пінг), форматах дат, чисел і валют, а також у складі й описі характеристик об'єктів. Успішна система збору даних має бути здатною інтерпретувати такі відмінності й трансформувати інформацію у єдиний стандартизований формат.

Окрім технічних і структурних складнощів, важливо враховувати юридичні та етичні обмеження. Використання веб-скрапінгу має відповідати умовам використання веб-сайтів, не порушувати законодавства у сфері захисту персональних даних (зокрема, GDPR у Європі чи CCPA у США), поважати приватність користувачів та авторські права на контент. Крім того, критично важливо дотримуватись принципу мінімізації навантаження на сервери джерел, аби не заважати їхньому функціонуванню.

Таким чином, розробка системи збору даних про нерухомість вимагає не лише глибокого технічного розуміння специфіки веб-розробки, а й уважного врахування юридичних, регіональних та етичних аспектів.

1.4 Постановка задачі

На основі попереднього аналізу предметної області, особливостей веб-скрапінгу та огляду наявних рішень, було сформульовано низку вимог до розроблюваної системи збору даних про нерухомість. Ці вимоги охоплюють як функціональні, так і нефункціональні аспекти, а також обмеження, що зумовлюють архітектурні та реалізаційні рішення.

Серед функціональних вимог передбачено комплексну підтримку усіх етапів життєвого циклу даних — від збору до доступу й моніторингу. Система повинна забезпечувати ефективний збір інформації з різних типів джерел, зокрема сайтів оголошень, агентств нерухомості, агрегаторів тощо. При цьому необхідно враховувати як статичні, так і динамічні веб-сторінки, що формують контент через JavaScript, а також API-інтерфейси [23]. Зібрані дані можуть мати різну структуру — від карток і списків до таблиць або детальних сторінок — і система має бути здатною розпізнавати та обробляти їх коректно.

У частині обробки даних передбачено нормалізацію та структурування інформації відповідно до уніфікованої схеми, яка дозволяє забезпечити цілісність і сумісність між даними, отриманими з різних джерел. Важливою функцією є реалізація валідації згідно з бізнес-правилами, виявлення та фільтрація неповних або некоректних записів, а також обробка дублікатів оголошень, що особливо актуально в умовах повторного розміщення об'єктів на різних платформах.

Щодо зберігання та доступу, система має бути здатною оперувати великими обсягами інформації без втрати продуктивності. Необхідно реалізувати API для отримання зібраних даних зовнішніми системами, а також передбачити можливість експорту в стандартні формати — JSON, CSV, XML — залежно від потреб користувачів.

Управління системою має супроводжуватись інструментами контролю процесів збору, включно з моніторингом продуктивності, виявленням аномалій та формуванням відповідних звітів. Це дозволяє оперативно реагувати на зміни в роботі скраперів або структури веб-сайтів.

Нефункціональні вимоги акцентують увагу на продуктивності, масштабованості, надійності, безпеці та розширюваності системи. Очікується, що система оброблятиме не менше 10 000 сторінок на годину з часом відгуку API не більше 500 мілісекунд. Архітектура має підтримувати горизонтальне масштабування

— тобто можливість розширення потужностей за рахунок додавання нових вузлів. Це важливо для адаптації до зростання кількості джерел та обсягів інформації.

З позиції надійності система повинна гарантувати високий рівень доступності (не менше 99%), бути стійкою до збоїв окремих компонентів і мати механізми відновлення у разі відмови. Щодо безпеки, необхідно передбачити захищене зберігання даних, контроль доступу через автентифікацію та авторизацію, а також дотримання принципів конфіденційності, включно з мінімізацією зібраної персональної інформації.

Гнучкість системи полягає у здатності швидко додавати нові джерела даних, підтримувати різні методи обходу захисту від ботів, а також легко інтегруватися з іншими програмними продуктами.

Окрім вимог, розробка системи обмежується рядом факторів. До технологічних обмежень належить орієнтація на відкриті технології та фреймворки, що забезпечує прозорість, масштабованість і можливість доопрацювання в майбутньому. Юридичні обмеження включають відповідність чинному законодавству щодо збору та обробки даних. Ресурсні обмеження вимагають оптимального використання обчислювальних ресурсів і пропускної здатності мережі, а часові — підтримки актуальності інформації при дотриманні допустимої частоти оновлення.

У сукупності зазначені вимоги формують технічне підґрунтя для побудови комплексної системи веб-скрапінгу, здатної відповідати сучасним викликам і потребам ринку нерухомості.

Таким чином в даному розділі було проведено аналіз предметної області, пов'язаної зі збором даних про нерухомість з веб-ресурсів. Визначено основні джерела даних та їх особливості, структуру інформації про об'єкти нерухомості, а також регіональні відмінності у представленні таких даних. Огляд існуючих рішень для веб-скрапінгу показав, що наявні інструменти та платформи мають суттєві обмеження щодо масштабованості, гнучкості, якості даних та вартості експлуатації при застосуванні до задач збору даних про нерухомість. Це підтверджує актуальність

розробки спеціалізованого рішення. Визначено основні виклики, пов'язані зі збором даних про нерухомість, зокрема технологічні (різноманітність технологій веб-розробки, захист від автоматизованого сканування), структурні (різноманітність форматів даних, мовні та регіональні особливості) та юридичні аспекти.

На основі проведеного аналізу сформульовано функціональні та нефункціональні вимоги до розроблюваної системи, а також визначено обмеження, які необхідно враховувати при її створенні. Це забезпечує чітке розуміння цілей та завдань розробки, а також формує основу для подальшого проектування архітектури системи.

2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ

2.1 Архітектура модуля та системи

Відповідно до вимог, опис функціональності реалізовано у контексті програмного модуля, який в рамках даної системи виконує ключові завдання з автоматизованого збору даних. Була спроектована архітектура системи збору даних про нерухомість, яка орієнтована на забезпечення масштабованості, гнучкості та ефективності. Її побудовано за модульним принципом, що дозволяє ізолювати функціональні компоненти, полегшує супровід і оновлення, а також підтримує паралельну розробку й тестування.

Центральним технологічним елементом архітектури є фреймворк Scrapy, який забезпечує базову функціональність веб-скрапінгу. Його використання дає змогу реалізувати гнучкий та розширюваний підхід до збору інформації, а також інтегрувати додаткові інструменти й сервіси для обробки динамічного контенту та захисту від блокувань.

До складу архітектури входить модуль скраперів (спайдерів) — набір спеціалізованих класів, які реалізують логіку збору даних з конкретних веб-сайтів. Їх організовано за географічним принципом, що полегшує масштабування на різні країни або регіони та врахування локальних специфік.

Система обробки даних базується на використанні ItemLoader, який виконує нормалізацію та структуризацію отриманих даних. Цей компонент інтегрує спеціалізовані процесори, що автоматизують обробку числових значень, дат, адрес та іншої інформації, підвищуючи якість і уніфікованість результатів.

Компоненти аналізу структури даних включають розроблені парсери, зокрема TableParser та AddressParser, які дозволяють адаптувати систему до різноманітних форматів представлення даних — від таблиць до неструктурованих адресних блоків.

Для підтримки динамічного контенту, що генерується JavaScript'ом, передбачено інтеграцію з Playwright [2] — браузерним інструментом для

автоматизованого рендерингу сторінок та взаємодії з DOM. Це забезпечує можливість роботи з сайтами, які не можна обробити традиційними інструментами парсингу.

Структура роботи проекту показана на Рисунку 2.1. Така структура забезпечує:

- ізоляцію компонентів різного призначення;
- можливість паралельної розробки різними командами;
- спрощене тестування окремих модулів;
- легке масштабування шляхом додавання нових компонентів.

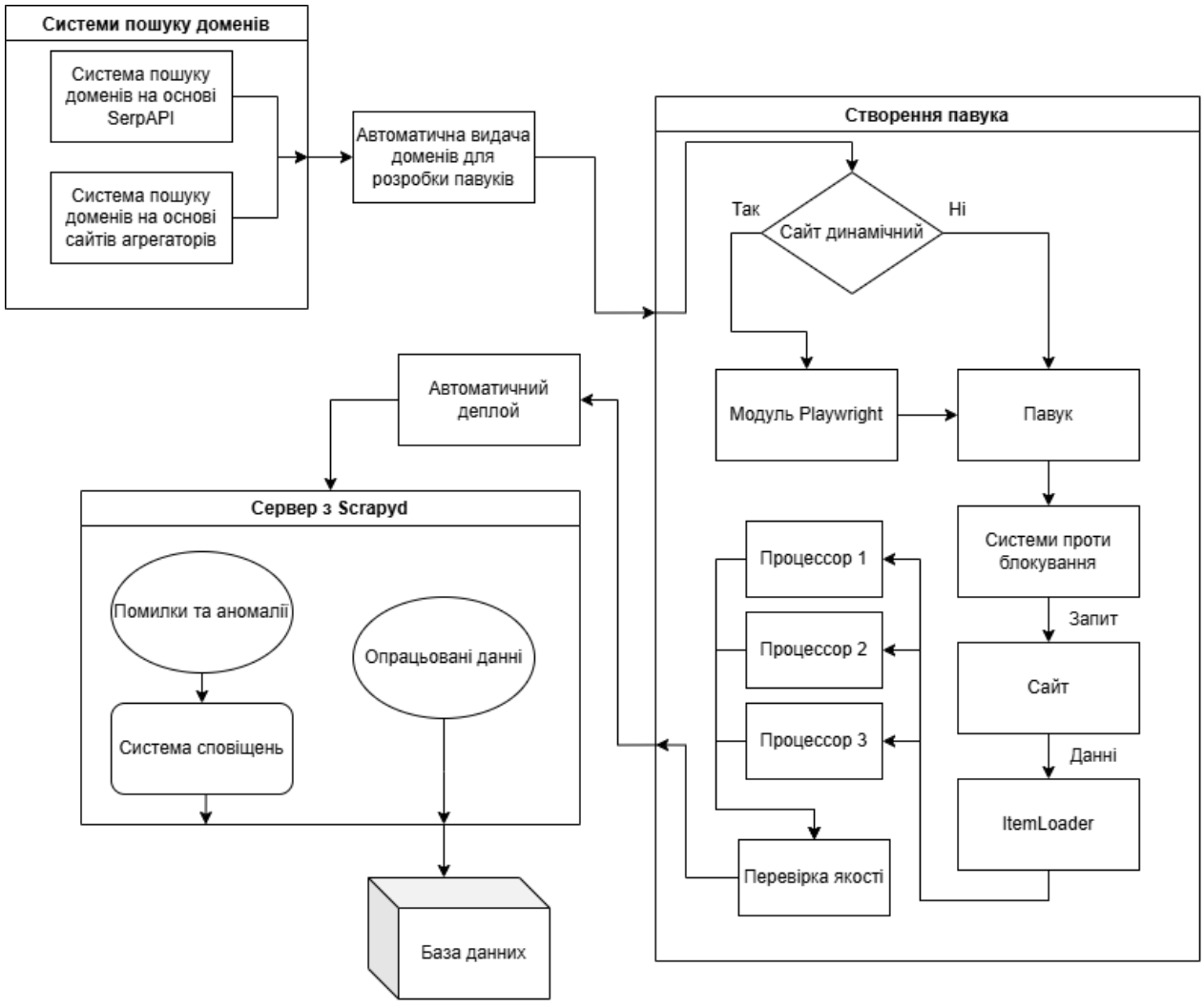


Рисунок 2.1 – Структура проекту

Зібрані дані зберігаються у базі даних PostgreSQL [10], яка також використовується для зберігання логів виконання, технічних метрик та службової інформації про джерела даних.

Кодова база проєкту організована за функціональним принципом, що забезпечує чітке розділення відповідальності між компонентами та спрощує супровід і масштабування системи:

```
OnlineMindsScrapers/
├── itemloader/          # Модуль обробки даних
├── spiders/             # Організовані за країнами скрапери
│   ├── Australia/
│   ├── Belgium/
│   ├── ...
├── localdbs/            # Локальні бази даних для тестування
├── countryboundaries/   # Геодані для валідації адрес
├── ...
scripts/
├── perfomancetracker/   # Системи моніторингу і звітності
├── websiteanalyser/     # Інструменти аналізу нових сайтів
│   └── crawler/
```

Система моніторингу та контролю якості включає набір інструментів для валідації даних, автоматичне виявлення аномалій, контроль відповідності бізнес-правилам і сповіщення через Telegram у разі критичних подій. Це дозволяє оперативно реагувати на помилки та підтримувати високу якість зібраної інформації.

З метою масштабування система включає механізми розширення, що забезпечують автоматичне виявлення нових доменів для скрапінгу, а також інтеграцію з Trello [9] для керування задачами, пов’язаними з підтримкою та розвитком скраперів. Такий підхід сприяє швидкому реагуванню на появу нових джерел даних та централізованому управлінню їх обробкою.

Важливою складовою є система захисту від блокувань, яка включає механізми ротації проксі, обхід CAPTCHA, а також реалізацію TLS-фінгерпринтингу —

технології, що імітує мережеву поведінку реального браузера для обходу обмежень доступу.

Інфраструктура розгортання та запуску реалізована на основі GitHub Actions [8], що забезпечує повний цикл CI/CD [27] (неперервної інтеграції та доставки). Для централізованого управління запуском скраперів використовується Scrapyd [4], який дозволяє планувати та відслідковувати виконання завдань у продакшн-середовищі.

2.2 Основні алгоритми збору та обробки даних

2.2.1 Універсальний ItemLoader та система процесорів

Для забезпечення єдиного підходу до обробки даних з різних джерел був розроблений універсальний ItemLoader, який визначає структуру даних про нерухомість та містить спеціалізовані процесори для нормалізації і валідації інформації.

Основними перевагами такого підходу є:

- уніфікація обробки даних незалежно від джерела;
- стандартизація форматів (дати, числа, валюти);
- централізоване вдосконалення логіки обробки;
- зменшення дублювання коду в окремих скраперах.

Структура ItemLoader включає:

- опис всіх можливих полів для об'єктів нерухомості;
- вхідні (input) процесори для попередньої обробки даних;
- вихідні (output) процесори для фінальної нормалізації;
- правила для об'єднання даних з різних джерел.

Приклад визначених процесорів під поля для збору данних зображено на Рисунку 2.2

```
class OnlineMindsItemLoader(OnlineMindsBaseItemLoader):
    default_item_class = OnlineMindsScrapersItem

    default_output_processor = TakeFirst()

    external_id_in = MapCompose(str, str.strip)

    city_in = MapCompose(str.strip, str.title)
    zipcode_in = MapCompose(str, str.strip)
    address_in = MapCompose(remove_whitespaces, str.strip)
    address_out = Compose(Join(", "), fix_comas, remove_whitespaces, str.strip)
    latitude_in = MapCompose(str, str.strip)
    longitude_in = MapCompose(str, str.strip)
    location_in = MapCompose(str, str.strip)
    location_out = add_location

    title_in = MapCompose(str.strip, remove_whitespaces)
    description_in = Compose(Join("\n"), str.strip, clean_description, str.strip)
    description_out = Join("\n")

    property_type_in = Compose(Join(""), str.strip, check_property_type, str.capitalize)
    available_date_in = MapCompose(format_date)

    room_count_in = MapCompose(take_value)
    bathroom_count_in = MapCompose(take_value)
    floor_in = MapCompose(take_value)
    square_meters_in = MapCompose(process_square_meters, take_value, convert_to_metric)
    year_built_in = MapCompose(take_value)
    energy_rating_in = MapCompose(str, remove_whitespaces, str.strip)
    rental_period_in = MapCompose(str, remove_whitespaces, str.strip)
```

Рисунок 2.2 – Приклад визначених процесорів

Особливу увагу приділено розробці спеціалізованих процесорів, які вирішують типові задачі обробки даних:

1. NumberExtractor — вилучає числові значення з текстових фрагментів, враховуючи різні формати запису та одиниці виміру:
   - Розпізнавання чисел в різних форматах (1 000, 1,000, 1.000).
   - Конвертація одиниць виміру (кв.м. → м², sq.ft → м²).
   - Нормалізація діапазонів значень.
2. PropertyTypeClassifier — визначає тип нерухомості на основі текстового опису:
   - Розпізнавання типу (квартира, будинок, комерційна нерухомість).

- Врахування мовних особливостей різних країн.
- Нормалізація до єдиної класифікації.

3. DateNormalizer — перетворює різні формати дат у стандартизований формат:
    - Обробка регіональних форматів (DD/MM/YYYY, MM/DD/YYYY).
    - Конвертація текстових представлень ("вчора", "минулого тижня").
    - Приведення до ISO 8601 формату.
4. TextCleaner — очищує текстові дані від зайвих символів та форматування:
    - Видалення HTML-тегів.
    - Нормалізація пробілів та переносів рядків.
    - Обробка спеціальних символів та кодувань.

2.2.2 Модуль автоматичної пагінації (AutoPagination)

Для ефективного збору даних з багатосторінкових ресурсів розроблено механізм AutoPagination, який автоматично визначає тип пагінації та забезпечує послідовний обхід всіх сторінок з результатами.

Модуль AutoPagination вирішує наступні задачі:

- автоматичне визначення типу пагінації на сайті;
- формування URL для наступних сторінок;
- відстеження прогресу обходу та запобігання зациклюванню;
- адаптація до змін у структурі сайту.

Система підтримує різні типи пагінації:

1. URL-based пагінація — використовує параметри URL для навігації:
    - https://example.com/properties?page=1
    - https://example.com/properties?page=2
2. Path-based пагінація — використовує шляхи URL:
    - https://example.com/properties/page/1
    - https://example.com/properties/page/2

3. JavaScript-based пагінація — для сайтів, що використовують AJAX [20] або інші JS-механізми:

- Імітація кліків на елементи навігації
- Відстеження змін у DOM після взаємодії
- Обробка нескінченного скролінгу (infinite scroll)

Фрагмент коду AutoPaginate зображено на Рисунку 2.3

```
def _parse_format_url(self, response):
    if self._stop_condition(response) and not self.stop_after:
        return

    yield from self._parse_card(response)

    if self._stop_condition(response) and self.stop_after:
        return

    self.index += self.index_increment
    new_urls = [self.format_url]
    new_payload = self.format_payload

    if not self.format_payload:
        if self.format_url:
            new_urls = [new_urls[0].format(self.index)]
        else:
            new_urls = self._find_new_urls(response)
    else:
        new_payload = new_payload.format(self.index)

    for url in new_urls:
        yield response.follow(
            url=url,
            headers=self.format_headers,
            body=new_payload,
            method=self.format_method,
            callback=self._parse_format_url,
        )
```

Рисунок 2.3 – Фрагмент коду AutoPaginate

2.2.3 Модуль аналізу табличних структур (TableParser)

Для ефективної обробки табличних даних, які часто використовуються на сайтах нерухомості для відображення характеристик об'єктів, розроблено спеціалізований TableParser (Додаток А). Цей компонент дозволяє автоматично

розпізнавати структуру таблиць та співвідносити заголовки з відповідними значеннями.

Основні функції TableParser:

– Ідентифікація табличних структур у HTML-документі.
– Аналіз семантики заголовків та співставлення з атрибутами нерухомості.
– Нормалізація даних з урахуванням контексту таблиці.
– Обробка складних випадків (об'єднані комірки, неявні заголовки).

Алгоритм роботи TableParser:

1. Виявлення табличних структур у DOM (як стандартних `<table>`, так і CSS-таблиць).
2. Аналіз заголовків та співставлення з відомими атрибутами нерухомості через словник ключових слів.
3. Розпізнавання форматів даних у комірках.
4. Нормалізація та структуризація отриманих даних.

Особливістю розробленого TableParser є здатність працювати з неструктурованими та нестандартними таблицями, які часто зустрічаються на сайтах нерухомості.

2.2.4 Модуль аналізу адрес (AddressParser)

Одним з найважливіших атрибутів об'єктів нерухомості є адреса, яка часто представлена в нестандартизованому форматі. Для вирішення цієї проблеми розроблено спеціалізований AddressParser (Додаток Б), який забезпечує аналіз та структуризацію адресної інформації.

Основні функції AddressParser:

– Виділення компонентів адреси (країна, місто, вулиця, будинок).
– Нормалізація скорочень та спеціальних позначень.
– Геокодування для отримання координат.
– Валідація адреси з використанням геопросторових даних.

AddressParser використовує комбінацію підходів:

1. Регулярні вирази для розпізнавання типових шаблонів адрес.
2. Словники географічних назв для ідентифікації компонентів.
3. Машинне навчання для складних випадків.
4. Інтеграцію з геокодинговими API для валідації та доповнення даних.

Особливістю системи є підтримка мультимовності та врахування регіональних особливостей форматування адрес.

Система зберігає словники географічних даних у модулі countryboundaries, що дозволяє проводити валідацію адрес навіть без доступу до зовнішніх API.

2.2.5 Модуль захисту від блокувань

Для запобігання блокуванню запитів з боку веб-сайтів у системі реалізовано комплексну архітектуру захисту, яка включає ротацію проксі, механізми обходу CAPTCHA та емуляцію параметрів з'єднання, що притаманні реальним браузерам. Таке багаторівневе рішення дозволяє ефективно взаємодіяти навіть із сайтами, що впроваджують складні системи протидії автоматизованому збору даних.

Система ротації проксі-серверів функціонує за трирівневою схемою. Перший рівень передбачає використання прямих запитів без проксі для некритичних джерел, де блокування малоймовірне або не має серйозних наслідків. У разі виникнення помилок типу 403 або 429 система переходить на другий рівень — використання звичайних проксі для базового маскування IP-адреси. Якщо ж і на цьому рівні продовжуються блокування, активується третій рівень — використання residential проксі, які імітують справжніх користувачів з домашніми підключеннями. Для кожного етапу передбачено до трьох спроб, після чого фіксується результат для подальшої оптимізації вибору стратегії. Такий підхід дозволяє економно використовувати ресурси, зокрема уникаючи надмірного навантаження на дорогі residential-проксі.

Ще одним поширеним засобом захисту на сайтах є CAPTCHA. Для її обходу в системі реалізовано автоматичне виявлення елементів CAPTCHA шляхом аналізу структури DOM. Після ідентифікації виконується інтеграція з сервісом 2Captcha [7], що забезпечує розпізнавання CAPTCHA людиною в реальному часі. Процес включає передачу ключових параметрів (зокрема sitekey у випадку reCAPTCHA) на сторонній сервіс, очікування відповіді та ін’єкцію отриманого розв’язку у відповідну форму на сторінці. З метою мінімізації витрат реалізовано кешування сесій, що знижує частоту звернень до ресурсу з CAPTCHA, а також зменшує кількість запитів у межах однієї сесії.

Для подолання систем виявлення автоматичних запитів, що базуються на TLS-фінгерпринтингу, впроваджено механізм емуляції TLS-з'єднання відповідно до параметрів, притаманних реальним браузерам. Основу цього підходу становить емуляція JA3-фінгерпринту — унікального відбитка, який формується на основі параметрів TLS-рукостискання. Система автоматично модифікує параметри з'єднання та динамічно підміняє HTTP-заголовки так, щоб вони відповідали цим параметрам. Механізм активується лише у випадках, коли зафіксовано блокування, що, імовірно, спричинене TLS-аналізом, зокрема помилки 403 при відсутності інших ознак.

Завдяки реалізованому підходу система забезпечує стійкість до сучасних механізмів захисту та може масштабовано працювати з великою кількістю джерел навіть за наявності складних бар'єрів на шляху автоматизованого збору даних.

2.2.6 Автоматизація життєвого циклу скраперів

Для забезпечення безперервної розробки, розгортання та підтримки системи збору даних про нерухомість було впроваджено автоматизований життєвий цикл скраперів. Цей підхід дозволяє суттєво скоротити ручні операції, мінімізувати помилки, а також забезпечити оперативне реагування на зміни у структурі веб-сайтів або вимогах до даних.

Основу життєвого циклу скраперів становить система CI/CD, побудована на платформі GitHub Actions. Вона охоплює всі ключові етапи — від тестування до розгортання та інтеграції з моніторинговою системою. Першим етапом є тестування нових або оновлених скраперів. Для цього виконується запуск з тестовими параметрами, перевіряється структура та якість отриманих даних, а також формується звіт про помилки та попередження. На другому етапі здійснюється валідація даних із залученням інтеграції з ChatGPT [5] для аналізу якості парсингу, порівняння отриманих результатів із еталонними прикладами та оцінки повноти вилученої інформації. Завершальним етапом є автоматичне розгортання скрапера: він публікується на сервері за допомогою Scrapyd API, оновлюються конфігурації системи запуску, а також активується зв’язок із моніторинговими компонентами. Для оптимізації процесу тестування передбачена можливість запуску CI/CD-процесу шляхом встановлення спеціальних міток у Pull Request.

Для управління виконанням скраперів реалізовано інтеграцію з Scrapyd — інструментом для розгортання та контролю за запуском Scrapy-проєктів. Система автоматично надсилає зібраний egg-пакет скрапера на Scrapyd-сервер, де він розгортається із заданими параметрами. Надалі система виконує планування запусків з урахуванням пріоритетності джерел, відслідковує статуси виконання, обробляє результати та формує відповідні логи. Такий підхід забезпечує прозорість процесів та полегшує виявлення помилок або збоїв у роботі окремих компонентів.

Для організації регулярного збору даних була реалізована система планування запусків на основі cron-механізму [33]. Розклад враховує як глобальні потреби, так і специфіку окремих регіонів. Зокрема, існує загальний графік, за яким усі активні скрапери запускаються тричі на добу, що забезпечує актуальність зібраної інформації. Крім того, для окремих країн або джерел з високою динамікою змін розроблено спеціалізовані розклади, які враховують місцевий час і частоту оновлень. Також реалізовано механізми балансування навантаження, що дозволяють уникати пікових перевантажень системи та рівномірно розподіляти ресурси.

Завдяки поєднанню автоматизованого тестування, CI/CD-процесів, централізованого управління запусками та адаптивного планування, система забезпечує стабільну та масштабовану роботу великої кількості скраперів у режимі реального часу.

2.2.7 Механізми розширення модулів

З метою забезпечення масштабованості системи збору даних про нерухомість та швидкого реагування на появу нових джерел інформації, у рамках архітектури було реалізовано механізми автоматичного розширення. Вони дозволяють виявляти перспективні ресурси, здійснювати їх первинну оцінку, інтегрувати в загальну базу та створювати задачі для подальшої розробки відповідних скраперів.

Ключовим компонентом є система автоматичного додавання нових доменів. Її робота базується на регулярному пошуку нових сайтів через SerpAPI [6] — інтерфейс для взаємодії з пошуковими системами. Пошук здійснюється за ключовими словами, що стосуються нерухомості, з урахуванням регіональних фільтрів для визначення потенційно релевантних ресурсів у різних країнах. Додатково використовується скрапінг агрегаторів, які містять переліки агентств або платформ, що працюють у відповідній галузі.

Після виявлення нових доменів система виконує їх первинний аналіз — визначає географічну приналежність, тип ресурсу (агрегатор, агентство, портал оголошень), а також оцінює орієнтовний обсяг доступної інформації. Зібрані метадані зберігаються у базі даних PostgreSQL, де надалі формуються рейтинги та пріоритети обробки. Відповідно до визначених критеріїв автоматично створюються задачі на розробку нових скраперів.

Для управління цим процесом використовується інтеграція з Trello — системою візуального управління проєктами. Кожен новий домен, що пройшов попередню оцінку, автоматично отримує окрему картку в Trello, де зазначаються всі необхідні параметри: країна, релевантність, тип контенту та рівень пріоритету. Це дозволяє

систематизувати задачі, розподіляти їх між учасниками команди відповідно до географічного чи технічного профілю, а також відслідковувати етапи виконання.

Крім автоматичного створення, картки в Trello оновлюються на основі результатів тестування скраперів, що інтегровані у CI/CD-процес. Це забезпечує повний зворотній зв'язок між технічною реалізацією та процесом управління задачами, дозволяючи команді оперативно реагувати на зміни та підтримувати актуальний стан усіх розроблюваних компонентів.

Завдяки впровадженню механізмів розширення система зберігає здатність адаптуватися до змін в екосистемі веб-ресурсів і оперативно нарощувати обсяг охоплених джерел без значного ручного втручання, що є критично важливим для збереження конкурентоспроможності та актуальності даних.

2.3 Інформаційне забезпечення модулів

2.3.1 Робота з динамічним контентом

У сучасних веб-застосунках, зокрема на платформах з оголошеннями про нерухомість, значна частина контенту формується динамічно за допомогою JavaScript, AJAX або нескінченного скролінгу. Це створює значні виклики для класичних інструментів веб-скрапінгу, які працюють із попередньо згенерованим HTML. Для ефективної роботи з такими ресурсами в межах системи були реалізовані спеціалізовані механізми.

Для обробки сторінок, що потребують виконання JavaScript, у систему інтегровано інструмент Playwright — сучасну платформу автоматизації браузера, яка дозволяє виконувати повноцінний рендеринг сторінки та взаємодіяти з її елементами. Інтеграція реалізована у вигляді middleware, який активується лише для тих сайтів, де контент безпосередньо недоступний через вихідний HTML.

Основні функції Playwright-інтеграції включають:

– рендеринг сторінок з JavaScript-контентом у headless-режимі;

– емулювання дій користувача, таких як прокручування, натискання кнопок, заповнення форм;

– аналіз стану DOM після завершення завантаження контенту;

– ін’єкцію власних скриптів для вирішення нестандартних задач доступу до даних.

Процес взаємодії з Playwright починається з визначення, чи потребує конкретний сайт використання браузера. Якщо так — запускається відповідне середовище, завантажується сторінка, виконуються потрібні дії, після чого з DOM вилучаються структуровані дані. Особливістю реалізації є автоматичне включення цього механізму лише у разі потреби, що дозволяє суттєво зменшити навантаження на систему та зекономити ресурси.

У багатьох випадках сайти використовують публічні або частково приховані API для отримання даних про об’єкти нерухомості. У системі реалізовано підхід, що базується на перехопленні та аналізі мережевих запитів, які браузер надсилає до серверів під час роботи сайту.

На першому етапі здійснюється дослідження запитів за допомогою інструментів розробника, зокрема аналізується вкладка Network у браузері. Після ідентифікації ключових запитів, які повертають структуровані дані (часто у форматі JSON), виконується реверс-інжиніринг їх параметрів та механізмів автентифікації. Отримана інформація дозволяє сконструювати аналогічні запити у рамках скрапера, що значно спрощує доступ до даних.

У випадках, коли API захищене, застосовуються додаткові методи: емулювання заголовків та cookies, підтримка токенів чи сесій, генерація підписів для валідації запитів, а також управління частотою запитів для уникнення блокування.

Окрему категорію становлять сайти, де дані підвантажуються динамічно під час прокручування сторінки. Такі механізми, як нескінченний скролінг або довантаження через AJAX, вимагають спеціального підходу до збору інформації.

У системі реалізовано кілька методів для обробки таких випадків:

– автоматичний скролінг сторінки до її кінця із симуляцією поведінки користувача для активації завантаження нових даних;

– перехоплення XHR-запитів, що виконуються під час скролінгу, та їх аналіз для подальшої автоматизації;

– ін’єкція JavaScript-скриптів, які модифікують поведінку сторінки або забезпечують доступ до прихованих структур.

Система самостійно визначає момент завершення довантаження, що дозволяє уникнути надлишкового використання ресурсів і забезпечити повний збір інформації.

Загалом, поєднання наведених механізмів дозволяє системі ефективно працювати з сучасними, технологічно складними веб-ресурсами й забезпечує високу якість отриманих даних навіть у випадках нестандартних рішень на стороні сайту.

2.3.2 Механізми високочастотного збору даних

Для збору даних з великих і динамічних веб-ресурсів, на яких інформація оновлюється з високою частотою, у системі було реалізовано спеціалізований компонент HighFrequencyRunner. Його мета — забезпечити часті цикли збору нових даних при одночасному зниженні навантаження на веб-сервери джерел і збереженні обчислювальної ефективності.

HighFrequencyRunner виконує запуск скраперів з інтервалом у 10 хвилин, що дозволяє оперативно фіксувати появу нових оголошень на сайтах. Для уникнення дублювання та зменшення обсягу оброблюваної інформації застосовується механізм диференціації даних. Він базується на використанні бази даних PostgreSQL, де зберігається історія вже оброблених URL-адрес разом із відповідними часовими мітками.

Алгоритм роботи компонента складається з кількох послідовних етапів. Спершу система отримує список актуальних посилань на об’єкти нерухомості з головної сторінки джерела. Далі виконується порівняння кожного посилання з базою вже оброблених, і до наступного етапу допускаються лише ті, що є новими або мають

ознаки оновлення. Після цього відбувається цільовий парсинг лише нових або змінених оголошень, після чого результати записуються у базу даних із фіксацією часу обробки.

Такий підхід дає змогу досягати майже реального часу збору даних, що особливо актуально для джерел з високою конкуренцією серед користувачів або коротким життєвим циклом оголошень. Завдяки обмеженню обсягу парсингу лише до нових записів, система не створює надмірного навантаження на цільові сайти, що важливо як з технічної, так і з етичної точки зору. HighFrequencyRunner ефективно поєднує точність, швидкість та оптимізацію ресурсів, доповнюючи загальну архітектуру системи збору даних.

2.3.3 Модуль зберігання та моніторингу даних

Для забезпечення надійного зберігання, моніторингу та контролю якості даних у системі збору інформації про нерухомість реалізовано комплексну інфраструктуру, що поєднує високопродуктивне сховище, систему моніторингу й механізми валідації. Основною метою цієї архітектури є забезпечення цілісності, доступності й аналітичної придатності зібраної інформації.

У якості центрального сховища використовується база даних PostgreSQL, яка забезпечує масштабованість, продуктивність та надійність роботи з великими обсягами структурованих даних. Система зберігання охоплює кілька спеціалізованих таблиць. Основна таблиця Properties містить усю інформацію про об’єкти нерухомості, тоді як таблиця SpiderRuns відображає параметри та результати запусків скраперів, включно з технічними метаданими. Таблиця Errors використовується для зберігання детальної інформації про виявлені помилки та збої, а Domains містить перелік оброблюваних доменів з відповідними описовими атрибутами. Для оптимізації доступу до даних використовуються індекси, що пришвидшують фільтрацію та пошук. Взаємодія з базою даних реалізована через ORM-бібліотеку

peewee [3], яка абстрагує роботу з SQL-запитами, надаючи гнучкий і зручний інтерфейс для операцій із даними.

Паралельно функціонує система моніторингу та алертингу, що дозволяє відстежувати стабільність і ефективність роботи всіх компонентів системи. До її функцій належить збір метрик запусків, зокрема кількості оброблених URL, знайдених об'єктів і виявлених помилок. Аналізується доступність джерел даних і тривалість виконання задач. У випадку відхилень від типових показників або появи ознак нестабільності, система автоматично формує сповіщення. Сповіщення надсилаються через інтеграцію з Telegram, що забезпечує миттєве інформування відповідальних осіб. Крім того, система щоденно генерує узагальнені звіти про стан скраперів і продуктивність системи в цілому. У разі виявлення критичних аномалій також автоматично створюються задачі у Trello, що дозволяє швидко розподіляти відповідальність і фіксувати проблеми в робочому процесі команди.

Механізм виявлення аномалій ґрунтується на аналізі історичних даних і статистичних порогах. Для кожного скрапера визначаються типові значення основних метрик, на основі яких розраховуються граничні значення. У випадку перевищення цих порогів формується відповідне попередження або сповіщення.

Для підтримання високої якості зібраних даних розроблено систему багаторівневої валідації. Вона включає перевірку відповідності даних бізнес-правилам (наприклад, допустимі діапазони цін або площ), геопросторову валідацію адрес за допомогою географічних даних, контроль повноти та цілісності, а також виявлення аномалій або викидів, які можуть свідчити про помилки у парсингу. У межах системи також реалізовано механізм трешхолдів: для кожного типу помилок задаються граничні значення, після перевищення яких скрапер автоматично маркується як нестабільний. Це дозволяє пріоритезувати задачі підтримки й покращення системи, фокусуючись на найбільш критичних джерелах або процесах.

Валідація є інтегрованою частиною загального пайплайну обробки даних, що дає змогу оперативно виявляти та усувати помилки ще на ранніх етапах, запобігаючи

накопиченню неточної чи неповної інформації в базі. Такий підхід забезпечує не лише надійне функціонування системи, але й високу якість кінцевих даних для подальшого аналізу чи використання у сторонніх сервісах.

Таким чином, у даному розділі було детально розглянуто архітектуру та основні компоненти розробленої системи автоматизованого збору даних про нерухомість. Представлена система відповідає всім функціональним та нефункціональним вимогам, визначеним у першому розділі. Виділено основні ключові особливості, як: модульна архітектура, спеціалізовані компоненти для обробки різних типів даних та структур, механізми роботи з динамічним контентом через інтеграцію з Playwright, багаторівнева система захисту від блокувань, повна автоматизація життєвого циклу скраперів через CI/CD та інтеграцію з Scrapyd, ефективне зберігання та моніторинг даних з використанням PostgreSQL, система автоматичного розширення через пошук та додавання нових джерел.

Розроблена архітектура системи дозволяє легко масштабувати її на сотні країн та тисячі веб-ресурсів, забезпечуючи високу якість та актуальність зібраних даних при оптимальному використанні обчислювальних ресурсів.

3 ПРОГРАМНО-ТЕХНОЛОГІЧНЕ ЗАБЕЗПЕЧЕННЯ МОДУЛЯ

3.1 Розгортання середовища

Для забезпечення стабільної та ефективної роботи модулів та відповідно і самої системи збору даних про нерухомість було розгорнуто повноцінне інфраструктурне середовище. Його архітектура побудована з урахуванням вимог до масштабованості, надійності, продуктивності та автоматизації процесів розгортання та моніторингу.

Основу інфраструктури складають кілька спеціалізованих компонентів. Сервер із встановленим Scrapyd виконує роль платформи для запуску та управління скраперами. База даних PostgreSQL використовується як основне сховище зібраної інформації та службових даних. Окремий сервер моніторингу відповідає за збір метрик продуктивності системи, стану джерел і контроль за аномаліями. Розробка, тестування та автоматичне розгортання скраперів здійснюється через GitHub-репозиторій із налаштованими GitHub Actions, що забезпечують повний CI/CD-процес.

Центральною частиною інфраструктури є Scrapyd — сервер для розгортання та запуску Scrapy-проєктів. Його конфігурація виконується через файл scrapy.cfg, де задаються параметри доступу, розміщення пакетів та назви проєктів. Для зручності адміністрування передбачено набір HTTP endpoint'ів, що дозволяють у програмний спосіб керувати запусками, переглядати стан завдань або структуру проєктів. Зокрема, доступні такі інтерфейси, як /listprojects, /listspiders, /listjobs та /schedule.json.

Для забезпечення збереження та швидкого доступу до великих обсягів даних, PostgreSQL налаштовано відповідно до вимог високої навантаженості. Була збільшена пам’ять кешування за рахунок параметра shared_buffers, налаштовано автоматичне вакуумування таблиць для запобігання фрагментації та зниження продуктивності. Важливу роль відіграє система індексації, яка забезпечує швидкий пошук та фільтрацію інформації. Крім цього, впроваджено регулярне резервне копіювання з використанням cron-завдань і спеціалізованих скриптів.

Схема бази даних побудована на основі чіткої структуризації основних функціональних блоків. Схема таблиць бази данних показана на рисунку 3.1.

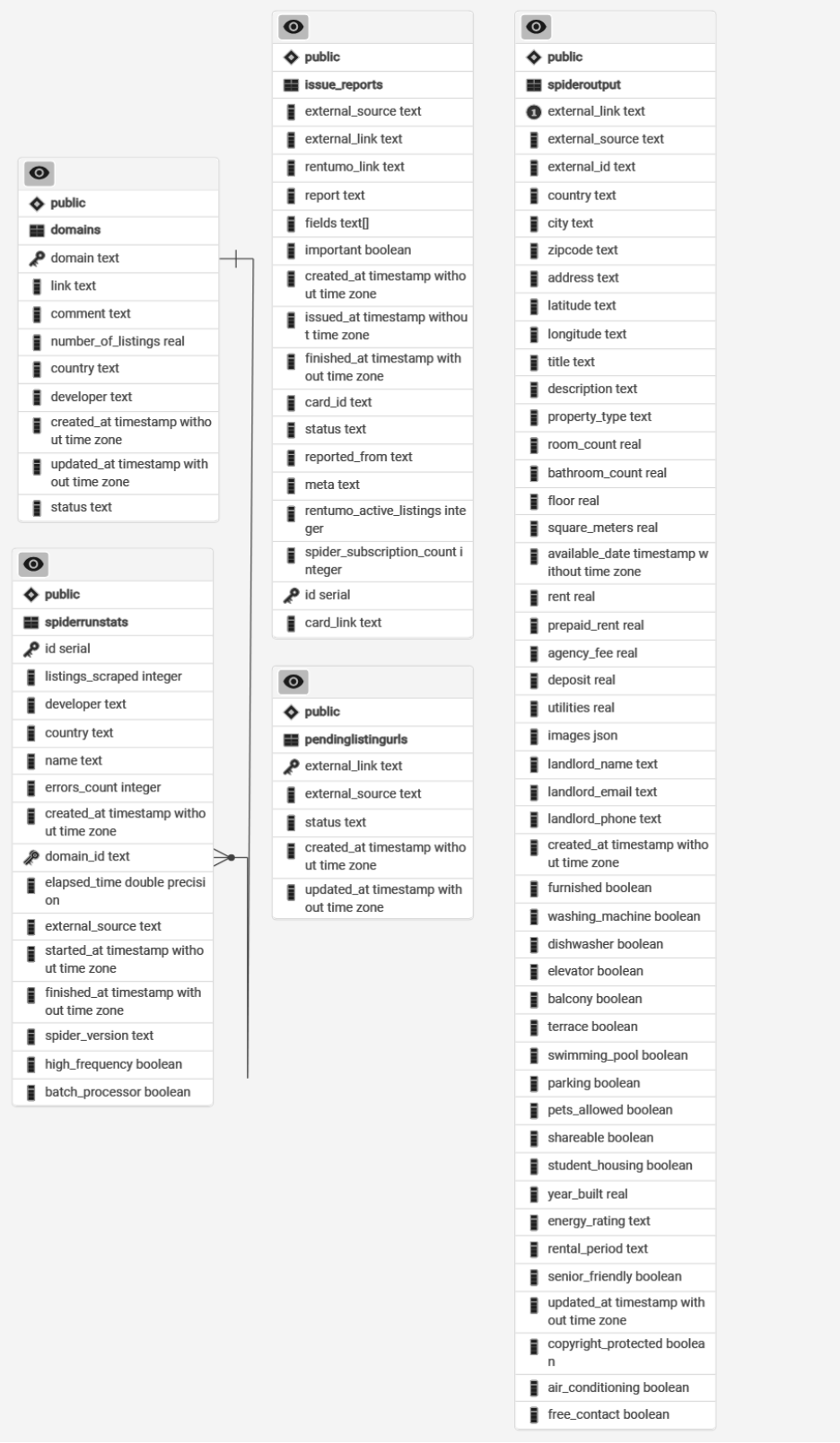


Рисунок 3.1 – Схема бази данних

Основна таблиця SpiderOutput містить інформацію про зібрані об’єкти нерухомості. Таблиця SpiderRunStats зберігає дані про виконання окремих запусків скраперів, включаючи технічні параметри, тривалість та результати. Виявлені помилки фіксуються у таблиці issue_reports, що дозволяє здійснювати їх подальший аналіз та автоматизовану діагностику. Нарешті, таблиця domains містить інформацію про оброблювані домени з відповідними метаданими, що дозволяє управляти джерелами скрапінгу централізовано.

Загальна архітектура розгортання дозволяє легко масштабувати систему, додавати нові компоненти або змінювати конфігурацію окремих частин без втрати стабільності або доступності сервісів.

3.2 Реалізація модулів

У процесі реалізації системи збору даних про нерухомість особливу увагу було приділено створенню модульної архітектури з чітко визначеними компонентами для обробки, структуризації та фільтрації інформації. Усі ключові модулі розміщені в проєктному каталозі OnlineMindsScrapers та відповідають за окремі функціональні блоки, що разом забезпечують узгоджену та масштабовану роботу скраперів.

Для уніфікованої обробки даних реалізовано універсальний PropertyItemLoader у каталозі itemloader. Він базується на стандартному ItemLoader з бібліотеки Scrapy, але доповнений доменно-специфічною логікою. Через механізм процесорів до нього підключено модулі для обробки числових значень (NumberExtractor), дат (DateNormalizer) та адрес (AddressProcessor). Ці процесори дозволяють автоматично нормалізувати формат вхідних даних, незалежно від їх джерела. Наприклад, числові поля очищуються від роздільників тисяч і конвертуються в єдину систему одиниць, дати уніфікуються до ISO-формату, а адреси розбиваються на структуровані компоненти з урахуванням локальних скорочень. Приклад коду підключення процесорів в ItemLoader зображено на рисунку 3.2

Модуль автоматичної пагінації реалізований у вигляді базового класу AutoPaginator, розміщеного в модулі __init__.py у каталозі spiders. Цей компонент відповідає за автоматичне виявлення та обхід сторінок із результатами на веб-сайтах. В залежності від структури ресурсу, він обирає один із трьох підходів: URLPaginator — для класичної параметризованої пагінації через URL; ButtonPaginator — для навігації за допомогою кнопок; InfiniteScrollPaginator — для роботи зі сторінками з нескінченним скролінгом. Тип пагінації визначається автоматично на основі аналізу HTML-структури.

```
class OnlineMindsItemLoader(OnlineMindsBaseItemLoader):
    default_item_class = OnlineMindsScrapersItem

    default_output_processor = TakeFirst()

    external_id_in = MapCompose(str, str.strip)

    city_in = MapCompose(str.strip, str.title)
    zipcode_in = MapCompose(str, str.strip)
    address_in = MapCompose(remove_whitespaces, str.strip)
    address_out = Compose(Join(", "), fix_comas, remove_whitespaces, str.strip)
    latitude_in = MapCompose(str, str.strip)
    longitude_in = MapCompose(str, str.strip)
    location_in = MapCompose(str, str.strip)
    location_out = add_location

    title_in = MapCompose(str.strip, remove_whitespaces)
    description_in = Compose(Join("\n"), str.strip, clean_description, str.strip)
    description_out = Join("\n")

    property_type_in = Compose(Join(""), str.strip, check_property_type, str.capitalize)
    available_date_in = MapCompose(format_date)
```

Рисунок 3.2 – Приклад підключення процесорів в ItemLoader

Модуль TableParser, що розміщується у файлі processors.py, реалізує логіку вилучення структурованих даних з таблиць. Його алгоритм включає ідентифікацію HTML-таблиць або CSS-структур, аналіз заголовків, зіставлення їх з атрибутами нерухомості, а також вилучення і нормалізацію даних із комірок. Особливістю

реалізації є здатність адаптуватися до нестандартних макетів таблиць, у яких заголовки можуть бути неявними або розташованими поза межами основного блоку.

Для обробки динамічного контенту, що завантажується JavaScript-ом, реалізовано middleware для Playwright у модулі middlewares.py. Він автоматично активується у випадках, коли стандартний парсинг HTML не дозволяє отримати необхідні дані. Middleware ініціює запуск headless-браузера, виконує рендеринг сторінки, обробляє DOM та передає вміст назад у скрапер. Визначення необхідності використання Playwright здійснюється на основі попередньої історії обробки сайту або структури URL. Приклад використання Playwright в павуку показано на рисунку 3.3

```
custom_settings = {
    "LOG_LEVEL": "INFO",
    "HTTPCACHE_ENABLED": False,
    "DOWNLOAD_HANDLERS": {
        "http": "scrapy_playwright.handler.ScrapyPlaywrightDownloadHandler",
        "https": "scrapy_playwright.handler.ScrapyPlaywrightDownloadHandler",
    },
    "PLAYWRIGHT_LAUNCH_OPTIONS": {"timeout": 20 * 1000},
}

def start_requests(self):
    yield scrapy.Request(
        url=self.start_url,
        callback=self.parse,
        meta=dict(
            playwright=True,
            playwright_page_methods=[
                PageMethod("wait_for_timeout", 10000),
            ],
        ),
    )
```

Рисунок 3.3 – Використання Playwright в павуку

Модуль аналізу адрес реалізований у вигляді класу AddressParser. Він дозволяє виділити ключові компоненти адреси (країна, місто, вулиця, будинок), нормалізувати

скорочення, а також здійснювати геокодування. Для останнього використовуються локальні дані з модуля countryboundaries, що забезпечує офлайн-перевірку відповідності адрес без обов'язкової інтеграції з зовнішніми API. Завдяки поєднанню регулярних виразів, словників та елементів машинного навчання, модуль здатен обробляти широкий спектр адресних форматів.

Для збору даних із високою частотою з джерел, які оновлюються щодня або щогодини, реалізовано компонент HighFrequencyRunner. Його основна логіка міститься у файлі runspiders.py, де визначено функцію для періодичного запуску скраперів через cron. Головною особливістю є фільтрація URL: перед обробкою кожне посилання порівнюється з базою даних, і в роботу надходять лише нові або змінені об'єкти. Це дозволяє суттєво зменшити навантаження на джерела та базу даних, забезпечуючи при цьому актуальність інформації. Приклад конфігурації crontab для запуску павуків та HighFrequency зображено на Рисунку 3.4

```
  GNU nano 6.2                                  /tmp/crontab.zhQ9sb/crontab
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
30 0 * * * /sbin/reboot
0 6 * * * cd /root/app && ./env/bin/python3 scripts/new_spiders_notifications_sender.py
0 6 * * * cd /root/app && ./env/bin/python3 scripts/spider_fixed_notifications_sender.py
0 6 * * * cd /root/app && ./env/bin/python3 -m scripts.perfomance_tracker.save
*/5 * * * * cd /root/app && ./env/bin/python3 scripts/issue_reporter.py >> /root/issue_reporter.log 2>&1
*/10 * * * * cd /root/app && ./env/bin/python3 run_spiders.py highfrequency # Run all high frequency spiders
0 1 * * * cd /root/app/scripts/website_analyser && ./venv/bin/python3 main.py --model=gpt >> ./w_cronlogS.txt 2>&1 #


0 8 * * * cd /root/app && ./env/bin/python3 run_spiders.py denmark > ./cronlogS.txt 2>&1 # SpiderRunner_denmark_10

0 13 * * * cd /root/app && ./env/bin/python3 run_spiders.py denmark > ./cronlogS.txt 2>&1 # SpiderRunner_denmark_15

0 18 * * * cd /root/app && ./env/bin/python3 run_spiders.py denmark > ./cronlogS.txt 2>&1 # SpiderRunner_denmark_20

0 8 * * * cd /root/app && ./env/bin/python3 run_spiders.py sweden > ./cronlogS.txt 2>&1 # SpiderRunner_sweden_10

0 13 * * * cd /root/app && ./env/bin/python3 run_spiders.py sweden > ./cronlogS.txt 2>&1 # SpiderRunner_sweden_15

0 18 * * * cd /root/app && ./env/bin/python3 run_spiders.py sweden > ./cronlogS.txt 2>&1 # SpiderRunner_sweden_20

0 8 * * * cd /root/app && ./env/bin/python3 run_spiders.py spain > ./cronlogS.txt 2>&1 # SpiderRunner_spain_10
```

Рисунок 3.4 – Crontab з запуском павуків та HighFrequency

Загальна структура модулів побудована таким чином, щоб кожен компонент можна було тестувати, повторно використовувати та масштабувати незалежно від

інших. Такий підхід дозволяє легко адаптувати систему до змін у форматі веб-сайтів, додавати нові функції або джерела без порушення вже наявної логіки.

3.3 Тестування та валідація роботи системи

3.3.1 Тестування логіки

У рамках забезпечення надійності та стабільності функціонування системи збору даних особливу увагу було приділено тестуванню логіки роботи її основних компонентів. Реалізовані механізми охоплюють як інтеграційне тестування скраперів у реальному середовищі, так і оцінку продуктивності всієї системи на рівні навантаження та використання ресурсів.

Інтеграційне тестування реалізоване через GitHub Actions і активується автоматично при створенні Pull Request із міткою "spider testing". У межах цього процесу скрапер запускається на заздалегідь визначеному наборі тестових даних або веб-сторінках, після чого результати збереження фіксуються у вигляді HTML та JSON-файлів. Далі ці результати передаються на аналіз, де оцінюється якість парсингу, зокрема відповідність очікуваній структурі, повнота вилучених даних, коректність нормалізації.

Однією з особливостей тестування є використання інтеграції з ChatGPT, яка дозволяє проводити автоматизовану перевірку змісту JSON-результатів порівняно з HTML-вихідними даними. Такий підхід значно підвищує точність і автоматизацію аналізу, знижуючи потребу в ручній перевірці. За підсумками тестування автоматично формується звіт, який додається до Pull Request.

Для оцінки ефективності роботи системи в умовах реального навантаження реалізовано механізми тестування продуктивності. Ці тести дозволяють вимірювати ключові характеристики, зокрема швидкість обробки сторінок, споживання CPU та оперативної пам’яті, навантаження на базу даних, а також обсяг переданих даних по мережі.

Відповідна логіка розміщена в модулі scripts/perfomancetracker/core/, який відповідає за збір, агрегацію та збереження показників продуктивності. Під час роботи кожного скрапера фіксуються часові метки, розмір зібраних даних, кількість оброблених URL, кількість помилок, а також середній час відповіді. Зібрані метрики записуються до бази даних і можуть бути використані для подальшого аналізу та оптимізації — наприклад, зміни стратегії ротації проксі, структури запитів до API або параметрів збереження.

Ці механізми дають змогу виявити вузькі місця в системі ще на етапі розробки та прийняття змін, що особливо критично при масштабуванні обсягів обробки або підключенні нових ресурсів з великою динамікою оновлення.

3.3.2 Валідація та обробка помилок

Для підтримання високої якості даних, зібраних із веб-ресурсів, у системі реалізовано багаторівневу валідацію та обробку помилок. Ці механізми дозволяють не лише вчасно виявляти проблеми, але й забезпечувати прозорий процес їх усунення, з мінімальним втручанням користувача.

Система валідації даних складається з трьох основних рівнів. Перший — це Field Validators, які перевіряють окремі поля на коректність і відповідність очікуваним форматам. Наприклад, валідатори для цін фільтрують нереалістичні значення, враховуючи граничні діапазони для конкретного регіону або типу нерухомості. Аналогічні правила застосовуються для площі, дат, кількості кімнат тощо. Другий рівень — Entity Validators — відповідає за перевірку цілісності об’єкта нерухомості як логічної одиниці. На цьому рівні перевіряється, чи наявні всі обов’язкові атрибути, та чи відповідають вони бізнес-логіці. Третій рівень — Cross-field Validators — оцінює зв’язки між різними полями, наприклад, узгодженість площі з кількістю кімнат або дати публікації з фактичним терміном дії оголошення. Приклад валідації ціни зображено на рисунку 3.5.

```
if (rental_period := item.get("rental_period")) and "rental_period" not in self.skip_requirem
    if len(rental_period) < 1 or len(rental_period) > 50:
        raise ValueError(f"Rental period is out of bounds [{rental_period}]")

# check for price bounds 2000-50000 Kron and 100-70000 Euro
if (rent := item.get("rent")) and "rent" not in self.skip_requirement_check:
    if spider.country.lower() == "denmark" or spider.country.lower() == "sweden":
        if rent < 2000 or rent > 50000:
            raise ValueError(f"Rent is out of bounds [{rent}]")
    else:
        if rent < 100 or rent > 70000:
            raise ValueError(f"Rent is out of bounds [{rent}]")

# prepaid_rent bounds 1000-50000 Kron and 50-70000 Euro
if (prepaid_rent := item.get("prepaid_rent")) and "prepaid_rent" not in self.skip_requirement
    if spider.country.lower() == "denmark" or spider.country.lower() == "sweden":
        if prepaid_rent < 1000 or prepaid_rent > 50000:
```

Рисунок 3.5 – Валідація ціни

Додатково валідаційна система підтримує концепцію трешхолдів — граничних значень, перевищення яких означає наявність критичних помилок. Наприклад, якщо понад 20% об'єктів із одного джерела мають відсутню адресу або площу, такий скрапер автоматично помічається як нестабільний.

Для відстеження проблем під час роботи системи впроваджено повноцінне структуроване логування. Усі повідомлення поділяються на чотири рівні: критичні помилки (Error Logs), що призводять до зупинки скрапера; попередження (Warning Logs), які не блокують роботу, але сигналізують про можливі відхилення; інформаційні записи (Info Logs) для моніторингу поточного стану; а також відлагоджувальні повідомлення (Debug Logs), що використовуються під час розробки або діагностики. Логи зберігаються у відповідній таблиці бази даних із прив’язкою до домену, часу, типу помилки та коротким описом.

Логування реалізовано за допомогою middleware, яке перехоплює винятки під час виконання скраперів, класифікує їх та направляє у відповідні канали. Для подальшого аналізу помилок розроблено спеціальний скрипт, який виконує агрегацію логів, виявляє повторювані шаблони та формує узагальнені звіти для аналітики.

З метою підвищення автоматизації супроводу системи реалізовано механізм автоматичного створення задач у Trello при виявленні критичних збоїв. Цей модуль базується на інтеграції з Trello API, яка реалізована у файлі TrelloAsignment.py. У разі виявлення проблем модуль автоматично формує картку з описом проблеми, додатковою інформацією з логів і прив’язкою до відповідного домену або скрапера. Встановлюється рівень пріоритету залежно від серйозності збою, а також визначається виконавець за попередньо встановленими правилами.

Завдяки поєднанню системи валідації, логування та автоматизованого створення задач у Trello, модуль підтримує замкнутий цикл контролю якості, що дозволяє виявляти, класифікувати та усувати помилки у найкоротші терміни без залучення надлишкових людських ресурсів.

3.3.3 Моніторинг і контроль якості

У системі збору даних про нерухомість впроваджено повноцінну інфраструктуру моніторингу й контролю якості, яка забезпечує прозорість, стабільність та оперативне реагування на проблеми. Вона охоплює всі етапи — від запуску скраперів до збереження результатів, аналізу продуктивності, контролю якості даних і автоматичного розширення джерел.

Для відстеження ефективності роботи скраперів реалізовано систему моніторингу запусків на базі модуля scripts/perfomancetracker/core/. Під час кожного запуску фіксується низка ключових метрик: кількість оброблених URL, знайдених об'єктів нерухомості, кількість і типи помилок, тривалість виконання, а також споживання системних ресурсів. Ці метрики зберігаються в базі даних і використовуються для побудови візуального дашборду, що надає аналітичне уявлення про стан і динаміку роботи системи.

Для оперативного інформування про критичні події реалізовано інтеграцію з Telegram, яка функціонує через спеціально налаштований бот у каталозі scripts/perfomancetracker/tgbot/. Бот надсилає повідомлення кількох типів: критичні

сповіщення при збоях чи падінні скраперів, щоденні звіти про кількість оброблених доменів та якість даних, повідомлення про погіршення продуктивності (наприклад, аномально низький обсяг результатів або надмірна тривалість), а також попередження про потенційні проблеми з якістю даних.

Ще один важливий компонент — автоматичне розширення системи через пошук нових сайтів. Механізм побудовано на використанні SerpAPI і реалізовано в модулі DomainsAsigner.py. Пошук здійснюється за ключовими словами, що відповідають тематикі нерухомості, з урахуванням країни чи регіону. Отримані результати аналізуються на релевантність (тип сайту, кількість оголошень, активність), після чого метадані зберігаються в базі даних, а для перспективних ресурсів автоматично створюються задачі в Trello на розробку відповідних скраперів.

Повний цикл запуску й моніторингу скраперів охоплює як стандартний, так і високочастотний режим роботи. У стандартному режимі скрипт schedule_spiders.py запускається за допомогою cron тричі на день. Він формує список активних скраперів із бази даних, запускає їх через Scrapyd API (Додаток В), зберігає результати в PostgreSQL та генерує звіти. У разі виявлення аномалій система автоматично надсилає сповіщення в Telegram і створює відповідні задачі в Trello.

У високочастотному режимі скрипт виконується кожні 10 хвилин із параметром --high-frequency. Він формує окремий список скраперів, оптимізованих для частого збору нових даних. Для кожного скрапера здійснюється обхід основної сторінки, виявлення нових URL і фільтрація вже оброблених об'єктів. Парсинг відбувається лише для нових оголошень, що дозволяє зменшити навантаження й забезпечити оперативне оновлення. Результати фіксуються в базі даних із відміткою про режим запуску, після чого формується окремий звіт.

Завдяки комплексному підходу до моніторингу, включаючи метрики, сповіщення, автоматизацію й інтеграцію з Trello, система підтримує високу стабільність і контрольованість процесів навіть за умови великої кількості джерел і частих оновлень.

Таким чином, у даному розділі було детально розглянуто практичну реалізацію та тестування системи автоматизованого збору даних про нерухомість. Реалізовано всі компоненти, визначені в архітектурі системи, та забезпечено їх ефективну взаємодію. Ключовими досягненнями практичної реалізації є: розгортання повноцінної інфраструктури на базі Scrapyd та PostgreSQL, що забезпечує стабільну роботу та масштабованість системи, реалізація спеціалізованих модулів для обробки різних типів даних та структур (ItemLoader, TableParser, AddressParser), що забезпечують високу якість та структурованість зібраної інформації.

1. Інтеграція з Playwright для ефективної обробки динамічного JavaScript-контенту, що дозволяє працювати з сучасними веб-сайтами.
2. Впровадження багаторівневої системи захисту від блокувань, що забезпечує стабільність роботи навіть на сайтах з активними механізмами захисту від скрапінгу.
3. Реалізація повної автоматизації життєвого циклу скраперів через CI/CD та інтеграцію з Scrapyd, що мінімізує ручні операції та підвищує ефективність розробки.
4. Впровадження комплексної системи моніторингу та контролю якості, що забезпечує високу надійність та точність зібраних даних.
5. Реалізація механізмів автоматичного розширення системи через пошук та інтеграцію нових джерел даних, що забезпечує постійне зростання бази даних.

Проведене тестування підтвердило відповідність системи всім функціональним та нефункціональним вимогам, визначеним у першому розділі. Система демонструє високу продуктивність, надійність та якість даних, що дозволяє ефективно використовувати її для збору та аналізу інформації про ринок нерухомості в різних країнах світу.

ВИСНОВКИ

У кваліфікаційній роботі було розроблено масштабовану систему автоматизованого збору та структуризації даних про нерухомість з великої кількості веб-ресурсів у різних країнах світу. Результати роботи дозволяють зробити наступні висновки:

1. Проведений аналіз предметної області продемонстрував відсутність універсальних рішень для збору даних про нерухомість, які б одночасно забезпечували масштабованість, гнучкість та стійкість до блокувань при оптимальних витратах ресурсів. Було виявлено основні технологічні, структурні та юридичні виклики, що виникають при розробці подібних систем.

2. Розроблена архітектура системи базується на фреймворку Scrapy з суттєвим розширенням його функціональності через додаткові модулі. Ключовою особливістю архітектури є її модульність, що забезпечує незалежну розробку, тестування та масштабування окремих компонентів.

3. Реалізовано універсальний ItemLoader зі спеціалізованими процесорами даних, що забезпечує стандартизацію обробки інформації з різних джерел та суттєво підвищує якість і уніфікованість зібраних даних. Модуль вирішує проблему різноманітності форматів представлення даних про нерухомість у різних країнах.

4. Розроблено компоненти для автоматичного аналізу структури даних (AutoPagination, TableParser, AddressParser), що дозволяють ефективно обробляти різні типи веб-сторінок з мінімальним налаштуванням для кожного джерела. Це суттєво спрощує процес додавання нових скраперів та підвищує їх стабільність.

5. Впроваджено механізми для роботи з динамічним контентом через інтеграцію з Playwright, що дозволило ефективно обробляти сучасні веб-сайти, які використовують JavaScript для рендерингу контенту. Реалізовано підходи для роботи з нескінченним скролінгом, AJAX-запитами та захищеними API.

6. Створено багаторівневу систему захисту від блокувань, що включає ротацію проксі, механізми обходу CAPTCHA та емуляцію TLS-фінгерпринту браузера.

Модуль обходу захисту від ботів динамічно адаптується до захисних механізмів веб-сайтів та оптимізує використання ресурсів на основі їх ефективності.

7. Реалізовано автоматизацію життєвого циклу скраперів через CI/CD систему на базі GitHub Actions та інтеграцію з Scrapyd. Це забезпечує автоматичне тестування, розгортання та моніторинг скраперів, мінімізуючи ручні операції та підвищуючи стабільність системи.

8. Впроваджено механізми високочастотного збору даних (HighFrequencyRunner), що дозволяють отримувати інформацію про нові оголошення майже в реальному часі з мінімальним навантаженням на джерела даних. Це особливо важливо для ринків нерухомості з високою динамікою.

9. Розроблено комплексну систему моніторингу та контролю якості з інтеграцією з Telegram для сповіщень та Trello для управління задачами. Система автоматично виявляє аномалії, створює завдання для їх виправлення та інформує відповідальних осіб, що забезпечує високу надійність роботи.

10. Створено механізми автоматичного розширення системи через пошук та інтеграцію нових джерел даних з використанням SerpAPI. Це забезпечує постійне розширення географічного покриття та обсягу зібраних даних без суттєвого збільшення витрат на розробку.

Практичне значення розробленої системи полягає в створенні повноцінної екосистеми для автоматизованого збору даних про нерухомість, яка може бути використана компаніями-агрегаторами, інвестиційними фондами, аналітичними агентствами та науково-дослідними установами. Система демонструє високу продуктивність (обробка понад 100 000 сторінок на годину), точність вилучення даних (понад 98% для структурованих даних) та масштабованість (підтримка 17 країн та понад 10 000 доменів).

Перспективи подальшого розвитку системи включають:

1. Розширення функціональності через впровадження алгоритмів машинного навчання для автоматичного визначення структури сайтів та вилучення даних без попереднього налаштування.

2. Інтеграцію з геоінформаційними системами для розширеного аналізу географічного розташування об'єктів нерухомості.

3. Розробку аналітичного модуля для виявлення трендів ринку нерухомості на основі зібраних даних.

4. Впровадження механізмів розподілених обчислень для подальшого підвищення продуктивності при обробці надвеликих обсягів даних.

5. Створення публічного API для доступу до зібраних даних зі збереженням конфіденційності чутливої інформації.

Система що містить розроблені модулі є стабільною, масштабованою та готовою до промислового використання, з можливістю легкої адаптації до нових країн, платформ та типів даних про нерухомість.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Scrapy. Scrapy 2.11 documentation. (n.d.). URL: https://docs.scrapy.org/en/latest/ (дата звернення: 10.01.2025).
2. Playwright. Playwright: Fast and reliable end-to-end testing for modern web apps. (n.d.). URL: https://playwright.dev/ (дата звернення: 16.01.2025).
3. peewee. Peewee ORM documentation. (n.d.). URL: http://docs.peewee-orm.com/ (дата звернення: 02.02.2025).
4. Scrapyd. Scrapyd 1.3 documentation. (n.d.). URL: https://scrapyd.readthedocs.io/en/stable/ (дата звернення: 02.02.2025).
5. OpenAI. OpenAI: We are researching and engineering safe and beneficial AGI. (n.d.). URL: https://openai.com/ (дата звернення: 12.03.2025).
6. SerpAPI. SerpApi: Real-time search engine results API. (n.d.). URL: https://serpapi.com/ (дата звернення: 09.01.2025).
7. 2Captcha. 2Captcha: Automatic CAPTCHA recognition service. (n.d.). URL: https://2captcha.com/ (дата звернення: 22.03.2025).
8. GitHub. About GitHub Actions. (n.d.). GitHub Docs. URL: https://docs.github.com/en/actions/learn-github-actions/about-github-actions (дата звернення: 11.03.2025).
9. Trello. Trello: Your team’s visual tool for planning anything. (n.d.). URL: https://trello.com/ (дата звернення: 26.01.2025).
10. PostgreSQL. PostgreSQL: The World's Most Advanced Open Source Relational Database. (n.d.). URL: https://www.postgresql.org/ (дата звернення: 02.02.2025).
11. Zillow. Zillow: Real Estate, Homes for Sale, Apartments & Houses for Rent. (n.d.). URL: https://www.zillow.com/ (дата звернення: 15.01.2025).
12. Rightmove. Rightmove: UK's number 1 property website. (n.d.). URL: https://www.rightmove.co.uk/ (дата звернення: 15.01.2025).
13. Immoweb. Immoweb: The leading real estate website in Belgium. (n.d.). URL: https://www.immoweb.be/en (дата звернення: 15.01.2025).

14. PropertyData. PropertyData: The UK's best property data platform. (n.d.). URL: https://www.propertydata.com/ (дата звернення: 16.01.2025).

15. CoreLogic. CoreLogic: Data, Analytics, and Services for the Housing Industry. (n.d.). URL: https://www.corelogic.com/ (дата звернення: 14.01.2025).

16. Apify. Apify: Web scraping, data extraction & RPA platform. (n.d.). URL: https://apify.com/ (дата звернення: 14.01.2025).

17. Zyte. Scrapy Cloud: The best web scraping platform. (n.d.). URL: https://www.zyte.com/scrapy-cloud/ (дата звернення: 12.01.2025).

18. ScrapingBee. ScrapingBee: The easiest web scraping API. (n.d.). URL: https://www.scrapingbee.com/ (дата звернення: 12.01.2025).

19. Thomson, A., Maglaras, L., & Moradpoor, N. (2025). A Novel TLS-Based Fingerprinting Approach that Combines Feature Expansion and Similarity Mapping. Sensors, 17(3), 120. URL: https://www.mdpi.com/1999-5903/17/3/120 (дата звернення: 28.03.2025).

20. MDN Web Docs. AJAX. (n.d.). URL: https://developer.mozilla.org/en-US/docs/Glossary/AJAX (дата звернення: 15.01.2025)

21. W3C. Document Object Model (DOM). W3C Recommendation. URL: https://www.w3.org/TR/REC-DOM-Level-1/ (дата звернення: 12.01.2025).

22. IETF. The WebSocket Protocol. (2011). RFC 6455. URL: https://www.rfc-editor.org/rfc/rfc6455 (дата звернення: 15.01.2025).

23. AWS. What is an API?. (n.d.). Amazon Web Services. URL: https://aws.amazon.com/what-is/api/ (дата звернення: 15.03.2025).

24. ResearchGate. Web Scraping Techniques and Applications: A Literature Review. (2023). URL: https://www.researchgate.net/publication/367719780_Web_Scraping_Techniques_and_Applications_A_Literature_Review (дата звернення: 08.02.2025).

25. ArXiv. A Comprehensive Survey of Social Media Bot Detection Techniques. (2025). URL: https://arxiv.org/abs/2503.22838 (дата звернення: 22.03.2025).

26. San Diego State University. Textual Analysis in Real Estate Research. (n.d.). URL: https://business.sdsu.edu/research/_files/_realestate/textual-analysis-in-real-estate.pdf (дата звернення: 16.03.2025).

27. Wikipedia. CI/CD. (n.d.). URL: https://en.wikipedia.org/wiki/CI/CD (дата звернення: 11.03.2025).

28. Cloudflare. What is a CAPTCHA?. (n.d.). URL: https://www.cloudflare.com/learning/bots/what-is-captcha/ (дата звернення: 08.06.2025).

29. Google. reCAPTCHA. (n.d.). URL: https://www.google.com/recaptcha/about/ (дата звернення: 25.03.2025).

30. Cloudflare. What is rate limiting?. (n.d.). URL: https://www.cloudflare.com/learning/bots/what-is-rate-limiting/ (дата звернення: 25.03.2025).

31. Imperva. IP Blocking. (n.d.). URL: https://www.imperva.com/learn/application-security/ip-blocking/ (дата звернення: 26.03.2025).

32. Snyk. Browser Fingerprinting Explained. (n.d.). URL: https://snyk.io/learn/browser-fingerprinting/ (дата звернення: 25.03.2025).

33. Cron. Cron - Wikipedia. (n.d.). URL: https://en.wikipedia.org/wiki/Cron (дата звернення: 02.02.2025).

34. Wikipedia. Parallel computing. (n.d.). URL: https://en.wikipedia.org/wiki/Parallel_computing (дата звернення: 20.01.2025).

35. Wikipedia. Asynchronous programming. (n.d.). URL: https://en.wikipedia.org/wiki/Asynchronous_programming (дата звернення: 20.01.2025).

36. Малко В., Биковий П. Програмний модуль збору даних про довгострокову оренду житла з використанням фреймворків Scrapy та Playwright. *Інтелектуальні інформаційні технології в прикладних дослідженнях (IITAR-2025)*: збірник тез студентської науково-практичної конференції, м. Тернопіль, 27-29 травня 2025 року. Тернопіль: ЗУНУ, 2025. С. 28-31.

37. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С., Ліп’яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп’ютерні науки» спеціальності 122 «Комп’ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

Додаток А

Код модуля TableParser

```
class TableParser:
    _keywoards = {
        "rent": {
            "keywoards": ["rent", "price", "cost", ...
            "exclude": ["deposit"],
        },
        "room_count": {
            "keywoards": ["bedroom", ...
            "exclude": ["bathroom"],
        },
        "bathroom_count": {
            "keywoards": ["bathroom", ...
            "exclude": ["bedroom"],
        },
        "property_type": {
            "keywoards": ["property type", "Category", ...
            "exclude": [],
        },
        ...
    }

    # add features to keywoards
    _keywoards.update({feature: {"keywoards": [feature], "exclude": []}
for feature in FEATURES.keys()})
    _fields_found = {}

    def __init__(self, response):
        self.response = response.copy()
        try:
            self.clean_response =
self._clear_html_content(self.response)
            self.tables = self.parse_tables()
        except:
            self.tables = {}

    def _find_fields(self, keyword) -> dict:
        if not self.tables:
            return {}

        found = {}
        keywoards = self._keywoards.get(keyword, {}).get("keywoards",
[])
        exclude = self._keywoards.get(keyword, {}).get("exclude", [])
        for keyword in keywoards:
            key, value = self.get(key=keyword, exclude=exclude,
with_key=True)
```

```
            if key and value:
                if key not in [key for key, _ in found.values()]:
                    found[keyword] = (key, value)

        return found

    def extract_values_from_table(self) -> dict:
        if self._fields_found:
            return self._fields_found
        fields = {}

        for field in self._keywoards.keys():
            if field_result := self._find_fields(field):
                fields[field] = field_result
        if fields:
            self._fields_found = fields
        return fields

    def _clear_html_content(self, response):
        # Define XPath to exclude unwanted elements
        excluded_xpath = """
            //header | //footer | //nav | //script | //style |
            //*[contains(concat(' ', normalize-space(@class), ' '), '
header ')] |
            //*[contains(concat(' ', normalize-space(@class), ' '), '
footer ')] |
            //*[contains(concat(' ', normalize-space(@class), ' '), '
nav ')] |
            //*[contains(concat(' ', normalize-space(@class), ' '), '
navbar ')] |
            //*[contains(translate(@id, 'HEADER', 'header'), 'header')]
|
            //*[contains(translate(@id, 'FOOTER', 'footer'), 'footer')]
|
            //*[contains(translate(@id, 'NAV', 'nav'), 'nav')] |
            //*[contains(translate(@id, 'NAVBAR', 'navbar'), 'navbar')]
        """

        # Remove excluded elements from the response
        for element in response.xpath(excluded_xpath):
            element.drop()

        # Extract the cleaned body content
        cleaned_body = response.xpath("//body").get()

        # Return the response with the cleaned body
        return response.replace(body=cleaned_body)

    def _clean_key_value(self, key, value):
        # clean it from whitespaces and newlines etc
        key = re.sub(r"\s+", " ", key).strip()
```

```
        value = re.sub(r"\s+", " ", value).strip()
        return key, value

    def _make_key_value(self, row, max_depth=5):
        def get_direct_texts(r):
            return [t.strip() for t in r.xpath('./text()').getall() if
t.strip()]

        def get_nested_texts(r):
            return [t.strip() for t in r.xpath('.//*/text()').getall()
if t.strip()]

        def get_all_texts(r):
            return [t.strip() for t in r.xpath('.//text()').getall() if
t.strip()]

        # Condition 0: Direct text contains a colon
        direct_texts = get_direct_texts(row)
        if direct_texts:
            direct_text = ' '.join(direct_texts)
            if ':' in direct_text:
                key_part, sep, value_part = direct_text.partition(':')
                key_part, value_part = key_part.strip(),
value_part.strip()
                if key_part and value_part:
                    return self._clean_key_value(key_part, value_part)

        # Condition 1: Exactly two child elements
        children = row.xpath('./*')
        if len(children) == 2:
            key_elem, value_elem = children
            key = key_elem.xpath('string(.)').get().strip()
            value = value_elem.xpath('string(.)').get().strip()
            if key and value:
                return self._clean_key_value(key, value)

        # Condition 2: Direct text and nested texts
        direct_texts = get_direct_texts(row)
        nested_texts = get_nested_texts(row)

        if direct_texts and nested_texts:
            # Get the full text content of the row
            all_texts = row.xpath('.//text()').getall()
            all_texts = [t.strip() for t in all_texts if t.strip()]

            # Determine which text appears first
            if all_texts[0] in direct_texts:
                # Direct text is first, so it's the key
                key = ' '.join(direct_texts)
                value = ' '.join(nested_texts)
            else:
```

```
                # Nested text is first, so it's the key
                key = ' '.join(nested_texts)
                value = ' '.join(direct_texts)

            return self._clean_key_value(key, value)

        # Condition 3: All texts (direct + nested) count
        all_texts = get_all_texts(row)
        if len(all_texts) == 2:
            return self._clean_key_value(all_texts[0], all_texts[1])
        elif len(all_texts) > 2:
            return self._clean_key_value(all_texts[0], '
'.join(all_texts[1:]))

        # Condition 4: Single text, explore nested elements iteratively
        if len(all_texts) == 1:
            from collections import deque
            queue = deque([(row, 0)])
            while queue:
                current_elem, depth = queue.popleft()
                if depth >= max_depth:
                    continue
                # Check each child element for key-value
                for child in current_elem.xpath('./*'):
                    # Reapply conditions to child
                    # Condition 0 for child's direct text
                    child_direct = get_direct_texts(child)
                    if child_direct:
                        child_direct_text = ' '.join(child_direct)
                        if ':' in child_direct_text:
                            kp, _, vp =
child_direct_text.partition(':')
                            kp, vp = kp.strip(), vp.strip()
                            if kp and vp:
                                return self._clean_key_value(kp, vp)
                    # Condition 1 for child's children count
                    child_children = child.xpath('./*')
                    if len(child_children) == 2:
                        k_elem, v_elem = child_children
                        k = k_elem.xpath('string(.)').get().strip()
                        v = v_elem.xpath('string(.)').get().strip()
                        if k and v:
                            return self._clean_key_value(k, v)
                    # Condition 2 for child's direct and nested
                    child_dir = get_direct_texts(child)
                    child_nest = get_nested_texts(child)
                    if child_dir and child_nest:
                        k = ' '.join(child_dir)
                        v = ' '.join(child_nest)
                        return self._clean_key_value(k, v)
                    # Condition 3 for child's all texts
```

```
                    child_all = get_all_texts(child)
                    if len(child_all) == 2:
                        return self._clean_key_value(child_all[0],
child_all[1])
                    elif len(child_all) > 2:
                        return self._clean_key_value(child_all[0], '
'.join(child_all[1:]))
                    # Add to queue for deeper exploration
                    queue.append((child, depth + 1))

        # Fallback: return entire text
        row_text = row.xpath('string(.)').get().strip()
        return self._clean_key_value(row_text, row_text)

    def parse_tables(self):
        # we should get all tr, li, .tablearea, and .row (but we need
to understand if its not too big elements)
        rows = self.clean_response.xpath("//tr | //li |
//div[contains(@class, 'tablearea')] | //div[contains(@class, 'row')] |
//div[contains(@class, 'propsRow')] | //div[contains(@class,
'listing_detail')] | //div[contains(@class, 'box')]")
        # we want to make key-value pairs from rows
        filtered_rows = []
        for row in rows:
            # Structural validation checks
            html_content = row.get()

            # clear from whitespaces and newlines and weird characters
            html_content = re.sub(r"\s+", "", html_content)
            html_content = re.sub(r"[\x00-\x1f\x7f-\x9f]", "",
html_content)

            # 1. Size-based filtering
            if len(html_content) > 300:
                continue

            # 2. Text density check
            raw_text = row.xpath('.//text()').getall()
            clean_text = [t.strip() for t in raw_text if t.strip()]
            if len(clean_text) < 2:  # Require at least 2 meaningful
text fragments
                continue

            filtered_rows.append(row)

        # Process filtered rows with quality checks
        tables = {}
        for row in filtered_rows:
            key, value = self._make_key_value(row)

            # Post-processing validation
```

```
            if not key or not value:
                continue
            if len(key) > 40 or len(value) > 100:
                continue
            if key.strip() == value.strip() and len(key.split()) < 3:
                continue

            tables[key] = value

        return tables

    def get(self, key, default=None, cutoff=0.62, with_key=False,
exclude=[], exact=False):
        """
        Search for the best match of the key in the dictionary keys
with performance optimizations.
        Keys are checked using split words for exclusions and matching.
        """
        normalized_key = key.lower().strip()
        normalized_exclude = [e.lower().strip() for e in exclude]

        # Add reciprocal exclusion for bedroom/bathroom
        if "bedroom" in normalized_key:
            normalized_exclude.append("bathroom")
        elif "bathroom" in normalized_key:
            normalized_exclude.append("bedroom")

        normalized_exclude_set = set(normalized_exclude)

        # Handle exact match case first
        if exact:
            # Build direct lookup map for exact matches
            exact_match_map = {}
            for orig_key in self.tables:
                norm_key = orig_key.lower().strip()
                if norm_key not in exact_match_map:
                    exact_match_map[norm_key] = orig_key

            if normalized_key in exact_match_map:
                matched_key = exact_match_map[normalized_key]
                if with_key:
                    return (matched_key, self.tables[matched_key])
                return self.tables[matched_key]
            else:
                return (default, default) if with_key else default

        # Precompute combined keys and values with their lowercase
versions
        combined = []
        for k, v in self.tables.items():
            combined.append((k, k.lower()))
```

```
            combined.append((v, v.lower()))

        # Filter using split words
        filtered_combined = []
        for orig_str, lc_str in combined:
            words = re.split(r'[^\w]', lc_str)  # Split on non-word
characters
            if not any(word in normalized_exclude_set for word in words
if word):
                filtered_combined.append((orig_str, lc_str))

        # Create lookup mapping for fast access
        lookup_map = {}
        for k, v in self.tables.items():
            lc_k = k.lower()
            lc_v = v.lower()
            lookup_map.setdefault(lc_k, []).append((k, v))
            lookup_map.setdefault(lc_v, []).append((k, v))

        # Try substring matches first
        substring_candidates = [lc for _, lc in filtered_combined if
normalized_key in lc]
        best_match = None

        if substring_candidates:
            # Fuzzy match on substring candidates
            matches = get_close_matches(normalized_key,
substring_candidates, n=1, cutoff=cutoff)
            if matches:
                best_match = matches[0]

        # Fallback to full fuzzy match
        if not best_match:
            all_candidates = [lc for _, lc in filtered_combined]
            matches = get_close_matches(normalized_key, all_candidates,
n=1, cutoff=cutoff)
            if matches:
                best_match = matches[0]

        # Return results
        if best_match:
            matches = lookup_map.get(best_match, [])
            if matches:
                k, v = matches[0]  # Get first match preserving
insertion order
                return (k, v) if with_key else v

        return (default, default) if with_key else default
```

Додаток Б

Код модуля AddressParser

```
class AddressParser:
    type_priority = {
        'city': 3,
        'town': 2,
        'village': 1,
    }


    _street_regexes = {
        'unitedkingdom': r"\b([A-Za-z]?\d{1,5}[A-Za-z]?(?:-
\d{1,5})?)\b\,?\s+{street}\b",
        'belgium': r"{street}\s*(\d{1,5}[A-Za-z]?(?:[\/]\d{1,5})?)\b",
        'denmark': r"\b(\d{1,5}(?:/\d{1,5})?[A-Za-z]?(?:-
\d{1,5})?)\s+{street}\b",
        …
    }

    _zipcode_regexes = {
        'denmark': r"(?:^|\s)(\d{4})(?=\s|$)",
        'netherlands': r"\d{4}\s?[A-Z]{2}",
        'italy': r"(?:^|\s)(\d{5})(?=\s|$)",
        'portugal': r"\d{4}-\d{3}",
        …
    }

    def __init__(self, country):
        self.country = country

        self.db_cities = City.select().where(City.country ==
self.country)
        self.db_streets = Street.select().where(Street.country ==
self.country)

        self.city_processor = KeywordProcessor(case_sensitive=False)
        self.street_processor = KeywordProcessor(case_sensitive=False)

        self.street_regex =
self._street_regexes.get(self.country.lower(), r'\d{4}\s{street}')
        self.zipcode_regex =
self._zipcode_regexes.get(self.country.lower(), r'\d{4}')

        self._load_data()


    def _load_data(self):
        for city in self.db_cities.iterator():
```

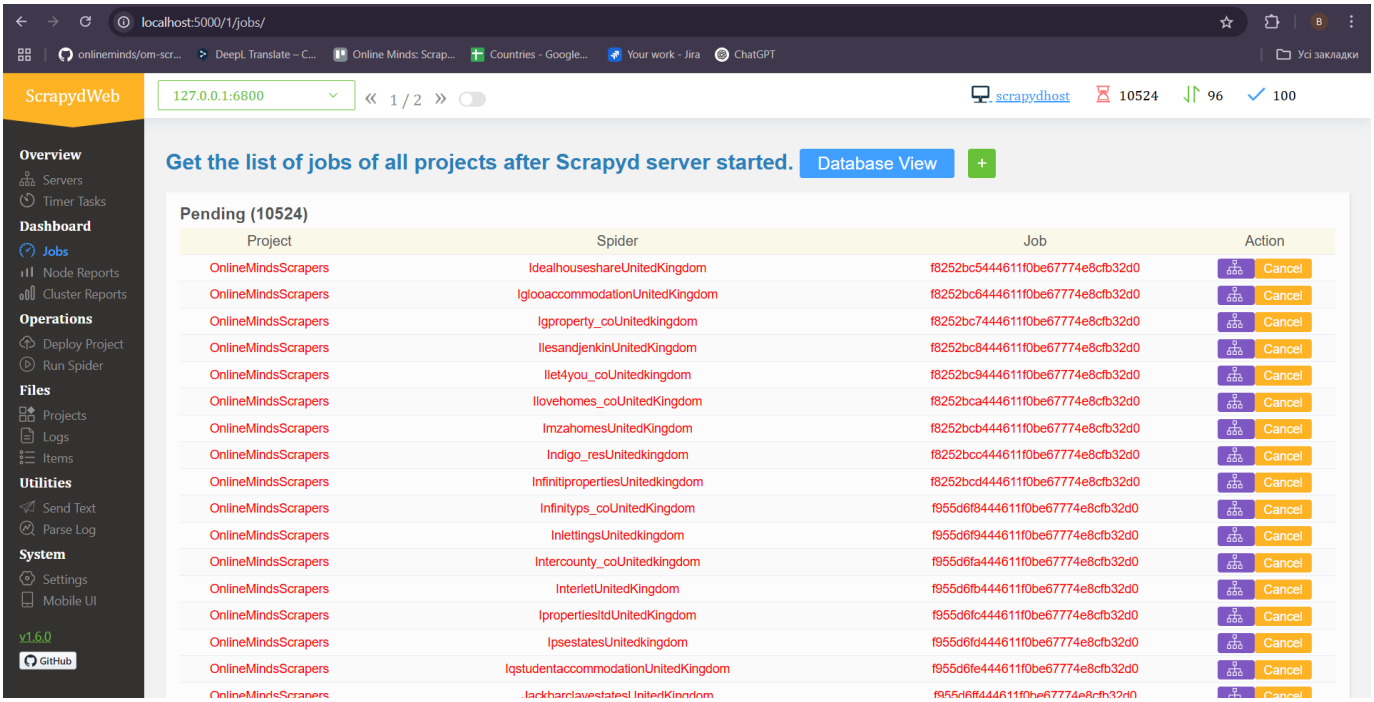
```
            self.city_processor.add_keyword(city.name.lower().strip())

        for street in self.db_streets.iterator():

self.street_processor.add_keyword(street.name.lower().strip())

    def _get_best_city(self, city_names: list[str]) -> City | None:
        """
        Returns city object with highest priority
        """
        city_objects =
self.db_cities.where(fn.LOWER(City.name).in_(city_names))
        if not city_objects:
            return None
        best_city_object = sorted(city_objects, key=lambda x:
self.type_priority.get(x.type, 0), reverse=True)[0]
        return best_city_object

    def _get_city_by_street(self, street_name: str) -> City | None:
        """
        Returns city name by street name if only one city found for
this street
        """
        street_object = self.db_streets.where(fn.LOWER(Street.name) ==
street_name.lower().strip()).limit(2)
        # if only one street found, return city of this street
        if street_object.count() == 1:
            try:
                return street_object[0].city.name
            except City.DoesNotExist:
                return None
        return None

    def extract_city(self, text:str) -> str | None:
        """
        Extracts city name from text
        """
        return self.extract_city_object(text).name

    def extract_city_object(self, text:str) -> City:
        """
        Extracts city object from text
        """
        keywoards = self.city_processor.extract_keywords(text)
        return self._get_best_city(keywoards)

    def extract_street(self, text:str) -> str:
        """
        Extracts street name from text
        """
        keywoards = self.street_processor.extract_keywords(text)
```

```
        if not keywoards:
            return None
        # return biggest streets first
        keywoards = sorted(keywoards, key=lambda x: len(x),
reverse=True)
        return keywoards[0]

    def __call__(self, text: str|list[str], loader_context=None,
as_dict=False, *args, **kwds):

        if isinstance(text, list):
            text = " ".join(text)

        street = self.extract_street(text)
        street_number = None
        between_text = None
        city = loader_context['loader'].value('city') if loader_context
else None
        zipcode = loader_context['loader'].value('zipcode') if
loader_context else None

        city_object = self.extract_city_object(text)

        if not city:
            if city_object:
                city = city_object.name
            elif street:
                city = self._get_city_by_street(street)

        if not zipcode:
            zipcode_group = re.search(self.zipcode_regex, text)
            if zipcode_group:
                zipcode = zipcode_group.group(0)
            else:
                zipcode = city_object.zipcode if city_object else None
            if zipcode:
                zipcode = zipcode.strip()

        if street:
            street = street.title()
            street_number_group =
re.search(self.street_regex.replace('{street}', street), text,
re.IGNORECASE)
            if street_number_group:
                street_number = street_number_group.group(1).strip()
                print(street_number)

        if street and city:
            pattern = re.escape(street) + r'(.*?)' + re.escape(city)
            pattern_reverse = re.escape(city) + r'(.*?)' +
re.escape(street)
```

```
            match = re.search(pattern, text)
            if not match:
                match = re.search(pattern_reverse, text)
            if match:
                between_text = match.group(1).strip()
            # delete street zip and city and street number from between
text
            if between_text:
                for part in [street_number, street, city, zipcode]:
                    if part:
                        between_text = between_text.replace(part, "")
                between_text = between_text.strip(',').strip(' ')


        if not street and not street_number and not between_text and
not city and not zipcode:
            return None

        if self.country == 'unitedkingdom' or self.country ==
'australia':
            result_list = [street_number, street, between_text, city,
zipcode]

        elif self.country == 'canada' or self.country == 'australia':
            result_list = [street_number, street, city, between_text,
zipcode]

        elif self.country == 'ireland':
            result_list = [street, street_number, city, between_text,
zipcode]
        else:
            result_list = [street, street_number, between_text
,zipcode, city]

        result = ", ".join(part for part in result_list if part)

        if as_dict:
            return {
                'street': street,
                'street_number': street_number,
                'between_text': between_text,
                'city': city,
                'zipcode': zipcode,
                'result': result
            }
        return result
```

Додаток В

Дешборд статусу запущених павуків



Додаток Г

Копії опублікованих результатів

Західноукраїнський національний університет

Факультет комп’ютерних інформаційних технологій

Кафедра інформаційно-обчислювальних систем і управління

ЗБІРНИК ТЕЗ ДОПОВІДЕЙ

Студентської науково-практичної конференції

ІНТЕЛЕКТУАЛЬНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В ПРИКЛАДНИХ ДОСЛІДЖЕННЯХ

(IITAR-2025)

27-29 травня 2025 року

Тернопіль

2025

Малко Владислав
студент групи КН-42
vlad@onlineminds.io
Биковий Павло
к.т.н., доцент
pb@wunu.edu.ua
Західноукраїнський національний університет
Тернопіль, Україна

ПРОГРАМНИЙ МОДУЛЬ ЗБОРУ ДАНИХ ПРО ДОВГОСТРОКОВУ ОРЕНДУ ЖИТЛА З ВИКОРИСТАННЯМ ФРЕЙМВОРКІВ SCRAPY ТА PLAYWRIGHT

З розвитком глобалізації та зростанням мобільності населення, потреба в актуальній інформації про ринок оренди житла стала критичною для багатьох категорій користувачів: від студентів та трудових мігрантів до аналітиків ринку нерухомості. Існуючі аналоги були або надто локалізованими (охоплювали лише одну країну чи регіон), або недостатньо детальними.

Початкова концепція передбачала охоплення кількох європейських країн, але з часом проєкт еволюціонував до масштабного рішення, що охоплює 17 країн світу. Саме тому було обрано назву "OnlineMindsScrapers", що відображає глобальний характер системи та її основну функцію — збір (скрапінг) даних з онлайн-ресурсів.

Основною технічною базою проєкту стали фреймворки Scrapy та Playwright. Scrapy використано як основу для створення павуків (spiders), що виконують збір даних з веб-сайтів, тоді як Playwright забезпечив можливість роботи з динамічним контентом, який генерується JavaScript.

Архітектура модуля поділена на декілька ключових компонентів:

1. Основні "павуки" (spiders) — окремі класи для кожного веб-ресурсу, структуровані за країнами
2. Pipelines — конвеєри обробки зібраних даних з валідацією та нормалізацією
3. Middlewares — проміжне програмне забезпечення для керування запитами та відповідями
4. ItemLoaders — класи для завантаження та обробки даних з різних веб-структур

У процесі розробки довелося подолати низку технічних викликів:

1. Обхід систем захисту від скрапінгу. Багато цільових веб-сайтів використовували Cloudflare та інші сервіси для виявлення автоматизованих запитів. Для вирішення цієї проблеми було розроблено **CloudflareSolverMiddleware**, який використовує 2Captcha API для обходу CAPTCHA та інших перевірок.

2. Робота з динамічним контентом. Значна частина сайтів використовувала JavaScript для завантаження даних після рендерингу сторінки. Для цього була впроваджена інтеграція з Playwright, що дозволило емулювати реального користувача та отримувати дані після повного завантаження сторінки.

3. Геоблокування та блокування IP-адрес. Для обходу цих обмежень був реалізований **ProxyMiddleware** та **RetryProxyMiddleware**, що дозволили використовувати проксі-сервери з різних регіонів та автоматично перемикатися між ними у разі блокування.

4. Різноманітна структура даних. Кожен веб-сайт мав унікальну структуру даних, що ускладнювало їх уніфіковану обробку. Для вирішення цієї проблеми розроблено гнучку систему **ItemLoader**, яка адаптується до різних форматів даних.

5. Нестабільна структура DOM. Багато сайтів періодично змінювали свою структуру, що призводило до помилок у роботі "павуків". Для подолання цієї проблеми створено систему моніторингу з використанням Slack для сповіщень про помилки та автоматичного перезапуску павуків.

6. Обмеження швидкості (rate limiting). Для уникнення блокування через надмірну кількість запитів впроваджено систему затримок та **HighFrequencyMiddleware** для роботи з ресурсами, що потребують особливого підходу до частоти запитів.

Для керування даними використано змішаний підхід: кешування HTTP-відповідей у файловій системі для розробки та тестування, а також повноцінну базу даних для продуктивного середовища.

В результаті розробки створено потужний модуль для збору даних про оренду житла, який успішно працює з сотнями веб-сайтів у 17 країнах, включаючи Європу, Північну Америку та Австралію. Модуль здатний обробляти різнотипні дані та надавати їх у уніфікованому форматі для подальшого аналізу.

Основні досягнення проєкту:

1. Розроблено масштабовану архітектуру, що дозволяє легко додавати нові країни та веб-ресурси без суттєвих змін основного коду

2. Створено потужну систему обробки та валідації даних, що забезпечує високу якість зібраної інформації

3. Реалізовано механізми обходу різних систем захисту від автоматичного збору даних

4. Впроваджено систему моніторингу та сповіщень про помилки, що дозволяє оперативно реагувати на зміни в структурі веб-ресурсів

Кожен "павук" здатен збирати детальну інформацію про оренду житла, включаючи ціни, характеристики нерухомості, наявність меблів, адреси, контактну інформацію орендодавців та інші важливі параметри.

Особливу увагу приділено якості даних — у модулі передбачено численні перевірки для виявлення та виправлення помилок, наприклад, валідація географічних координат для підтвердження, що об'єкт дійсно знаходиться в заявленій країні.

Система активно використовується для збору даних, які згодом аналізуються для виявлення тенденцій ринку оренди житла, цінової динаміки та інших важливих показників. Модуль демонструє високу ефективність та надійність у реальних умовах експлуатації.

Подальший розвиток проєкту передбачає додавання нових країн, вдосконалення механізмів обходу захисту та розширення аналітичних можливостей системи для більш детального аналізу зібраних даних.

Список використаних джерел

1. Scrapy documentation. URL: https://docs.scrapy.org/
2. Playwright for Python documentation. URL: https://playwright.dev/python/
3. Heydon, R. (2021). Web Scraping with Python: Collecting More Data from the Modern Web. O'Reilly Media.
4. Mitchell, R. (2018). Web Scraping with Python: Collecting Data from the Modern Web. O'Reilly Media.
5. Kazil, J., & Jarmul, K. (2020). Data Wrangling with Python. O'Reilly Media.