НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

«На правах рукопису»
УДК _004.738.5:004.6_

До захисту допущено:
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____2025 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою
«Інтегровані інформаційні системи»

зі спеціальності 126 «Інформаційні системи та технології»

на тему: «Інформаційна система автоматизованого збору даних про товари в інтернет-магазинах»

Виконав:
студент 2 курсу, групи ІА-42мп
Притуленко Олег Олександрович _____

Керівник:
професор каф. ІСТ, д.т.н., проф.
Корнієнко Богдан Ярославович _____

Рецензент:
доцент каф. ТПЗА, к.т.н., доц.
Ладієва Леся Ростиславівна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ — 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Рівень вищої освіти — другий (магістерський)

Спеціальність — 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інтегровані інформаційні системи»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____2025 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Притуленку Олегу Олександровичу

1. Тема дисертації «Інформаційна система автоматизованого збору даних про товари в інтернет-магазинах», науковий керівник дисертації Корнієнко Богдан Ярославович, д.т.н., проф., затверджені наказом по університету від «06» 11 2025 р. № 4841-с

2. Термін подання студентом дисертації «15» 12 2025 р.

3. Об’єкт дослідження: система для автоматизованого збору даних з інтернет-магазинів.

4. Вихідні дані: автоматизований збір даних про товари, функціональна можливість збору даних за розкладом, зберігання історії зборів даних, можливість експорту зібраних даних в різних форматах (Excel, CSV, JSON, Google Sheets, SQL, TXT та TSV).

5. Перелік завдань, які потрібно розробити: провести аналіз предметної області, провести аналіз існуючих рішень, спроєктувати архітектуру системи, розробити підсистему збору даних, розробити серверну частину та API, розробити клієнтську частину, провести тестування системи, підготувати пояснювальну записку.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: діаграма прецедентів, діаграма компонентів, ER-діаграма бази даних, загальний

алгоритм роботи користувача з системою збору даних, блок-схема роботи алгоритму збору даних, діаграма послідовності роботи користувача з системою збору даних, діаграма послідовності підсистеми автоматизованого запуску процесу збору даних, діаграма станів процесу збору даних.

7. Орієнтовний перелік публікацій: не планується

8. Дата видачі завдання 01.09.2025 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Аналіз предметної області та формування вимог	07.09.2025	
2.	Аналіз існуючих рішень та підходів до збору даних	14.09.2025	
3.	Проєктування архітектури системи	28.09.2025	
4.	Розробка підсистеми автоматизованого збору даних	12.10.2025	
5.	Розробка серверної та клієнтської частин	19.10.2025	
6.	Тестування системи	26.10.2025	
7.	Розробка стартап-проєкту	2.11.2025	
8.	Оформлення дисертації	7.12.2025	

Студент Олег ПРИТУЛЕНКО

Науковий керівник Богдан КОРНІЄНКО

РЕФЕРАТ

Інформаційна система автоматизованого збору даних про товари в інтернет-магазинах: 126 с., 28 табл., 34 рис., 9 дод., 51 джерел.

Ключові слова: АВТОМАТИЗАЦІЯ ЗБОРУ ДАНИХ, ВЕБСКРЕПІНГ, ЕЛЕКТРОННА КОМЕРЦІЯ, ІНТЕРНЕТ-МАГАЗИНИ, ПАРСИНГ HTML, AIOHTTP, DJANGO, REDIS, SELENIUM.

Автоматизований збір даних з вебресурсів є важливою технологією сучасного цифрового середовища. Ручне копіювання та обробка інформації з інтернет-магазинів неефективні через зростання обсягів даних та їхню динамічну зміну, що створює потребу в системах, які автоматизують збір даних, роблячи його швидким, точним та доступним для користувачів без технічних навичок.

Метою роботи є створення інформаційної системи для ефективного збору та зберігання даних з інтернет-магазинів з використанням сучасних технологій вебскрепінгу. Для цього проводився аналіз існуючих методів парсингу вебконтенту та можливості застосування різних інструментів для роботи з HTML.

Об'єктом дослідження є процес автоматизованого збору та зберігання інформації про товари з інтернет-магазинів.

Предметом дослідження є методи вебскрепінгу, архітектурні рішення для систем збору даних та технології збору даних інформації з HTML-сторінок для забезпечення ефективного моніторингу електронної комерції.

Для виконання роботи було використано методи проєктування програмного забезпечення для розробки архітектури системи, технології веброзробки, вебскрепінгу та парсингу для створення модулів збору даних, а також механізми кешування та планування завдань для забезпечення ефективної роботи системи.

Як результат, було розроблено інформаційну систему для автоматизованого збору даних з інтернет-магазинів за допомогою налаштовуваних шаблонів та періодичного оновлення інформації. Система може використовуватися для моніторингу цін конкурентів та дослідження ринкових трендів у сфері електронної комерції.

ABSTRACT

Information system for automated product data collection from online stores: 126 p., 28 tab., 34 draw., 9 app., 51 sources.

Keywords: AIOHTTP, DATA COLLECTION AUTOMATION, DJANGO, E-COMMERCE, HTML PARSING, ONLINE STORES, REDIS, SELENIUM, WEB SCRAPING.

Automated data collection from web resources is an important technology in the modern digital environment. Manual copying and processing of information from online stores is inefficient due to the growth of data volumes and their dynamic change, which creates a need for systems that automate data collection, making it fast, accurate, and accessible to users without technical skills.

The objective of this work is to create an information system for efficient collection and storage of data from online stores using modern web scraping technologies. To achieve this goal, an analysis of existing web content parsing methods was conducted and the possibilities of applying various tools for working with HTML were investigated.

The object of research is the process of automated collection and storage of product information from online stores.

The subject of research is web scraping methods, architectural solutions for data collection systems, and technologies for extracting information from HTML pages to ensure effective e-commerce monitoring.

To accomplish this work, software design methods were used to develop the system architecture, web development technologies, web scraping and parsing to create data collection modules, as well as caching and task scheduling mechanisms to ensure efficient system operation.

As a result, an information system was developed for automated data collection from online stores using configurable templates and periodic information updates. The system can be used to monitor competitor prices and research of market trends in the e-commerce sector.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

БД — база даних

Парсинг — автоматичний аналіз тексту з метою вилучення необхідної інформації

СУБД — система управління базою даних

Вебскрепінг — автоматизований збір даних з вебресурсів

AI — Artificial Intelligence — галузь інформатики, що вивчає методи створення систем, здатних виконувати інтелектуальні завдання, притаманні людині

AJAX — Asynchronous JavaScript and XML — технологія, яка забезпечує асинхронну взаємодію вебсторінки з сервером без повного перезавантаження сторінки

API — Application Programming Interface — набір інтерфейсів та правил, що дозволяють програмам взаємодіяти між собою

CAPTCHA — Completely Automated Public Turing test to tell Computers and Humans Apart — механізм автентифікації, що відрізняє людину від автоматизованих програм використовуючи тестові завдання

CRUD — Create, Read, Update, Delete — набір базових операцій над даними у системах управління базами даних

CSRF — Cross-Site Request Forgery — тип атаки, за якої зловмисник змушує користувача виконувати небажані дії на вебсайті, на якому той авторизований

CSS — Cascading Style Sheets — мова стилів, яка визначає зовнішній вигляд елементів вебсторінок

CSV — Comma-Separated Values — текстовий формат для зберігання табличних даних із розділенням значень комами

DRF — Django REST Framework — програмний фреймворк, що полегшує створення RESTful API на основі Django

GDPR — General Data Protection Regulation — регламент Європейського Союзу щодо захисту персональних даних та контролю їх обробки

HTML — HyperText Markup Language — стандартизована мова розмітки для структурування вебсторінок

HTTP — HyperText Transfer Protocol — протокол передачі гіпертекстових даних у мережі Інтернет

IP — Internet Protocol — протокол адресації та маршрутизації мережевого трафіку

ISO/IEC — International Organization for Standardization / International Electrotechnical Commission — система міжнародних технічних стандартів

JSON — JavaScript Object Notation — текстовий формат структурованих даних

JWT — JSON Web Token — формат токенів для автентифікації та авторизації

JSX — JavaScript XML — розширення синтаксису JavaScript для опису інтерфейсів у стилі XML

ML — Machine Learning — підгалузь штучного інтелекту, що досліджує алгоритми, здатні навчатися на основі даних

MTV — Model, Template, View — архітектурна парадигма фреймворку Django, що структурує програмну логіку, представлення та дані

MVC — Model, View, Controller — архітектурний шаблон розділення логіки, даних та представлення

ORM — Object-Relational Mapping — технологія, що забезпечує роботу з реляційними базами даних через об’єкти програмування

PEP — Python Enhancement Proposal — документ зі стандартами та покращеннями мови програмування Python

PWA — Progressive Web App — тип вебзастосунків, що забезпечують функціональність, подібну до нативних мобільних застосунків

REST — Representational State Transfer — архітектурний стиль побудови розподілених систем, що базується на стандартизованих HTTP-запитах

SaaS — Software as a Service — модель постачання програмного забезпечення як онлайн-сервісу

SPA — Single Page Application — вебзастосунок, що динамічно змінює вміст без повного перезавантаження сторінки

SQL — Structured Query Language — мова запитів, призначена для роботи з реляційними базами даних

TCP — Transmission Control Protocol — протокол доставки даних у мережах

TSV — Tab-Separated Values — текстовий формат, у якому дані подаються у вигляді таблиці, а значення в рядках розділені табуляцією

TXT — Plain Text File — текстовий формат, що містить лише символи без структури чи розмітки

UI — User Interface — інтерфейс користувача

URL — Uniform Resource Locator — уніфікований адресний ідентифікатор для доступу до ресурсів в інтернеті

UX — User Experience — досвід взаємодії користувача з продуктом

XML — Extensible Markup Language — мова розмітки для представлення структурованих даних

XPath — XML Path Language — мова для вибірки й навігації в XML-документах

XSS — Cross-Site Scripting — атака через впровадження шкідливого коду у вебсторінку

ВСТУП

Стрімкий розвиток інформаційних технологій та електронної комерції призвів до суттєвого збільшення обсягів даних, які щоденно публікуються в мережі Інтернет. Інтернет-магазини, торговельні майданчики та інші вебресурси містять велику кількість інформації про товари, ціни, характеристики, відгуки, наявність тощо. Ці дані мають значну цінність для бізнесу, маркетингової аналітики, наукових досліджень та прийняття управлінських рішень. Проте обсяг і динамічність зміни таких даних робить їх ручне збирання практично неможливим або надзвичайно неефективним.

В умовах сучасного інформаційного середовища зростає потреба в автоматизованих системах, здатних швидко збирати, обробляти та структурувати інформацію з вебресурсів. Такі системи дають змогу отримувати актуальні дані в реальному часі, порівнювати пропозиції конкурентів, аналізувати тенденції ринку тощо.

Хоча на міжнародному ринку існують численні інструменти для автоматизованого збору даних, їх адаптація до умов українського сегмента електронної комерції є обмеженою. В Україні подібні систем розвинені недостатньо, і на даний момент для малого та середнього бізнесу практично відсутні доступні та ефективні рішення, які б забезпечували вигідні цінові умови та відповідали локальним потребам.

Завданням даної роботи є створення інформаційної системи, яка забезпечує автоматизований збір, обробку, оновлення та зберігання даних з інтернет-магазинів. Розробка такої системи спрямована на підвищення ефективності процесів аналітики та моніторингу електронної комерції та оптимізацію отримання актуальної інформації про товари.

При розробці системи особливу увагу буде приділено створенню інструментів, що дозволяють користувачеві самостійно налаштовувати процес збору даних, переглядати та отримувати зібрані дані у форматах, зручних для подальшого аналізу. Такий підхід забезпечить гнучкість системи та можливість її

адаптації під різні категорії товарів, джерела даних та специфіку бізнес-процесів українських інтернет-магазинів.

При проєктуванні системи важливо здійснити аналіз різних підходів до збору даних та можливих архітектурних рішень, з метою вибору оптимального варіанту, який забезпечить достатню швидкість збору інформації та ефективність її обробки. Такий підхід дозволить створити систему, здатну стабільно працювати з великими обсягами даних і задовольняти потреби користувачів у своєчасному отриманні актуальної інформації.

Враховуючи зростання конкуренції на ринку електронної торгівлі, своєчасне отримання достовірної інформації про динаміку цін, наявність продукції та тенденції споживчого попиту має стратегічне значення. Саме тому розробка інформаційної системи автоматизованого збору даних є актуальним завданням, що має як практичну, так і дослідницьку цінність.

Отже, метою даної роботи є створення системи, яка надає можливість користувачеві налаштовувати процес збору даних з вебресурсів, переглядати зібрану інформацію та отримувати її в зручній для подальшої аналітики формі. Реалізація такої системи сприятиме підвищенню ефективності роботи з інформаційними ресурсами та розширить можливості аналізу ринку електронної комерції.

1 ОПИС ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Інформаційна система автоматизованого збору даних з вебресурсів повинна забезпечувати автоматизацію процесу збору та зберігання інформації з інтернет-магазинів без необхідності володіння навичками програмування, надаючи користувачам можливість самостійно налаштовувати, які дані необхідно зібрати. Система повинна підтримувати роботу як зі статичними, так і з динамічними вебсторінками, автоматично визначаючи оптимальний метод збору даних.

Важливою функцією системи є можливість налаштування періодичності автоматичного збору даних. Зібрані дані повинні зберігатись з можливістю перегляду історії попередніх зборів даних та завантаження файлів із зібраною інформацією.

Для керування процесом збору необхідно реалізувати зручний інтерфейс, який дозволить користувачам створювати, редагувати та видаляти шаблони.

1.1 Мета та завдання магістерської дисертації

Основною метою цієї роботи є створення інформаційної системи автоматизованого збору даних з інтернет-магазинів, яка б надавала користувачам можливість ефективно збирати, структурувати та аналізувати інформацію про товари без необхідності володіння навичками програмування. Система повинна забезпечити гнучкість налаштування процесу збору даних, автоматизацію регулярного оновлення інформації та зручні інструменти для роботи із зібраними даними.

Проаналізувавши поставлену задачу, можна виділити наступні завдання для вирішення:

— інтуїтивно зрозумілий інтерфейс для налаштування автоматизованого збору даних;

— можливість налаштування переліку полів даних для збору з кожного джерела;

— система управління джерелами даних з можливістю додавання, редагування та видалення;

— підтримка роботи як зі статичними, так і з динамічними вебсторінками;

— можливість налаштування періодичності автоматичного збору даних;

— структуроване зберігання зібраної інформації;

— експорт даних у різних форматах (Excel, CSV, JSON, Google Sheets, SQL, TXT та TSV) для подальшої обробки.

1.2 Актуальність інформаційної системи

У сучасному цифровому світі обсяг інформації, що публікується в інтернет-магазинах, зростає експоненційно. Щодня у світі створюється близько 402,74 мільйона терабайтів даних [1]. За прогнозами, до кінця 2025 року цей показник зросте до 181 зетабайта [1], значна частина яких припадає на сектор електронної комерції. Інтернет-магазини постійно оновлюють асортимент товарів, змінюють ціни, публікують нові характеристики продукції та відгуки покупців. Для бізнес-аналітики, моніторингу конкурентів, порівняння цін та прийняття стратегічних рішень необхідний швидкий та структурований доступ до цієї інформації.

Традиційний підхід до збору інформації з інтернет-магазинів передбачає ручне копіювання даних, що є надзвичайно трудомістким та неефективним процесом. Для аналізу навіть невеликої вибірки товарів з декількох джерел може знадобитися кілька годин або днів роботи. При цьому високий ризик людської помилки, непослідовності у форматуванні даних та неможливість забезпечити регулярне оновлення інформації робить такий підхід непридатним для серйозних бізнес-застосувань. Крім того, ручний збір даних не дозволяє швидко реагувати на зміни ринкової ситуації, що є критичним у конкурентному середовищі електронної комерції.

Сучасні компанії потребують автоматизованих рішень для збору та аналізу даних конкурентів. Маркетологи використовують інформацію про ціни для динамічного ціноутворення, менеджери з продажу аналізують асортимент

конкурентів для оптимізації власного каталогу, аналітики досліджують тренди ринку на основі доступності товарів та коливань цін. Відсутність ефективних інструментів для автоматизованого збору даних призводить до втрати конкурентних переваг та прийняття рішень на основі застарілої інформації. Особливо гостро ця проблема постає для малого та середнього бізнесу, який не має значних ресурсів для найму окремих спеціалістів з аналітики даних.

Актуальність розробки такої системи підтверджується зростаючим попитом на рішення з автоматизації бізнес-процесів у сфері електронної комерції. Компанії, які мають доступ до якісних та актуальних даних про ринок, отримують значні конкурентні переваги. Можливість швидко реагувати на зміни цін конкурентів, аналізувати попит та оптимізувати асортимент безпосередньо впливає на прибутковість бізнесу. У цьому контексті інформаційна система автоматизованого збору даних стає не просто зручним інструментом, а необхідною складовою успішної діяльності сучасної компанії у сфері електронної комерції.

1.3 Загальний опис предметної області

Інтернет-магазини є одним з найважливіших джерел комерційної інформації в сучасному цифровому середовищі. Вони містять великі масиви структурованих даних про товари, їх характеристики, ціни, наявність на складі, відгуки покупців та інші важливі параметри. Автоматизований збір таких даних дозволяє вирішувати різноманітні бізнес-задачі: моніторинг цінової політики конкурентів, аналіз ринкових трендів, формування власних товарних каталогів, дослідження споживчого попиту та оптимізація асортименту.

Однією з переваг автоматизованого збору даних є економія часу та людських ресурсів, оскільки система може обробляти тисячі вебсторінок за короткий проміжок часу без втручання людини. Це дає змогу використовувати вебскрепінг для виконання комплексних завдань, таких як моніторинг цін у режимі реального часу, відстеження змін в асортименті товарів, аналіз сезонних

коливань попиту, збір інформації про нові продукти на ринку та формування баз даних для машинного навчання.

Системи автоматизованого збору даних мають особливо важливе значення для бізнесу в умовах високої конкуренції на ринку електронної комерції. Вони використовуються як для безпосереднього аналізу конкурентного середовища, так і для ведення моніторингу власного позиціонування на ринку, для оптимізації цінової стратегії, аналітичної підтримки та прийняття управлінських рішень. Так, наприклад, в сучасній роздрібній торгівлі почали широко застосовуватись системи динамічного ціноутворення на основі даних вебскрепінгу. Вони використовуються для відстеження цін конкурентів, аналізу попиту, моніторингу змін в асортименті, виявлення цінових аномалій, а також для надання актуальної інформації бізнес-аналітикам у реальному часі.

1.4 Аналіз методів збору даних з вебресурсів

Існує декілька підходів до отримання інформації з інтернет-магазинів та інших вебсайтів, кожен з яких має свої переваги, обмеження та сфери застосування. Вибір конкретного методу залежить від технічних характеристик цільового вебресурсу, обсягу даних, що потрібно зібрати, вимог до швидкості обробки та доступних ресурсів системи.

У цьому розділі проводиться аналіз основних методів збору даних з вебресурсів, їх технічних особливостей та можливостей застосування для розробки інформаційної системи автоматизованого збору даних з інтернет-магазинів.

1.4.1 Використання публічних API

Прикладний програмний інтерфейс (Application Programming Interface — API) — це програмний інтерфейс, який дозволяє різним програмним системам взаємодіяти між собою за визначеними правилами. Багато великих інтернет-

магазинів та торгових платформ надають публічні або партнерські API для доступу до своїх даних.

Використання API має цілу низку суттєвих переваг. По-перше, дані надаються у структурованому вигляді, оскільки інформація передається у стандартизованих форматах, таких як JSON або XML, що значно спрощує її подальшу обробку. По-друге, API розробляється та підтримується безпосередньо власниками ресурсу, що гарантує офіційну підтримку та стабільність роботи системи. Також цей метод забезпечує високу швидкість отримання даних, адже відсутня необхідність завантажувати та шукати дані в HTML-коді сторінок. Крім того, API зазвичай має детально описану документацію, яка істотно спрощує процес інтеграції. Врешті-решт, використання API є офіційним та юридично легітимним способом отримання даних, що уникає правових ускладнень.

Водночас метод має недоліки, які варто враховувати. Головною проблемою є обмежена доступність, оскільки більшість інтернет-магазинів взагалі не надають публічного API або обмежують доступ для всіх окрім офіційних партнерів. Навіть за наявності API часто існують ліміти на кількість запитів за певний проміжок часу, що може обмежувати масштаби збору даних. Також через API може бути доступна лише частина інформації порівняно з тією, що відображається безпосередньо на сайті. Для роботи з API зазвичай необхідна попередня реєстрація та отримання спеціальних ключів доступу, що ускладнює початкову інтеграцію. Окремі комерційні API можуть вимагати оплати за використання, що робить цей метод фінансово витратним для деяких проектів.

Для реалізації системи автоматизованого збору даних використання API є найбільш бажаним варіантом, проте через обмежену доступність такого інтерфейсу в більшості інтернет-магазинів, цей метод не може бути застосовуваний.

1.4.2 Збір даних зі статичного HTML-контенту

Збір даних зі статичного HTML-контенту полягає в аналізі HTML-коду вебсторінок, який повністю формується на сервері та не змінюється після завантаження. Цей метод передбачає завантаження HTML-документа та витягування необхідної інформації за допомогою аналізу DOM-структури.

З технічного боку метод реалізується через кілька послідовних кроків. Спочатку використовуються спеціалізовані HTTP-бібліотеки, такі як requests, urllib або axios, для завантаження HTML-коду сторінки. Після цього застосовуються парсери HTML, зокрема BeautifulSoup, lxml або Cheerio, які будують DOM-дерево документа. Далі здійснюється пошук потрібних елементів за допомогою CSS-селекторів або XPath-виразів. На завершальному етапі відбувається витягування текстового вмісту, атрибутів та інших необхідних даних із знайдених елементів.

Цей підхід має низку вагомих переваг. Найголовнішою є простота реалізації, адже не потрібні складні інструменти для емуляції повноцінного браузера. Метод забезпечує високу швидкість обробки завдяки відсутності необхідності виконувати JavaScript-код. Також він характеризується низьким споживанням ресурсів, оскільки пошук та збір даних з HTML вимагає значно менше оперативної пам'яті та процесорного часу порівняно з іншими методами. Стабільність роботи досягається завдяки тому, що статичний контент не змінюється після завантаження сторінки, що суттєво полегшує процес відлагодження. Крім того, метод відзначається хорошою масштабованістю, що дозволяє одночасно обробляти велику кількість сторінок без значних витрат ресурсів.

Проте існують і певні недоліки, які обмежують сферу застосування методу. Головним обмеженням є те, що він працює виключно з сайтами, які використовують серверний рендеринг HTML. Метод не надає доступу до динамічного контенту, тобто неможливо отримати дані, які завантажуються асинхронно через AJAX-запити після початкового завантаження сторінки. Існує

сильна залежність від структури HTML-розмітки, що означає необхідність оновлення CSS-селекторів або XPath-виразів при кожній зміні дизайну чи структури сайту.

Цей метод є оптимальним для збору даних з традиційних інтернет-магазинів, які генерують повний HTML-код на сервері. Він забезпечує високу продуктивність та ефективність при роботі з великими обсягами даних.

1.4.3 Збір даних із динамічного контенту з використанням headless-браузерів

Сучасні вебдодатки часто використовують JavaScript-фреймворки для динамічного формування контенту на стороні клієнта. У таких випадках розмітка, отримана при первинному завантаженні сторінки, не містить всієї необхідної інформації. Для роботи з такими ресурсами використовуються headless-браузери — повноцінні веббраузери без графічного інтерфейсу.

Існує кілька основних інструментів для роботи з динамічним контентом. Selenium є бібліотекою для автоматизації браузерів, яка найчастіше використовується для тестування вебдодатків, і для роботи з кожним з основних браузерів потрібно завантажити додаткові компоненти - драйвери для Chrome, Firefox, Microsoft IE та Edge є окремими виконуваними файлами [2]. Playwright - це бібліотека автоматизації з відкритим кодом для тестування браузерів та вескрепінгу [3].

Метод використання headless-браузерів має численні переваги. Headless-браузери безперешкодно інтегруються з інструментами автоматизації, що полегшує автоматизацію взаємодії з вебсайтами та витягування даних. Вебсайти, які використовують JavaScript для відмальовування контенту, часто не можуть бути оброблені HTML-парсерами [4]. Headless-браузери можуть легко обробляти такі сайти, гарантуючи збір всього динамічного контенту [4]. Це забезпечує повний доступ до контенту, оскільки можна отримати всі дані, які бачить звичайний користувач браузера. Важливою особливістю є можливість взаємодії з

елементами сторінки, що дозволяє імітувати реальні дії користувача, такі як натискання миші, прокрутка або введення тексту. Крім того, headless-браузер здатен обходити прості системи захисту, оскільки для вебсервера він виглядає як звичайний відвідувач.

Існують також і суттєві недоліки, які необхідно враховувати. Головною проблемою є висока витрата ресурсів, адже запуск повноцінного браузера вимагає значних обчислювальних потужностей та оперативної пам'яті. Метод характеризується низькою швидкістю роботи, оскільки виконання JavaScript-коду та відмальовування сторінки займає набагато більше часу порівняно із обробкою статичної розмітки.

Для розробки інформаційної системи автоматизованого збору даних цей метод є необхідним для підтримки роботи з сучасними інтернет-магазинами, що використовують JavaScript-фреймворки. Рекомендується використовувати Selenium завдяки його широкій поширеності, великій спільноті розробників, наявності численних бібліотек для різних мов програмування та можливості роботи з усіма популярними браузерами. Бібліотека Selenium також має добре розписану документацію та велику кількість готових рішень для поширених задач автоматизації збору даних.

1.4.4 Гібридний підхід до збору даних

Гібридний підхід передбачає комбінування різних методів збору даних залежно від характеристик конкретного вебресурсу. Система автоматично визначає тип контенту та обирає оптимальний спосіб обробки.

Принципи роботи гібридної системи базуються на послідовному аналізі та виборі найефективнішого методу для конкретної задачі. Спочатку відбувається первинний аналіз вебресурсу для визначення типу контенту та способу його формування. Якщо сайт використовує серверний рендеринг і всі дані доступні в статичній розмітці, то застосовується простий збір даних зі статичних сторінок за допомогою HTTP-бібліотек. У випадку, коли вміст сторінки формується

динамічно, тобто в браузері користувача, то система автоматично переключається на використання headless-браузера. Також подібні системи можуть здійснювати кешування результатів для зменшення кількості повторних запитів.

Гібридний підхід має низку переваг, порівняно з використанням одного методу збору даних з вебсторінок. Найважливішою є оптимальна продуктивність, оскільки для кожного ресурсу застосовується найбільш ефективний метод обробки. Система володіє високою універсальністю, що дозволяє їй працювати з різними типами вебсайтів. Також досягається значна економія обчислювальних ресурсів, тому що ресурсомісткий headless-браузер використовується виключно за необхідності, лише у випадку, коли збір даних зі статичних сторінок не може забезпечити отримання потрібних даних. Крім того, подібні системи показують високу адаптивність, автоматично перемикаючись між методами при виявленні змін у структурі або способі формування контенту на сайті. Також забезпечується вищий рівень надійності, завдяки наявності кількох варіантів збору даних одночасно.

Проте існують і певні недоліки, пов'язані зі складністю реалізації такого рішення, оскільки потрібно реалізувати логіку вибору методу, механізми автоматичного перемикання та надійну систему обробки помилок. Необхідність підтримки декількох технологій призводить до збільшення обсягу програмного коду та кількості залежностей проєкту. До того ж, можуть виникати конфлікти між різними форматами даних, оскільки різні методи можуть повертати дані в різних структурах, що вимагає додаткової обробки результатів.

Отже, гібридний підхід, що поєднує збір даних зі статичної розмітки та використання headless-браузерів, є найбільш оптимальним рішенням для інформаційної системи автоматизованого збору даних про товари в інтернет-магазинах. Він дозволяє забезпечити баланс між продуктивністю та універсальністю, що є вкрай необхідним для ефективної роботи з різноманітними вебресурсами, які використовують як, традиційний серверний, так і клієнтський рендеринг вебсторінок.

На основі проведеного аналізу для реалізації інформаційної системи автоматизованого збору даних з інтернет-магазинів обрано гібридний підхід, що включає:

— пріоритетне використання збору даних зі статичного HTML, що забезпечить максимальну продуктивність при роботі з традиційними інтернет-магазинами;

— підтримку роботи з динамічним контентом, використовуючи бібліотеку Selenium, для забезпечення можливості збору даних зі сторінок із динамічним завантаженням контенту;

— автоматичне визначення типу контенту, тобто система буде аналізувати структуру сторінки та автоматично обирати оптимальний метод збору даних, що забезпечить оптимальне використання ресурсів, при цьому зберігаючи універсальність та якість зібраних даних.

Такий підхід дозволить створити гнучку та ефективну систему, здатну працювати з різноманітними вебресурсами, забезпечуючи при цьому оптимальне використання обчислювальних ресурсів та високу швидкість обробки даних.

1.5 Огляд існуючих систем

Аналіз існуючих систем автоматизованого збору даних виявляє кілька спільних недоліків, які обмежують їх застосування для широкого кола користувачів. По-перше, більшість комерційних рішень (ParseHub, Import.io) використовують модель монетизації, за якої ключові функції, такі як автоматизований періодичний збір даних, хмарне зберігання та розширені можливості експорту, доступні лише в платних тарифах вартістю від п’ятдесяти до п’ятиста доларів на місяць, що робить такі системи недоступними для малого бізнесу.

Крім того, існуючі рішення не завжди надають достатню гнучкість у налаштуванні процесу збору даних — користувачі або обмежені функціональними можливостями інтерфейсів користувача (ParseHub, Import.io),

або змушені працювати безпосередньо з кодом (Scrapy Cloud), що вимагає навичок програмування та ускладнює використання таких систем для нетехнічних користувачів. До того ж, багато систем мають проблеми зі складністю використання, оскільки налаштування збору даних із вебсайтів з динамічним відмальовуванням контенту часто вимагає технічних знань або звернення до служби підтримки.

Створена в цій роботі система отримала назву «UniScraper». В таблиці 1.1 наведено порівняння систем автоматизованого збору даних ParseHub [5], Scrapy Cloud [6], Import.io [7] та UniScraper.

Таблиця 1.1 — Порівняння функціональних можливостей створюваної системи із існуючими системами

	ParseHub	Scrapy Cloud	Import.io	UniScraper
Інтуїтивно зрозумілий інтерфейс для налаштування збору даних	+	-	+	+
Можливість налаштування переліку полів даних для збору	+	+	+	+
Система управління джерелами даних	+	+/-	+/-	+
Підтримка роботи зі статичними та динамічними вебсторінками	+	+	+/-	+
Налаштування періодичності автоматичного збору даних	+	+	+	+
Структуроване зберігання зібраної інформації	+	+	+	+
Експорт даних у різних форматах	+	+	+	+

Отже, проведене порівняння функціональних можливостей демонструє, що створювана система UniScraper не має суттєвих технологічних переваг над вже існуючими рішеннями. Більшість базових можливостей, необхідних для ефективного збору даних з вебресурсів, вже реалізовані в подібних системах.

Проте ключовою перевагою UniScraper є її орієнтація на український ринок, що дозволяє більш точно врахувати локальні особливості, специфіку українських вебресурсів, мовні нюанси, правові аспекти та потреби вітчизняних користувачів.

Висновки до розділу 1

У цьому розділі було проведено аналіз предметної області, у межах якого було визначено мету та завдання даної роботи. Також було обґрунтовано актуальність створення системи автоматизованого збору даних та проведено загальний опис предметної області, що дозволило визначити основні процеси та об’єкти, які підлягатимуть автоматизації.

Крім того, було проведено аналіз предметної області, що дозволило визначити ключову роль даних у сучасних сферах діяльності та окреслити основні підходи до їх отримання з вебресурсів. Також було розглянуто різні методи збору інформації з вебресурсів.

Аналіз існуючих систем продемонстрував різноманіття доступних інструментів і платформ, підкресливши важливість правильного вибору методів з урахуванням специфіки поставлених завдань. Отримані результати слугують основою для формування вимог до майбутньої інформаційної системи та обґрунтування вибору відповідних технічних рішень.

2 ЗАСОБИ РОЗРОБКИ

В цьому розділі описано засоби та методи, що використовувалися під час розробки інформаційної системи автоматизованого збору даних. Використані технології визначають продуктивність, масштабованість і надійність системи, а також впливають на можливість подальшого розширення її функціоналу, а отже, правильність їх вибору є критично важливою.

2.1 Вибір архітектури системи

Правильне визначення архітектури системи дає змогу сформувати ключові вимоги й обмеження проєкту, що у подальшому полегшує вибір потрібних технологій та засобів розробки. Це безпосередньо впливає на продуктивність, масштабованість і тривалість життєвого циклу продукту, тому є одним із найважливіших етапів створення програмного забезпечення.

Далі будуть розглянуті такі варіанти побудови архітектури вебдодатків, як клієнт-серверний, монолітний та мікросервісний підходи.

2.1.1 Клієнт-сервер

Клієнт-серверна архітектура — це модель обчислювальної системи, в якій велика кількість клієнтів (віддалених процесорів) надсилає запити до централізованого сервера (комп’ютера-хоста), який обслуговує ці запити. Клієнти забезпечують інтерфейс для користувачів, щоб вони могли надсилати запити серверу і отримувати результати його відповідей. Сервер очікує на запити від клієнтів і потім відповідає на них [8].

Однією з основних переваг клієнт-серверної моделі є її централізація. Наявність центрального сервера спрощує керування та оновлення даних і послуг. Ця модель також забезпечує покращену масштабованість, оскільки можна додавати більше клієнтів, не впливаючи на продуктивність сервера. Вищий рівень

безпеки — це ще одна перевага, оскільки всі дані зберігаються на стороні сервера [9].

З іншого боку, недоліки моделі клієнт-сервер включають «єдину точку відмови»: якщо сервер перестає працювати, потерпає вся мережа. Також велика кількість запитів може спричинити перевантаження сервера й знизити продуктивність [9].

З архітектурної точки зору, клієнт-серверна модель забезпечує чіткий поділ ролей: сервери керують ресурсами, а клієнти відповідають за інтерфейс користувача. Така сегрегація робить розробку і підтримку системи простішими [9].

2.1.2 Монолітна

Монолітна архітектура — це стиль побудови програмного забезпечення, у якому всі компоненти додатку, включно з інтерфейсом користувача, бізнес-логікою та доступом до даних, об’єднані в одну єдину кодову базу та розгортаються як один блок. Такий підхід забезпечує простоту розробки і тестування на початкових етапах, проте складність масштабування та внесення змін зростає зі збільшенням розміру системи.

Коли додаток створюється з однією кодовою базою, розробляти його простіше. Розгортання легше — лише один виконуваний файл або директорія, що спрощує процес. Завдяки централізованій кодовій базі один API часто може виконувати ті ж функції, які в мікросервісах розподілені між багатьма API, що покращує продуктивність. Тестування спрощене, оскільки додаток — це єдиний блок, енд-ту-енд тести можна проводити швидше, ніж у розподілених системах. Дебаг також легший — весь код знаходиться в одному місці, тому простіше простежити запит і знайти проблему [10].

Проте у великому монолітному додатку розробка стає складнішою і повільнішою. Неможливо масштабувати окремі компоненти — для масштабування потрібно дублювати весь додаток. Якщо в будь-якому модулі

виникає помилка, це може вплинути на доступність усього додатку. Зміни у фреймворку або мові програмування зачіпають весь додаток, що робить оновлення дорогим і трудомістким. Навіть невелика зміна вимагає повторного розгортання всього моноліту [10].

2.1.3 Мікросервіси

Мікросервісна архітектура (скорочено «мікросервіси») — це архітектурний стиль для розробки додатків, який дозволяє поділяти велику систему на менші незалежні частини, кожна з яких має свою сферу відповідальності. Щоб обробити один запит від користувача, додаток на мікросервісах може викликати багато внутрішніх сервісів, які спільно формують відповідь [11].

Кожен мікросервіс — це окремий сервіс, призначений для реалізації певної функції додатку або виконання конкретної бізнес-завдання. Мікросервіси між собою комунікують через прості інтерфейси (API), вирішуючи бізнес-логіку кожного окремого аспекту системи [11]. Мікросервісна архітектура має наступні переваги:

— високий рівень гнучкості завдяки самостійності кожного сервісу;

— легке масштабування окремих частин системи;

— можливість окремого оновлення та розробки сервісів;

— помилки одного сервісу не впливають на роботу інших;

— використання найбільш придатних технологій для кожного модуля.

Недоліки мікросервісної архітектури:

— складність розробки і підтримки: велика кількість сервісів потребує управління комунікацією між ними;

— мережеві виклики між сервісами можуть впливати на продуктивність;

— складніше відлагоджувати і тестувати систему в цілому;

— потребує належної організації логування, моніторингу та безпеки;

— може зростати складність розгортання через велику кількість автономних компонентів.

Мікросервісна архітектура підходить для великих і динамічних додатків, де важлива масштабованість, автономність команд і швидке оновлення окремих функцій. Вона забезпечує більшу гнучкість і стійкість, ніж монолітна архітектура, але водночас вимагає ретельного планування, управління сервісами та інфраструктурної підтримки.

Врешті решт, для розробки інформаційної системи автоматизованого збору даних про товари з інтернет-магазинів було обрано клієнт-серверну архітектуру, оскільки вона забезпечує централізоване управління даними, спрощує адміністрування та підтримку системи, дозволяє одночасно обслуговувати велику кількість користувачів і гарантує підвищений рівень безпеки інформації. Такий підхід оптимально відповідає вимогам ефективності, масштабованості та надійності обраної системи.

2.2 Вибір мови програмування

Для створення серверної частини інформаційної системи обрано мову програмування Python з використання фреймворку Django.

Мова програмування Python є інтерпретованою, об’єктно орієнтованою мовою високого рівня, що характеризується динамічною типізацією.

Мова програмування Python вирізняється розвиненим набором стандартних бібліотек, що охоплює широкий спектр модулів для розв’язання практичних і науково-технічних завдань, зокрема для роботи з файловими системами, мережевими протоколами, обробки даних та виконання чисельних обчислень. Важливою особливістю мови є лаконічний та легко інтерпретований синтаксис, який комплексних програмних продуктів. Для управління пам'яттю мова програмування Python використовує динамічну типізацію та комбінацію підрахунку посилань і сміттєзбирача з виявленням циклів [12].

Також мова програмування Python є однією з найпопулярніших мов для розробки бекенду для сучасних проектів. Однією з її головних переваг є наявність великої кількості фреймворків для створення бекенду, які значно спрощують

розробку вебдодатків та сервісів. Серед найбільш відомих фреймворків варто відзначити Django, Flask та FastAPI, які дозволяють швидко створювати REST-сервіси та прототипи вебдодатків. Використання таких фреймворків дозволяє скоротити час розробки, підвищити стабільність і безпеку системи та забезпечує можливість масштабування проєкту у майбутньому.

До галузей, в яких застосовується мова програмування Python входять веброзробка, наукові обчислення, освіта, додатки з графічним інтерфейсом користувача, розробка програмного забезпечення, бізнес-додатки, інтернет речей, штучний інтелект та машинне навчання тощо [13].

Окрім того, мова програмування Python активно використовується у сфері обробки даних та вебскрепінгу завдяки великій кількості спеціалізованих бібліотек. Для отримання даних із вебресурсів широко застосовуються бібліотеки requests для роботи з HTTP-запитами та BeautifulSoup або lxml для парсингу HTML і XML документів. Для більш складних випадків, коли потрібно взаємодіяти з динамічними вебсторінками, використовуються інструменти на зразок Selenium або Playwright, що дозволяють автоматизувати роботу браузера.

Таким чином, Python поєднує корисні засоби для розробки бекенду та ефективні інструменти для збору й обробки даних із вебресурсів. Ця універсальність робить його оптимальним вибором для проєктів, які потребують поєднання серверної логіки та автоматизованого збору інформації з різних джерел.

Для створення клієнтської частини інформаційної системи було обрано мову програмування TypeScript з використання відповідного фреймворку.

Мова програмування TypeScript є високорівневою мовою програмування, яка додає статичну типізацію з опціональними анотаціями типів до JavaScript. Вона розроблена для створення великих додатків і транслюється в JavaScript [14]. Оскільки TypeScript є надбудовою до мови програмування JavaScript, існуючий код JavaScript може бути адаптований до TypeScript, і програми TypeScript можуть безперешкодно використовувати програми, написані на JavaScript [14].

Однією з ключових особливостей TypeScript є його система статичної типізації. Це означає, що коли ви пишете код на TypeScript, ви оголошуєте тип кожної змінної, функції чи об'єкта, що полегшує виявлення помилок та запобігання багам [15]. Система типізації TypeScript виявляє помилки на ранньому етапі під час компіляції, а не під час виконання, зменшуючи ймовірність того, що баги потраплять у продакшн. Раннє виявлення помилок у середовищах розробки, покращує управління кодом.

Також TypeScript широко використовується в розробці фронтенд вебдодатків. Мова TypeScript може використовуватися з популярними JavaScript фреймворками та бібліотеками, такими як React, Angular та Vue.js [16].

Фреймворки, такі як Angular та React, пропонують нативну або загальну підтримку TypeScript, що полегшує створення інтерактивних та масштабованих вебдодатків [17]. Інтелектуальне автодоповнення коду TypeScript та багата підтримка інструментів розробки (завдяки Microsoft Visual Studio Code) значно підвищують продуктивність розробників. Використовуючи TypeScript, можна отримати такі функції, як автоматичні підказки, навігація по коду та перевірка помилок у режимі реального часу, що робить програмування швидшим та приємнішим [18]. Розглянемо основні переваги мови програмування TypeScript [19]:

— виявлення помилок під час компіляції, оскільки TypeScript ідентифікує помилки на етапі компіляції, тоді як JavaScript виявляє їх під час виконання;

— використання опціональної статичної типізації, що дозволяє додавати типи до змінних, функцій, властивостей тощо;

— наявність підтримки інструментів розробки, які пропонують підказки в режимі реального часу під час написання коду;

— структурування коду, що допомагає ефективно організовувати програму;

— підтримка просторів імен, шляхом визначення модулів;

— підтримка використання інтерфейсів;

— документація API, що потенційно зменшує кількість помилок.

Проте TypeScript має і ряд недоліків. Для розробників, які знайомі з чистим JavaScript, може існувати потреба в перенавчанні при переході на TypeScript. Необхідно розуміти анотації типів, інтерфейси та інші специфічні для TypeScript функції [20]. TypeScript вводить передові концепції програмування, такі як «generics», інтерфейси, переліки (enums) та складні визначення типів, які не є частиною стандартного JavaScript. Для молодших розробників або команд, які переходять з чистого JavaScript, розуміння цих концепцій може стати перешкодою [21]. Код TypeScript повинен бути трансльований у JavaScript перед запуском у браузерах. Додатковий крок компіляції додає час до процесу розробки [20]. Крім того, компіляція вимагає додаткового кроку для перетворення TypeScript на JavaScript для виконання у браузері. Навчання розробників TypeScript може бути складним завданням. Команди, які володіють JavaScript, можуть потребувати близько 2-3 місяців, щоб стати продуктивними з TypeScript, і близько шести місяців, щоб досягти вільного володіння мовою [19]. Хоча екосистема TypeScript зростає, вона не така обширна, як у JavaScript. Деякі сторонні бібліотеки можуть не мати визначень типів TypeScript, що може вимагати додаткової роботи для інтеграції з окремим TypeScript проектом [20]. Покращена читабельність коду, забезпечена згаданими раніше синтаксичними покращеннями та анотаціями типів, має свої недоліки. По-перше, потрібно писати більше коду, і це може сповільнити процес розробки [22].

Отже, TypeScript є потужним інструментом для веброзробки, який додає статичну типізацію та розширені можливості до JavaScript. Також TypeScript надає особливо значні переваги в середніх і великих проєктах та є особливо корисним у галузях, орієнтованих на дані, таких як фінанси, управління даними про продукти тощо. Мова TypeScript представляє еволюцію у світі розробки програмного забезпечення, надаючи систему статичної типізації, яка підвищує безпеку та якість коду. Незважаючи на необхідність трансляції в JavaScript, його велика спільнота та зростання популярності роблять його хорошим вибором для розробників, які шукають більш надійний та ефективний спосіб написання коду на JavaScript.

2.3 Вибір фреймворків та бібліотек

Для розроблення сучасних вебсистем застосовуються фреймворки, що забезпечують формальну архітектурну основу програмного проєкту та пропонують набір інтегрованих інструментів і сервісів. Ці засоби можуть бути розширені й налаштовані відповідно до функціональних і нефункціональних вимог конкретного проєкту, що сприяє підвищенню ефективності та стандартизації процесу розробки. Окрім фреймворків, важливу роль відіграють спеціалізовані бібліотеки, які забезпечують готовий інструментарій для виконання окремих завдань — від роботи з мережевими запитами до взаємодії з базами даних чи реалізації складних алгоритмів. Використання фреймворків і бібліотек значно прискорює розробку, підвищує надійність системи та спрощує підтримку програмного забезпечення.

Крім того, сучасні інформаційні вебсистеми потребують комплексного підходу до забезпечення безпеки, який охоплює технічні, організаційні та правові аспекти захисту інформації [23]. Моделювання інформаційної безпеки в вебсистемах є основою для проєктування ефективних механізмів захисту та забезпечує можливість прогнозування поведінки системи під впливом різних загроз [24]. Впровадження систем захисту інформаційних ресурсів на базі сучасних технологій забезпечує надійний захист вебзастосунків від актуальних загроз [25].

2.3.1 Серверна частина

Для реалізації серверної частини проєкту було обрано фреймворк Django. Django є високорівневим вебфреймворком для мови програмування Python, розробленим з метою прискорення процесу створення вебзастосунків та забезпечення дотримання принципів чистого й структурованого програмного коду. Він реалізує архітектуру «модель–шаблон–представлення» (Model-Template-View), що дозволяє чітко розмежувати логіку бізнес-процесів, інтерфейс

користувача та роботу з даними. Завдяки цьому підходу забезпечується легка підтримуваність, масштабованість і повторне використання компонентів проєкту.

Однією з ключових особливостей фреймворку Django є його орієнтація на автоматизацію рутинних завдань, що часто зустрічаються у веброзробці. Серед таких завдань — робота з базами даних через вбудовану ORM (Object-Relational Mapping), формування запитів, валідація даних, управління міграціями баз даних, а також формування адміністративної панелі для керування інформацією. Це дозволяє розробникам зосередитися на реалізації унікальної логіки застосунку, замість того щоб витрачати час на реалізацію повторюваних операцій. Крім того, Django має потужну систему маршрутизації, яка забезпечує гнучку обробку URL та запитів.

Ефективне управління інформаційними ризиками є критично важливим для забезпечення надійності вебсистем [26]. Тож однією важливою характеристикою фреймворку є вбудована система безпеки. Django автоматично захищає від багатьох поширених вебзагроз, таких як SQL-ін’єкції, міжсайтовий скриптинг (XSS), підробка запитів між сайтами (CSRF) та інші. Це дозволяє значно знизити ризик вразливостей на ранніх етапах розробки. Фреймворк також підтримує модульну систему додатків, що сприяє організації великого проєкту у вигляді незалежних компонентів, кожен з яких може бути повторно використаний або розширений. Як зазначають дослідники, вразливості вебдодатків переважають за кількістю та потенційною шкодою порівняно з іншими проблемами інформаційної безпеки, а більшість зовнішніх атак на корпоративні інформаційні системи спрямовані саме на експлуатацію вразливостей веб-додатків [27]. Тому системи аутентифікації та механізми захисту є критично важливими компонентами захисту інформаційних ресурсів у веб-середовищі [28]. Інформаційна безпека забезпечується підтримкою конфіденційності, цілісності та доступності інформаційних ресурсів системи [29].

Особливості Django включають також підтримку шаблонів для формування HTML-сторінок, інтернаціоналізацію та локалізацію, що дозволяє створювати застосунки для різних мов і регіонів. а також багатий набір готових до

використання бібліотек для обробки сесій, автентифікації користувачів, кешування та роботи з файлами. Оптимізація системи захисту інформації вимагає врахування специфіки архітектури та функціональних вимог конкретного проєкту [30]. Завдяки активній спільноті та постійному розвитку фреймворку, розробники мають доступ до численних розширень, що дозволяє легко інтегрувати додатковий функціонал. Переваги використання фреймворку Django:

— вбудована система безпеки, що захищає від найпоширеніших вебзагроз;

— наявність ORM, яка забезпечує зручну взаємодію з базами даних без написання сирих SQL-запитів;

— автоматично згенерована адміністративна панель для керування даними;

— масштабованість і придатність для великих проєктів;

— структурована архітектура, що полегшує розробку, супровід і тестування;

— модульна система додатків, що дозволяє повторно використовувати компоненти;

— підтримка шаблонів, інтернаціоналізації та локалізації;

— багатий набір готових бібліотек для сесій, автентифікації та роботи з файлами.

Недоліки використання фреймворку Django:

— відносно високий поріг входу для початківців через складність внутрішньої структури;

— надмірність у випадках дуже малих або вузькоспеціалізованих застосунків;

— менш гнучкий підхід порівняно з іншими фреймворками, що може обмежувати можливість налаштування проєкту в окремих випадках;

— потенційна потреба у додатковому налаштуванні для проєктів із високим навантаженням.

Як було зазначено раніше, початково архітектура Django проєкту побудована на основі паттерну проєктування «Модель-Шаблон-Представлення» (MTV — «Model-Template-View»), який є адаптацією паттерну проєктування

Модель-Представлення-Контролер (MVC — «Model-View-Controller»). До того ж, архітектура Django-проєкту можна змінити за необхідності.

«Модель-Представлення-Контролер» — це паттерн проєктування архітектури, який розділяє створювану програму на три основні компоненти: модель, представлення та контролер. Цей підхід надає ряд переваг [31]:

— розділення бізнес-логіки, логіки інтерфейсу користувача та логіки введення від користувача;

— повний контроль над HTML і URL-адресами, що полегшує розробку архітектури вебдодатків;

— можливість легко замінювати компоненти програми завдяки їх ізольованості один від одного;

— підтримка розробки за тестами, написаними до створення самої програми.

Модель — це компонент, який відповідає за виконання завдань, пов’язаних з даними (наприклад, таблиця в базі даних, JSON-файл тощо). Компонент «модель» бере дані з місця зберігання, виконує їх обробку та передає оброблені дані в представлення [32]. Тобто цей елемент відповідає за бізнес логіку додатку.

Представлення — це компонент, який використовується для відображення даних із компонента «модель». Він також може бути використаний для отримання даних від користувачів (за допомогою форм) та для передачі отриманих даних в компонент «модель» [32].

Контролер — це програмний код, який з’єднує компоненти «представлення» та «модель». Він також визначає, як відображати представлення та як реагувати на дані, введені користувачем [32].

Фреймворк Django використовує модифіковану версію класичної архітектури MVC, де традиційні компоненти отримали нові назви: те, що в MVC називається Контролером, у Django виступає як «представлення» (view), а класичне «представлення» перетворилося на «шаблон» (template). Попри таку зміну термінології, фундаментальні принципи архітектурного патерну залишаються незмінними.

Архітектура MTV у Django забезпечує наступні ключові переваги:

— кожен компонент системи має власну зону відповідальності;

— код пишеться один раз і використовується багаторазово, уникаючи дублювання;

— ізольована структура компонентів спрощує написання тестів;

— проєкти легко розширюються та адаптуються під зростаючі потреби.

Аналіз моделі взаємодії компонентів системи з точки зору інформаційної безпеки дозволяє виявити потенційні вразливості на різних рівнях архітектури [33].

Варто також згадати Django Rest Framework (DRF) — фреймворк для створення RESTful API за допомогою фреймворку Django, який забезпечує серіалізацію моделей, відображення даних за допомогою стандартних представлень на основі функцій або деталізацію за допомогою представлень на основі класів для складнішої функціональності [34]. В цій роботі DRF знадобиться для реалізації REST API, яке в свою чергу потрібне для взаємодії клієнта та сервера системи.

Фреймворк DRF значно спрощує розробку API завдяки вбудованій системі аутентифікації та авторизації, яка підтримує різні методи — від базової HTTP-аутентифікації до токенів і OAuth. Фреймворк також надає зручний вебінтерфейс для перегляду та тестування API безпосередньо в браузері.

Ключова особливість DRF — це гнучка система серіалізаторів, які легко перетворюють складні типи даних у формати JSON або XML з валідацією вхідних даних. До того ж, фреймворк має підтримку вбудованої пагінації, фільтрації та сортування. Завдяки системі ViewSets та Routers, DRF автоматично генерує URL-маршрути для стандартних CRUD-операцій, значно скорочуючи кількість написаного коду.

Отже, фреймворк Django поєднує в собі стабільність, продуманість архітектури та велику кількість функціональних можливостей, що робить його необхідним інструментом для розробки сучасних вебзастосунків різного масштабу. Його особливості забезпечують як швидку розробку, так і підтримку

високих стандартів безпеки та ефективності, що є вкрай необхідним для реалізації даної інформаційної системи.

2.3.2 Клієнтська частина

Для створення клієнтської частини інформаційної системи збору даних про товари в інтернет-магазинах було обрано бібліотеку React та мову програмування TypeScript.

Бібліотека React є бібліотекою мови програмування JavaScript, що використовується для створення користувацьких інтерфейсів вебзастосунків. Вона орієнтована на компонентний підхід, що дозволяє розбивати інтерфейс на незалежні, повторно використовувані частини, які керують власним станом і логікою. Це забезпечує зручність у розробці, тестуванні та підтримці великих проєктів, де зміни в одній частині інтерфейсу не повинні впливати на інші. Завдяки використанню віртуального DOM, бібліотека React оптимізує процес оновлення сторінок, значно зменшуючи необхідність повторного відмальовування елементів, що значно підвищуючи продуктивність системи.

В бібліотеці React існує два типи компонентів: класу (інша назва «стану») та функціональні компоненти (компоненти «без стану») [35]. Метод створення компонентів класу, що використовують стан і оголошуються за допомогою ключового слова «class», вважається застарілим [35]. Натомість функціональні компоненти є сучасним і рекомендованим способом розробки в React.

Варто також згадати JSX (JavaScript XML) — це спеціальний синтаксис, який використовується в React для спрощення створення користувацьких інтерфейсів [36]. JSX дає можливість писати код схожий на HTML безпосередньо в JavaScript, що дозволяє ефективніше створювати компоненти інтерфейсу користувача [36]. Хоча JSX виглядає як звичайний HTML, насправді це розширення синтаксису для JavaScript [36].

Поєднання React з TypeScript надає можливість використовувати статичну типізацію при розробці інтерфейсів. Це забезпечує передбачуваність, захист від

типових помилок та зручність інтеграції у великі проєкти. Типізація компонентів, властивостей та функцій дозволяє організувати код таким чином, щоб помилки виявлялися ще до виконання програми, що суттєво підвищує надійність та стабільність системи. TSX, тобто формат файлів, у яких поєднується TypeScript і JSX, дає змогу одночасно описувати розмітку та типи, створюючи структурований і зрозумілий код.

Бібліотека React дотримується декларативного підходу: розробник описує, яким має бути інтерфейс у певному стані, а React самостійно забезпечує оновлення DOM відповідно до змін даних. Це значно спрощує роботу з динамічними інтерфейсами, асинхронними операціями та взаємодією з API. Завдяки розвитку екосистеми, React пропонує широкий вибір інструментів для керування станом, маршрутизації, взаємодії з REST і GraphQL, а також значну кількість сторонніх бібліотек, що дозволяє адаптувати його до систем будь-якої складності.

Переваги використання бібліотеки React:

— компонентний підхід, що забезпечує повторне використання коду;

— оптимізація оновлення DOM завдяки віртуальному DOM;

— інтеграція з TypeScript для забезпечення типізації та раннього виявлення помилок;

— декларативний підхід до створення інтерфейсів, що спрощує роботу з динамічними елементами;

— багатий набір інструментів для керування станом, маршрутизації та роботи з API;

— велика спільнота та активна розробка, що забезпечує доступ до численних бібліотек та розширень.

Недоліки використання бібліотеки React:

— висока складність для початківців через необхідність розуміння JSX, компонентів та стану;

— потреба у додаткових бібліотеках для повноцінної організації маршрутизації або управління станом на великих проєктах;

— постійні оновлення та зміни в екосистемі, що потребують від розробника регулярного навчання;

— на початкових етапах продуктивність може знижуватись при дуже великих та глибоко вкладених компонентах без оптимізації.

Отже, бібліотека React у поєднанні з мовою програмування TypeScript забезпечує ефективну, масштабовану та безпечну розробку сучасних вебзастосунків, дозволяючи створювати високоякісні інтерактивні інтерфейси та підтримувати їх на професійному рівні у великих і складних проєктах, що повністю задовольняє потреби цього проєкту.

2.3.3 Вибір технологій для підсистеми збору даних

Ключовою складовою даної системи є підсистема автоматизованого збору даних про товари, відповідно важливим завданням є правильний підбір технологій для її реалізації. Далі буде розглянуто технології, використані для створення підсистеми збору даних.

2.3.3.1 Бібліотека aiohttp

Бібліотека aiohttp — це асинхронна клієнт-серверна бібліотека HTTP для використання асинхронності в Python. Вона надає простий API для створення HTTP-запитів та обробки як клієнтської, так і серверної функціональності [37]. Як і бібліотека requests, aiohttp розроблена для простоти використання та обробки низькорівневих операцій з протоколом HTTP [37].

Головна перевага бібліотеки aiohttp над бібліотекою requests полягає в тому, що вона побудований на основі бібліотеки asyncio, а це означає, що вона може обробляти багато запитів одночасно, не блокуючи виконання програми [37]. Це може призвести до значного покращення продуктивності під час створення багатьох невеликих запитів або при роботі з повільними або ненадійними мережевими з'єднаннями [37].

Бібліотека aiohttp підтримує як HTTP, так і «WebSocket» протоколи, що робить її універсальним інструментом для розробки вебдодатків. Вона включає вбудовану підтримку сесій, що дозволяє зберігати файли «cookie» та заголовки між запитами, а також автоматично обробляє перенаправлення та час очікування відповіді.

Для серверної частини бібліотека aiohttp надає можливість створювати високопродуктивні вебсервери з підтримкою проміжного програмного забезпечення, маршрутизації та шаблонів. Бібліотека також підтримує потокову передачу даних для завантаження та вивантаження великих файлів без повного завантаження їх у пам'ять, що особливо корисно при роботі з великими обсягами даних. Іншою важливою особливістю є те, що aiohttp використовує «пул з'єднань», що дозволяє повторно використовувати TCP-з'єднання для кількох HTTP-запитів, дозволяючи зменшити витрати на встановлення з'єднання, що є ефективним при виконанні великої кількості запитів до одного сервера.

2.3.3.2 Бібліотека Selenium

Бібліотека Selenium — це відкритий комплекс інструментів для автоматизованого тестування вебресурсів і взаємодії з браузерами [38]. За його допомогою можна створювати тестові сценарії різними мовами програмування з метою автоматизації перевірки роботи вебзастосунків [38]. Використання механізмів, які відтворюють дії користувача в браузері, дає змогу автоматизувати також рутинні операції, зокрема заповнення форм і переміщення між сторінками [38].

Важливою можливістю бібліотеки Selenium є робота з динамічними вебсторінками, у яких дані змінюються без повного перезавантаження сторінки. Завдяки підтримці керування виконанням сценаріїв у реальному часі та вбудованим механізмам очікування, бібліотека дає змогу правильно обробляти елементи, які з’являються, оновлюються або завантажуються асинхронно. Ця особливість дає можливість отримувати вміст сторінок, який генерується під час

взаємодії користувача із вебсторінкою або завантажується з сервера через фонові запити. Як результат, бібліотека Selenium широко застосовується не тільки для тестування, а й для автоматизованого збору даних, що значно розширює напрямки її використання.

Як висновок, для цієї роботи Selenium є надзвичайно важливим інструментом, який дозволить збирати дані зі сторінок, які динамічно завантажують свій контент.

2.3.3.3 Бібліотека BeautifulSoup4

Бібліотека BeautifulSoup4 — це бібліотека Python, яка була розроблена для розбору HTML та XML документів [39]. Вона спрощує процес вебскрепінгу, дозволяючи розробникам легко орієнтуватися, шукати та змінювати дерево розбору вебсторінки [39]. За допомогою бібліотеки BeautifulSoup4 можна витягувати певні елементи, атрибути та текст зі складних вебсторінок за допомогою інтуїтивно зрозумілих методів [39]. Ця бібліотека абстрагує складність структур HTML та XML, дозволяючи зосередитися на отриманні та обробці необхідних даних [39]. Бібліотека beautifulSoup4 підтримує кілька «парсерів» (таких як вбудований у Python «html.parser», «lxml» та «html5lib»), що дає розробникам гнучкість у виборі найкращого інструменту для їх завдання [39].

Додатковою перевагою бібліотеки BeautifulSoup4 є її здатність правильно обробляти «неідеальний» HTML-код, який часто зустрічається на реальних вебсторінках. Бібліотека вміє автоматично згладжувати помилки розмітки, відновлювати відсутні теги та формувати цілісне дерево документа, що значно підвищує надійність та ефективність вебскрепінгу в умовах непередбачуваних або некоректних даних.

Отже, бібліотека BeautidulSoup4 є невід’ємним інструментом для виконання поставлених в цій роботі задач. В той час коли бібліотеки aiohttp та Selenium виконують збір розмітки сторінки, BeautifulSoup4 відповідає за пошук та збір потрібних даних в розмітці сторінки.

2.3.4 Вибір технологій для модуля автоматизованого збору даних

Важливою функцією даної інформаційної системи є можливість автоматизованого запуску збору даних за розкладом. Для реалізації цієї функції було використано бібліотеку DjangoQ.

Бібліотека DjangoQ є бібліотекою черг завдань, планувальника та робочих процесів для фреймворку Django, що використовує багатопроцесорну обробку мови програмування Python. Ця бібліотека розроблена спеціально для інтеграції з фреймворком Django та надає розробникам можливість виконувати фонові завдання асинхронно, не блокуючи основний потік виконання вебдодатку. Бібліотека DjangoQ описується як «native Django task queue, scheduler and worker application using Python multiprocessing» [40], що підкреслює її природну інтеграцію з екосистемою фреймворку Django.

Архітектура бібліотеки DjangoQ побудована на принципі розподіленої обробки завдань через систему брокерів повідомлень. Брокер збирає пакети завдань від екземплярів бібліотеки Django та ставить їх у чергу для обробки кластером. Бібліотека підтримує різноманітні брокери, включаючи сховище Redis, яке є типовим варіантом, а також Disque, IronMQ, Amazon SQS, MongoDB тощо. Сховище Redis забезпечує найкращу продуктивність та не потребує додаткового налаштування кешування в фреймворку Django для моніторингу, тоді як MongoDB пропонує високу масштабованість та надійність для повідомлень з гарантією доставки принаймні один раз.

Основна функціональність бібліотеки включає підтримку асинхронних завдань через функцію «async_task», яка дозволяє швидко передавати завдання для виконання кластером робочих процесів. Розробники можуть створювати завдання безпосередньо з коду, передаючи функцію та її параметри, а також опціонально визначати хуки для обробки результатів виконання. Система автоматично серіалізує завдання, підписує їх, використовуючи модуль «django.core.signing», який містить секретний ключ проєкту, та опціонально стискає перед відправкою брокеру. Це забезпечує безпеку та можливість

виконання пакетів завдань лише тими кластерами та серверами фреймворку Django, які мають однаковий секретний ключ та ім'я кластера.

Особливо корисною є вбудована система планування, яка дозволяє налаштовувати виконання завдань за розкладом. Розкладами можна керувати через сторінки адміністратора або безпосередньо з коду за допомогою функції «schedule» або моделі «Schedule» [40]. Система підтримує різні типи розкладів, включаючи щохвилинні, щогодинні, щоденні, щотижневі, щомісячні та щорічні інтервали, а також гнучкі вирази для складніших сценаріїв планування. До того ж, розклади можуть бути визначені в програмному коді.

Архітектура бібліотеки DjangoQ заснована на кількох ключових компонентах, які синхронно працюють для забезпечення ефективної обробки завдань. Процес під назвою «pusher» безперервно перевіряє брокера на наявність нових завдань, перевіряє їх підпис та розпаковує завдання у внутрішню чергу завдань. Робочі процеси беруть завдання з черги та встановлюють таймер зворотного відліку. Після виконання завдання таймер скидається, результати зберігаються у пакет, який потім передається до черги результатів. Монітор результатів перевіряє чергу оброблених пакетів та зберігає як успішні, так і невдалі завдання в базу даних Django або сховище для кешу.

Налаштування бібліотеки DjangoQ здійснюється через словник під назвою «Q_CLUSTER» у файлі налаштувань проєкту Django. Розробники можуть визначати кількість робочих процесів, яка за замовчуванням дорівнює кількості ядер процесора, встановлювати максимальний час очікування для максимального часу виконання завдання, налаштувати очищення завдань перед перезапуском робочого процесу для звільнення пам'яті. Також ця бібліотека дає можливість контролювати, скільки завдань зберігається в пам'яті одним кластером, що допомагає розподіляти навантаження та витрати пам'яті. Додатково можна стискати великі пакети завдань та призначати робочі процеси до конкретних ядер процесора.

Управління кластером здійснюється через термінальні команди фреймворку Django, що значно спрощує процес розгортання та моніторингу фонових завдань. Переваги використання бібліотеки DjangoQ включають:

— просту інтеграцію з фреймворком Django;

— можливість використання бази даних фреймворку Django як брокера через ORM, що усуває необхідність додаткової інфраструктури для невеликих проєктів;

— вбудовану інтеграцію з адміністративною панеллю фреймворку Django для зручного управління завданнями та розкладами;

— підтримку підписаних та стиснених пакетів для безпеки та ефективності передачі даних;

— гнучку систему хуків для обробки результатів виконання завданя.

Недоліки DjangoQ полягають у наступному:

— бібліотека більше підходить для малих та середніх додатків з простішими вимогами [41];

— типовий брокер Redis не підтримує квитанції про отримання повідомлень, що означає можливість втрати завдань у випадку збою кластера або перевищення часу очікування робочими процесами;

— порівняно менша екосистема розширень порівняно з більш поширеними рішеннями, такого як Celery;

— обмежена документація.

Бібліотека особливо зручна для випадків, коли потрібно швидко реалізувати фонову обробку завдань у Django-проєкті без налаштування складної інфраструктури. Вона ідеально підходить для типових завдань вебдодатків, таких як відправка електронної пошти, обробка завантажених файлів, генерація звітів та періодичне оновлення даних. Бібліотека DjangoQ була створена з метою спрощення процесу виконання довготривалих завдань у вебдодатках [42], надаючи розробникам інструмент, який є легшим у налаштуванні та використанні порівняно з більш складними альтернативами.

Отже, використані бібліотека DjangoQ та сховище Redis цілком задовольняють потреби даного проєкту.

2.4 Вибір баз даних

Для реалізації поставлених задач, в роботі було використано 3 підходи до збереження даних. Для зберігання постійних даних, наприклад, про користувачів та їх налаштування, було використано реляційну базу даних, для зберігання даних протягом короткого часу — нереляційну, а для зберігання результату збору даних було використано окремі формати файлів.

2.4.1 СУБД SQLite

Система управління базами даних SQLite — це вбудована бібліотека, яка реалізує самодостатній, серверонезалежний та безконфігураційний транзакційний рушій баз даних SQL. Її вихідний код є відкритим, тому SQLite є повністю безкоштовною для будь-якого використання — комерційного чи приватного. Бібліотека SQLite є однією з найпоширеніших систем керування базами даних у світі, що застосовується в незліченній кількості застосунків, зокрема у багатьох відомих проєктах [43].

Бібліотека SQLite функціонує як вбудована СУБД. На відміну від більшості SQL-систем, вона не потребує окремого серверного процесу. Читання та запис даних здійснюються безпосередньо у звичайні дискові файли, а повноцінна SQL-база даних, яка містить таблиці, індекси, тригери та подання, зберігається в одному файлі. Формат файлу є міжплатформним, що дозволяє безперешкодно переносити бази даних між тридцятидвобітними та шістдесятичетирибітними системами. Такі властивості роблять SQLite популярним вибором як формат зберігання файлів застосунків [43].

Надійність бібліотеки SQLite забезпечується завдяки дуже ретельному тестуванню перед кожним релізом. Автоматизований тестовий пакет виконує

мільйони тестів, що охоплюють сотні мільйонів SQL-операторів і забезпечують стовідсоткове покриття розгалужень. Бібліотека SQLite коректно реагує на помилки виділення пам’яті та збої файлових операцій, а транзакції залишаються ACID-сумісними навіть у разі аварійного завершення роботи системи чи зникнення живлення [43]. При проєктуванні систем зберігання даних важливо враховувати комплексний підхід до забезпечення безпеки, який охоплює технічні та організаційні аспекти захисту інформації [44]. Математичне моделювання системи управління базами даних дозволяє формалізувати вимоги до захисту та цілісності інформації [45]. Моделювання інформаційної безпеки на рівні бази даних є основою для проєктування ефективних механізмів контролю доступу та забезпечує можливість прогнозування поведінки системи під впливом різних загроз [46]. Дослідження механізмів захисту баз даних дозволяє оцінити ефективність запроваджених засобів забезпечення конфіденційності та цілісності інформації [47]. Оцінка ризиків порушення цілісності даних є невід'ємною частиною проєктування надійних систем зберігання інформації [48].

Отже, можливостей бібліотеки SQLite цілком достатньо для реалізації поставлених завдань у межах цієї інформаційної системи. Її самодостатність, відсутність потреби в окремому сервері, висока надійність, підтримка транзакцій ACID та кросплатформність забезпечують стабільну роботу та коректне зберігання даних навіть за обмежених ресурсів. Завдяки згаданим раніше перевагам бібліотека SQLite є оптимальним вибором для системи, що потребує компактності, простоти інтеграції та впевненого функціонування без складної інфраструктури.

2.4.2 Сховище Redis

Для зберігання головних сторінок та сторінок товарів, які користувачі завантажують на сторінці створення шаблону, потрібно використовувати інструмент, що давав би можливість зберігати дані протягом короткого часу. Для цього завдання обрано сховище Redis.

Сховище Redis (REmote DIctionary Server) — це NoSQL-сховище вигляду ключ-значення з відкритим кодом, що зберігається в оперативній пам'яті та використовується переважно як кеш застосунків або база даних швидкого реагування [49]. Воно зберігає дані в пам'яті, а не на диску чи твердотільному накопичувачі, що допомагає забезпечити дуже високу швидкість, надійність та продуктивність [49].

На відміну від простих сховищ «ключ-значення», Redis функціонує як повноцінний сервер структур даних, забезпечуючи підтримку широкого спектра типів даних, таких як: унікальні несортовані рядкові елементи, бінарно-безпечні дані, бітові масиви, хеш-таблиці, списки тощо [49].

Коли додаток залежить від зовнішніх джерел даних, затримки та пропускна здатність цих джерел можуть створювати вузькі місця продуктивності, особливо зі зростанням трафіку або масштабуванням додатка [49]. Одним із способів покращення продуктивності в таких випадках є зберігання та обробка даних безпосередньо в оперативній пам'яті, фізично ближче до додатка [49]. Сховище Redis створене саме для цього завдання, оскільки, воно зберігає всі дані в оперативній пам'яті, забезпечуючи максимально швидку продуктивність під час читання або запису даних, а також пропонує вбудовані можливості реплікації, які дозволяють розміщувати дані фізично ближче до користувача для мінімальної затримки [49].

Сховище Redis підтримує механізм автоматичного керування часом життя збережених записів, що дає змогу системі самостійно вилучати дані після завершення встановленого періоду їх актуальності. Крім того, сховище Redis забезпечує високошвидкісні операції читання та модифікації даних, завдяки зберіганню інформації безпосередньо в оперативній пам’яті. За потреби повного очищення сховища достатньо виконати його перезапуск, що робить процес оновлення або скидання стану максимально простим.

У сукупності такі можливості перетворюють Redis на ефективний інструмент для систем, у яких пріоритетом є мінімальна затримка обробки запитів. Це особливо актуально для реалізації тимчасових буферів даних,

керування станом активних користувацьких сесій та оперативного опрацювання великої кількості короткоживучих запитів, що є важливим для розроблюваної в цій роботі інформаційної системи.

2.4.3 Формати JSON та CSV

Для зберігання даних про налаштування збору даних та зібрані товари було вирішено використовувати два типи файлів, а саме JSON та CSV відповідно.

Формат JSON (JavaScript Object Notation) — це легкий формат обміну даними [50]. Він є простим для читання та написання людиною, а також розбору та генерування машинами [50]. Формат JSON є текстовим форматом, який є повністю незалежним від мови програмування, але використовує умовності, знайомі програмістам мов сімейства C, включно з C, C++, C#, Java, JavaScript, Perl, Python та багатьма іншими [50].

Формат JSON побудований на двох структурах [50]:

— колекція пар «ім’я/значення», яка у різних мовах реалізується як об’єкт, запис, структура, словник, хеш-таблиця, список із ключами або асоціативний масив;

— упорядкований список значень, який в більшості мов програмування реалізується як масив, вектор, список або послідовність.

Це універсальні структури даних, які підтримують майже всі сучасні мови програмування.

Кожен JSON-файл містить у собі об’єкт. Об’єкт — це неупорядкований набір пар «ім’я/значення» [50]. Об’єкт починається та закінчується фігурною дужкою [50]. Кожне ім’я супроводжується символом двокрапки, а пари «ім’я/значення» розділяються символом коми [50].

Формат CSV (Comma-Separated Values) — це простий текстовий формат для зберігання та обміну структурованими табличними даними. У CSV-файлі кожен рядок відповідає одному запису, а поля запису відокремлюються комами або іншими роздільниками. Цей формат не передбачає вкладених структур або

форматування тексту, що забезпечує його високу сумісність із різними програмними продуктами, такими як табличні процесори, бази даних, аналітичні інструменти та мови програмування.

Файли формату CSV часто застосовуються для імпорту та експорту даних між системами, передачі інформації між різними додатками та у процесах підготовки даних для аналізу. Завдяки своїй простоті та відкритості формат є універсальним і легко опрацьовується навіть без спеціалізованого програмного забезпечення. До переваг формату CSV можна віднести простоту та зрозумілість, широку сумісність, невеликий обсяг пам’яті, необхідної для зберігання та легкість автоматизації.

До недоліків формату CSV можна віднести:

— відсутність структури для складних даних, а отже цей формат не підходить для вкладених або ієрархічних даних;

— відсутність типізації, через те, що всі дані зберігаються у вигляді тексту, що може ускладнювати обробку числових значень або дати та часу;

— ризик некоректного розділення полів, у випадках, якщо значення містять символ роздільника, потрібне додаткове або обрамлення у лапки;

— відсутність метаданих, тому що формат не містить інформації про структуру таблиці, типи даних або форматування.

Висновки до розділу 2

У цьому розділі було визначено основні засоби розробки інформаційної системи, розглянуто та проаналізовано можливі архітектурні підходи, обґрунтовано вибір мови програмування, а також підібрано відповідні фреймворки й бібліотеки для серверної, клієнтської частин і підсистеми збору даних. Окрему увагу приділено вибору форматів та систем керування даними, що забезпечують ефективне зберігання, обробку та подальше використання інформації.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

У цьому розділі наведено технічний опис створеної інформаційної системи, що включає її архітектуру, інформаційну модель та ключові функціональні компоненти.

3.1 Архітектура інформаційної системи

Програмне забезпечення, розроблене в межах цієї роботи, ґрунтується на клієнт-серверній архітектурі: клієнтська частина забезпечує представлення й взаємодію з даними, тоді як серверна частина відповідає за їх приймання, оброблення та подальше зберігання. Діаграму прецедентів можна побачити в у додатку Б, архітектуру створеної системи у вигляді діаграми компонентів зображено в додатку В, а в додатку Г зображено структурну діаграму.

Серверна частина системи відповідає за виконання двох ключових функцій. По-перше, вона здійснює надсилання запитів до цільових вебресурсів для збору необхідної інформації. Цей процес включає аналіз структури сторінок інтернет-магазинів та витягування релевантних даних згідно з визначеними користувачем правилами збору даних. По-друге, сервер забезпечує взаємодію з таблицями бази даних та файловою системою, що дозволяє зберігати зібрану інформацію у структурованому вигляді та забезпечувати швидкий доступ до неї.

Для оптимізації роботи системи та підвищення швидкості доступу до даних використовується механізм кешування через Redis. Збережена кеш-пам'ять у Redis дозволяє зменшити навантаження на базу даних та прискорити відповіді на повторювані запити користувачів. Зібрані дані зберігаються у файловій системі, що забезпечує надійне та структуроване зберігання інформації, а також спрощує процес доступу до цих даних.

3.2 Опис інформаційної моделі

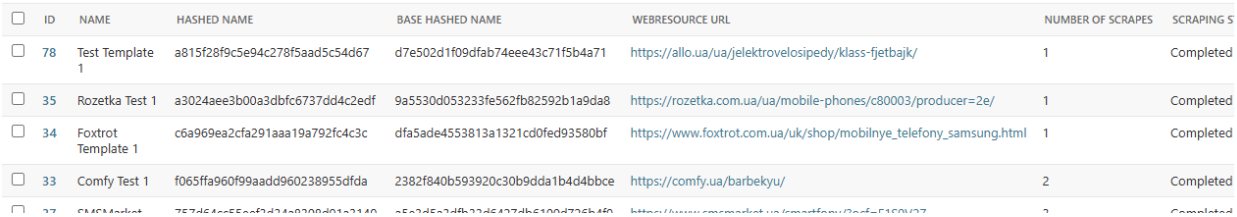
Створений програмний продукт містить дві окремих бази даних, кожна з яких має окремо поставлені в цій роботі задачі.

Спочатку розглянемо основну/реляційну базу даних, а саме SQLite, яку було використано для зберігання постійних даних. Розглянемо таблиці, які містить реляційна БД в цій програмі: шаблони (Template), користувачі (User) та заплановані завдання (Scheduled task).

Таблиця «Template» призначена для зберігання основної інформації про шаблони, що дає можливість системі знати шлях до зібраних файлів та налаштувань цього шаблону, контролювати виконання та статус цього шаблону. Структуру таблиці зображено в таблиці 3.1, а приклад заповнення даними на рисунку 3.1 на прикладі вбудованої у Python-фреймворк Django панелі адміністратора.

Таблиця 3.1 — Структура таблиці «Шаблон»

Назва поля таблиці	Тип даних	Призначення
Name	Text	Назва шаблону
base_hashed_name	Text	Хеш посилання
hashed_name	Text	Унікальний хеш посилання
webresource_url	URL	Посилання
number_of_scrapes	Integer	Кількість завершених зборів даних
last_scrape	DateTime	Дата завершення останнього збору даних
is_automated	Boolean	Чи потрібно запускати цей шаблон автоматизовано
delay_between_scrapes	Integer	Проміжок часу між виконанням збору даних, в секундах
scraping_status	Text	Статус поточного шаблону
scraping_started_at	DateTime	Дата та час початку збору даних
scraping_progress	Integer	Прогрес виконання збору даних (від 0 до 100%)
scraping_error	Text	Помилка, яка виникла під час збору даних
User	Foreign Key (Table User)	Зовнішній ключ таблиці «User»



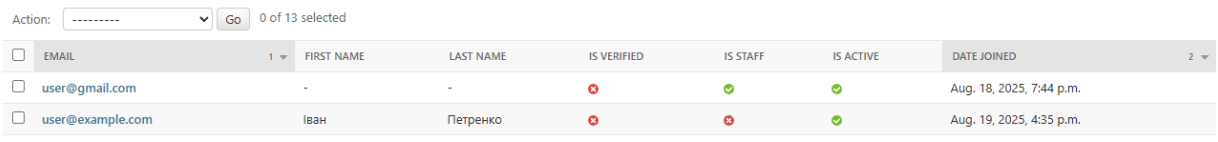
	ID	NAME	HASHED NAME	BASE HASHED NAME	WEBRESOURCE URL	NUMBER OF SCRAPES	SCRAPING S
	78	Test Template 1	a815f28f9c5e94c278f5aad5c54d67	d7e502d1f09dfab74eee43c71f5b4a71	https://allo.ua/ua/jelektrovelosipedy/klass-fjetbajk/	1	Completed
	35	Rozetka Test 1	a3024aee3b00a3dbfc6737dd4c2edf	9a5530d053233fe562fb82592b1a9da8	https://rozetka.com.ua/ua/mobile-phones/c80003/producer=2e/	1	Completed
	34	Foxtrot Template 1	c6a969ea2cfa291aaa19a792fc4c3c	dfa5ade4553813a1321cd0fed93580bf	https://www.foxtrot.com.ua/uk/shop/mobilnye_telefony_samsung.html	1	Completed
	33	Comfy Test 1	f065ffa960f99aadd960238955dfda	2382f840b593920c30b9dda1b4d4bbce	https://comfy.ua/barbekyu/	2	Completed

Рисунок 3.1 — Приклад заповнення таблиці «Шаблон»

Метою таблиці «User» є зберігання інформації про користувачів системи та управління їхніми обліковими записами. Ця таблиця забезпечує функціонал автентифікації та авторизації користувачів, дозволяючи їм безпечно входити до системи та отримувати доступ до персоналізованих даних. Кожен користувач ідентифікується за унікальною електронною адресою, яка виступає основним ідентифікатором для входу в систему. Поля активності та адміністративних прав забезпечують гнучке управління доступом користувачів до різних рівнів функціональності програми. Структуру таблиці зображено в таблиці 3.2, а приклад заповнення даними на рисунку 3.2.

Таблиця 3.2 — Структура таблиці «Користувач»

Назва поля таблиці	Тип даних	Призначення
Email	Text	Електронна пошта
first_name	Text	Ім’я
last_name	Text	Прізвище
is_active	Boolean	Чи користувач онлайн
is_staff	Boolean	Чи користувач є адмінвстратором
date_joined	DateTime	Дата створення облікового запису
date_updated	DateTime	Дата останнього оновлення облікового запису



Action: --------- Go 0 of 13 selected

	EMAIL	FIRST NAME	LAST NAME	IS VERIFIED	IS STAFF	IS ACTIVE	DATE JOINED
	user@gmail.com	-	-	✗	✓	✓	Aug. 18, 2025, 7:44 p.m.
	user@example.com	Іван	Петренко	✗	✗	✓	Aug. 19, 2025, 4:35 p.m.

Рисунок 3.2 — Приклад заповнення таблиці «Користувач»

Останньою таблицею є «ScheduledTask», метою якою є управління запланованими завданнями в системі. Ця таблиця забезпечує автоматизоване

виконання періодичних операцій збору даних з інтернет-магазинів згідно з визначеним розкладом. Кожен запис у таблиці представляє окреме заплановане завдання, яке містить інформацію про назву задачі, функцію для виконання, тип розкладу та параметри повторення. Поле функції вказує на конкретний метод у системі, який буде викликано під час виконання завдання, що дозволяє гнучко налаштовувати різні типи операцій збору даних. Тип розкладу визначає інтервал виконання завдання, наприклад, у хвилинах, годинах або днях, а параметр повторень встановлює кількість запусків або вказує на безперервне виконання. Таблиця також фіксує час наступного та останнього запуску завдання, що дозволяє відстежувати історію виконання та забезпечує можливість моніторингу успішності операцій збору даних. Структуру таблиці зображено в таблиці 3.3, а приклад заповнення даними на рисунку 3.3.

Таблиця 3.3 — Структура таблиці «Заплановане завдання»

Назва поля таблиці	Тип даних	Призначення
Id	Integer	Унікальний ідентифікатор завдання
Name	Text	Назва запланованого завдання
Func	Text	Шлях до функції для виконання
schedule_type	Text	Тип розкладу (хвилини, години, дні)
Repeats	Integer	Кількість повторень виконання
Cluster	Text	Ім'я кластера для виконання
next_run	DateTime	Дата та час наступного запуску
last_run	DateTime	Дата та час останнього запуску
Success	Boolean	Статус успішності останнього виконання

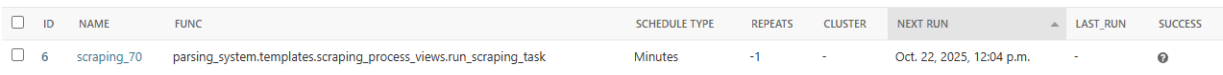


Рисунок 3.3 — Приклад заповнення таблиці «Заплановане завдання»

Варто також зазначити, що завдяки вбудованим інструментам фреймворку Django, таблиці бази даних не створювались безпосередньо в SQLite, а прописувалося в спеціальному файлі у вигляді програмного коду. Приклад створення таблиці в Django зображено на рисунку 3.4. Для змінення бази даних необхідно прописати дві команди: «python manage.py makemigrations» та «python manage.py migrate» — після чого відповідну базу даних в SQLite буде змінено відповідно до моделей, прописаних в коді програми.

```
class User(AbstractBaseUser, PermissionsMixin):
    email = models.EmailField(unique=True)
    first_name = models.CharField(max_length=150, blank=True)
    last_name = models.CharField(max_length=150, blank=True)
    is_verified = models.BooleanField(default=False)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)
    date_joined = models.DateTimeField(default=timezone.now)
```

Рисунок 3.4 — Приклад створення моделі в Django

Для тимчасового збереження зібраних вебсторінок було використано розподілене сховище пар ключ-значення Redis, яке дає можливість зберігати дані за унікальним ключем, нагадуючи структуру асоціативного масиву. Корисною рисою Redis є можливість зберігання пари ключ-значення протягом певного часу, вказавши значення відповідного аргументу під час операції створення пари ключ-значення.

Сховище Redis зберігає дані в оперативній пам’яті, а отже, чудово підходить для зберігання не важливих даних, наприклад кешу. Це підходить для завдання цієї роботи, оскільки зібрані сторінки швидко втрачають свою актуальність та повинні зберігатись протягом короткого часу.

Для запуску Redis необхідно запустити Redis-сервер, який є відповідальним за зберігання даних, та Redis-клієнт, який дає можливість взаємодіяти із Redis-сервером (виконувати операції читання та запису даних). При вимкненні сервера всі дані, які він зберігав, будуть втрачені.

Після того, як користувач зробив запит для відмальовування цільової сторінки, ввівши її посилання, підпрограма збору даних здійснює завантаження та обробку розмітки цієї сторінки. Після цього відбувається збереження розмітки в сховищі Redis. Наприклад, користувач ввів посилання «https://allo.ua/ua/jelektrovelosipedy/klass-fjetbajk/». Після цього в сховищі з'явиться відповідний запис (посилання хешується), зображений на рисунку 3.5, який буде зберігатись протягом тридцяти хвилин.

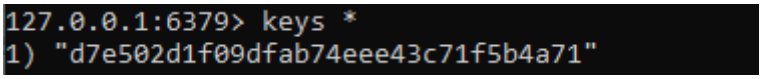


Рисунок 3.5 — Приклад запису пари ключ-значення в сховище Redis

Отже, сховище Redis підходить для цієї роботи, оскільки дані про знайдені сторінки швидко втрачають актуальність та змінюються. При цьому, це сховище дає можливість пришвидшити роботу програми та зменшити навантаження на сервери цільових сайтів, оскільки сторінки будуть зберігатись протягом певного часу. Також це дає можливість користувачам миттєво отримати результат при повторному введенні однакового посилання, без очікування на повторний збір та обробку зібраної сторінки.

3.3 Опис механізму роботи підходу до збору даних

Для реалізації підсистеми збору даних в рамках цієї роботи необхідно вибрати підхід, який би дозволив збирати дані швидко, використовуючи при цьому не надто велику кількість ресурсів. Саме тому було вирішено використовувати асинхронний підхід до організації програми.

Асинхронне програмування являє собою підхід, за якого програма може продовжувати виконання подальших дій, не очікуючи завершення попередніх операцій. Такий механізм дає змогу запобігти блокуванню під час виконання ресурсомістких або тривалих процесів, зокрема мережевих звернень, операцій введення/виведення та опрацювання значних обсягів даних.

Основним в управлінні асинхронною програмою є цикл подій (event loop). Це процес, який очікує та відправляє події або повідомлення, що називаються обіцянкою (promise) у програмі [51]. Як частина циклу подій, ви можете створити зворотний виклик, який дозволяє циклу подій передавати інформацію з програми іншому фрагменту коду, зазвичай це основна функція програми [51].

За час поки додаток очікує завершення операції, він може виконувати інші задачі [51]. Це дозволяє виконувати потенційно ресурсоємні завдання, не змушуючи користувача чекати на їх завершення [51]. Await — це функція або операція багатьох мов програмування, яка дозволяє реалізовувати асинхронне програмування [51].

Асинхронне програмування є ефективним при виконанні наступних операцій:

— мережеві запити;

— операції читання або запису файлів;

— тривалі анімації;

— операції з великою кількістю обчислень.

Варто також розглянути й інші підходи до організації програми, а саме, синхронне та паралельне програмування.

Синхронне програмування передбачає послідовне виконання інструкцій, коли кожна операція повинна повністю завершитися, перш ніж розпочнеться наступна. Такий підхід є простим у реалізації та передбачуваним у плані контролю потоку, однак він виявляється малоефективним у випадках, коли окремі операції займають тривалий час. Особливо це відчутно у програмах, що активно працюють із мережею або виконують численні операції введення/виведення, оскільки під час очікування результату програма фактично простоює.

Паралельне програмування, на відміну від синхронного, дає змогу виконувати декілька обчислювальних задач одночасно за допомогою багатоядерних процесорів або розподілених систем. Воно забезпечує значне прискорення для обчислювально інтенсивних задач, проте потребує більшого обсягу апаратних ресурсів і складніших механізмів синхронізації. Використання

паралельності найефективніше для задач, які можна розділити на незалежні частини, що виконуються одночасно.

У підсумку, саме асинхронний підхід є найбільш доречним для реалізації поставленого завдання, адже він дає змогу скоротити загальний час роботи програми за умов великої кількості HTTP-запитів, не потребуючи при цьому збільшення обчислювальних ресурсів.

3.4 Опис підпрограми збору даних

Основою підпрограми збору даних є JSON-файли із налаштуваннями збору даних, які називаються в системі «configuration.json» (рис. 3.6), кожен з яких відноситься до окремого шаблону та має унікальні дані, задані користувачем під час створення шаблону, за допомогою яких можна зібрати дані за вказаним посиланням. В додатку Е зображено блок-схему роботи алгоритму збору даних.

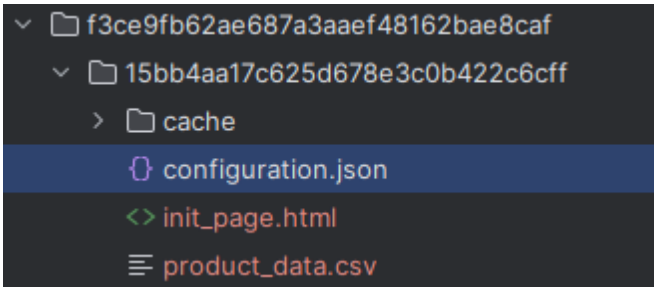


Рисунок 3.6 — Вигляд шаблону із зібраними даними в файловій системі

Кожен файл повинен відповідати чітко визначеній структурі, інакше процес збору даних із вказаного інтернет-магазину буде неможливим. Перелік необхідних для файлу полів:

— «base_url» — посилання, за яким необхідно збирати дані;

— «pagination» — об’єкт на сторінці, який містить пагінацію;

— «elements_to_parse» — список атрибутів, з яких потрібно збирати дані;

— «parent_element» — батьківський об’єкт продукту на головній сторінці, який містить в собі всі об’єкти (elements_to_parse), в яких source = «Main Page»;

— «hashed_name» — унікальний хеш шаблону, необхідний для його пошуку в файловій системі;

— «is_automated» — булеве значення, яке вказує на те, чи потрібно виконувати автоматизований збір даних за цим шаблоном;

— «scraping_frequency» — частота автоматизованого збору даних в хвилинах;

— «product_page_url» — посилання на сторінку продукту (є необов’язковим);

— «product_page_url_object» — об’єкт, який містить посилання на сторінку продукту;

— «webresource_type» — може мати значення «selenium» або «aiohttp», що вказує на те, використовуючи яку бібліотека необхідно збирати дані.

Певні елементи є необов’язковими, наприклад, «scraping_frequency», «product_page_url» та «product_page_url_object». Також такі елементи, як «pagination», «parent_element», «product_page_url_object» та «webresource_type», генеруються системою автоматично.

Певні характеристики мають вкладені параметри. Наприклад, елементи «parent_element» та «product_page_url_object» містять кілька вкладених полів:

— «tag» — тег певного об’єкта в розмітці сторінки;

— «attrs» — атрибути певного елемента в розмітці сторінки.

Елемент «pagination» генерується системою автоматично та необхідний для пошуку всіх сторінок з продуктами. Цей елемент має наступну структуру:

— «number_of_pages» — кількість знайдених за посиланням сторінок;

— «pagination_link_template» — шаблон сторінки пагінації, наприклад: https://allo.ua/ua/jelektrovelosipedy/klass-fjetbajk/p-PAGE_NUMBER/ — під час процесу збору даних система згенерує список потрібних сторінок, заміняючи «PAGE_NUMBER» на номер сторінки;

Елемент «elements_to_parse» є масивом та містить в собі об’єкти, дані з яких потрібно зібрати. Цей елемент має наступну структуру (приклад на рис. 3.7):

— «field_name» — назва атрибута, яку задав користувач;

— «type» — тип даних, які потрібно зібрати (можливі варіанти: рядок, число та посилання);

— «tag» — тег певного об’єкта в розмітці сторінки;

— «attrs» — атрибути певного елемента в розмітці сторінки;

— «source» — вказує, на якій сторінці потрібно збирати значення атрибута, на сторінці з пагінацією, або на сторінці продукту.

Рисунок 3.7 — Приклад елементу «elements_to_parse» в середині файлу «configuration.json»

Елемент «webresource_type» також визначається системою автоматично. У випадку, якщо сайт є статичним та дає можливість зібрати дані для атрибутів за допомогою бібліотеки «aiohttp», тоді значення елемента «webresource_type» матиме значення назви цієї бібліотеки. Для випадків, коли контент вебсторінки підвантажується автоматично, для отримання розмітки сторінки буде використано бібліотеку Selenium, а значення елементу «webresource_type» буде «selenium».

Після того, як користувач зберігає шаблон, для нього створюється спеціальна папка зі шляхом «data/base_hashed_name/hashed_name», де «data» — це папка, в якій зберігаються результати першого, або останнього, збору даних.

«base_hashed_name» та «hashed_name» це два значення хешу, згенерованого на основі посилання на вебресурс.

Після того, як для певного шаблону буде повторно проведено процес збору даних, дані з папки «data» будуть перенесені в папку «archive» (рис. 3.8), а в папці «data» з’являться результати останнього збору даних.

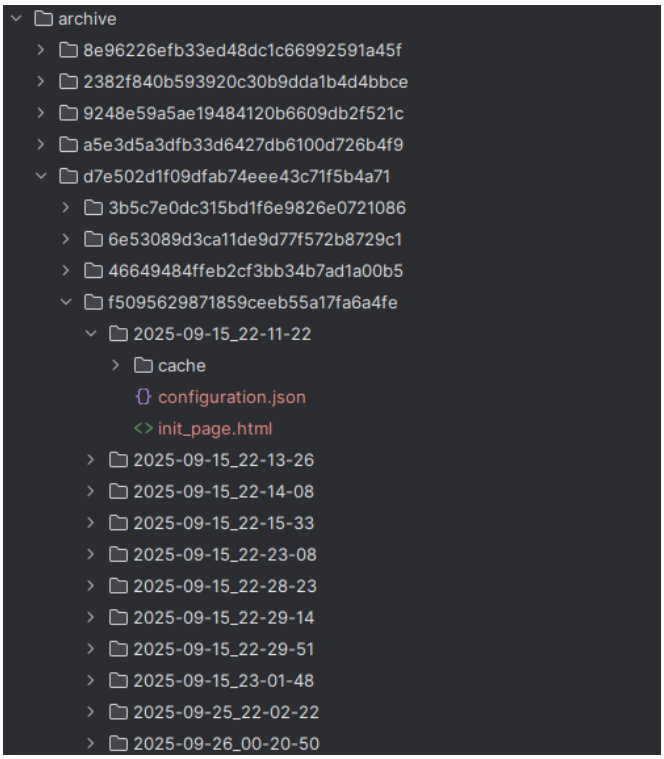


Рисунок 3.8 — Приклад вмісту папки «archive»

Варто також приділити увагу процесу збору даних. Перш за все, підпрограма збору даних відкриває посилання на вебресурс, яке вказав користувач при створенні шаблону, та проводить аналіз сторінки за цим посиланням, а саме: шукає пагінацію, та батьківський об’єкт продуктів. Початкова сторінка зберігається в папці відповідного шаблону під назвою «init_page.html».

Після завершення початкового аналізу та генерації і збору посилань на всі необхідні сторінки, починається процес збору вебсторінок, та їх збереження в папці «cache» (рис. 3.9).

Також підсистема збору даних генерує файл «scrape_info.json», в якому зберігається інформація про початок та закінчення процесу збору даних та дані про початок та закінчення збору розмітки для кожної сторінки (рис. 3.10).

Як було зазначено раніше, створена система також має можливість автоматизованого запуску процесу збору даних за розкладом. Діаграму послідовності роботи підсистеми автоматизованого запуску процесу збору даних зображено в додатку И.

До того ж, у додатку К представлено послідовність станів та переходів, які проходить процес збору даних під час свого виконання.

Рисунок 3.9 — Вміст папки «cache»

Рисунок 3.10 — Приклад вмісту файлу «scrape_info.json»

3.5 Опис модуля обчислення часу збору даних

В рамках даної роботи було також створено модуль обчислення часу виконання процесу збору даних. Для цього було розроблено формулу часу збору даних:

$$T = Ts + Tp,$$

де Ts — загальний час виконання вебскрепінгу;

Tp — загальний час виконання парсингу даних.

Загальний час вебскрепінгу, тобто час збору розмітки всіх потрібних сторінок, визначається за формулою:

$$Ts = ((Nm \div P) + (Np \div P)) \times t, \quad (3.1)$$

де Nm — кількість вебсторінок пагінації, з яких потрібно зібрати дані;

Np — кількість вебсторінок продуктів, з яких потрібно зібрати дані;

P — кількість потоків;

t — середній час, який витрачається на отримання розмітки однієї сторінки.

Загальний час парсингу даних, тобто вилучення потрібних даних із зібраних сторінок, визначається за формулою:

$$Tp = (Nm \times tm) + (Np \times tp),$$

де Nm — кількість сторінок пагінації;

Np — кількість сторінок продуктів;

tm — час, який витрачається на парсинг даних з однієї сторінки пагінації;

tp — час, який витрачається на парсинг даних з однієї сторінки продукту.

Результати роботи створеної функції наведено в таблиці 3.4, а на рисунку 3.11 зображено графік порівняння очікуваного та фактичного часу збору даних на основі результатів роботи створеної функції.

Таблиця 3.4 — Результати роботи створеної функції

Загальна кількість сторінок	Кількість сторінок упагінації	Кількість сторінок продуктів	Очікуваний час виконання (секунди)	Середній фактичний час виконання (секунди)
30	30	0	9.58	38
3	3	0	4.64	8
152	3	149	60.8	182
3	3	0	5.28	5
128	3	124	49.81	178
3	3	0	5.86	6
89	3	86	36.11	93
99	99	0	46.62	98

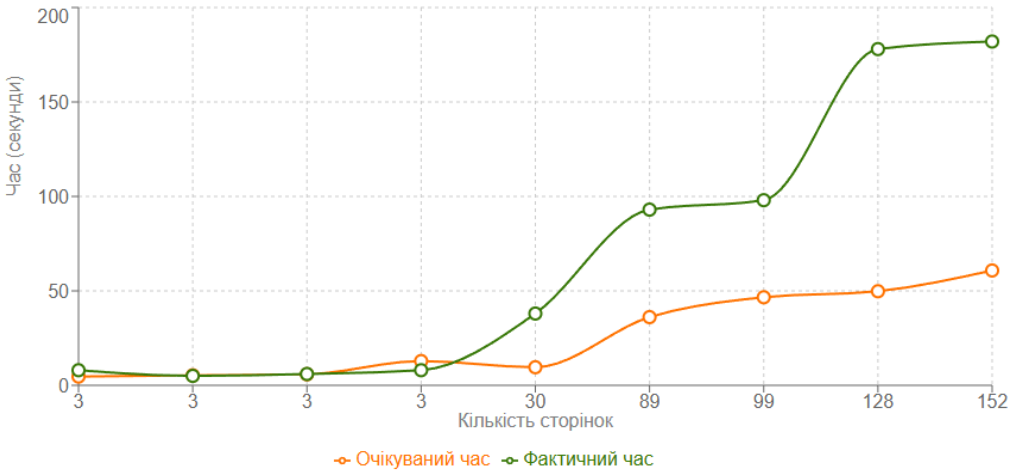


Рисунок 3.11 — Порівняння очікуваного та фактичного часу збору даних залежно від кількості сторінок

Аналіз результатів показав значне відхилення фактичного часу від теоретичного, особливо при збільшенні кількості сторінок. Для 3 сторінок середнє відхилення становить ~1.3x, для 30 сторінок — 3.97x, для 89-152 сторінок — 2.5-3.0x.

Виявлено нелінійну залежність між кількістю сторінок та реальним часом виконання. Для точної оцінки часу виконання розроблено адаптивний коефіцієнт затримок, який динамічно змінюється залежно від кількості вебсторінок.

Для обчислення значення коефіцієнта було застосовано метод кусково-лінійної інтерполяції з чотирма інтервалами, каліброваними на емпіричних даних.

В результаті на основі формули (3.1), додавши адаптивний коефіцієнт затримок, було створено оновлену формулу:

$$Ts = ((Nm \div P) + (Np \div P)) \times t \times k,$$

де k — коефіцієнт «випередження», для середнього часу збору розмітки однієї сторінки.

Реалізацію обчислення коефіцієнта k можна побачити на рисунку 3.12, а результати роботи оновленої функції в таблиці 3.5. Також на рисунку 3.13 зображено графік порівняння очікуваного та фактичного часу збору даних на основі результатів роботи оновленої функції.

```python
def calculate_adaptive_overhead(total_pages: int) -> float:  1 usage
    if total_pages <= 5:
        return 1.0
    elif total_pages <= 30:
        return 1.0 + (3.5 - 1.0) * ((total_pages - 5) / (30 - 5))
    elif total_pages <= 100:
        return 1.0 + (0.8 - 1.0) * ((total_pages - 30) / (100 - 30))
    else:
        return min(1.09 + 0.00015 * (total_pages - 100), 1.2)
```

Рисунок 3.12 — Функція обчислення коефіцієнта k на мові програмування Python

Таблиця 3.5 — Результати роботи оновленої функції

Загальна кількість сторінок	Кількість сторінок пагінації	Кількість сторінок продуктів	Очікуваний час виконання (секунди)	Середній фактичний час виконання (секунди)

30	30	0	41.64	38
3	3	0	12.85	8
152	3	149	192.93	182
3	3	0	8.63	5
128	3	125	123.87	178
3	3	0	10.74	6
89	3	86	62.91	93
99	99	0	84.24	98

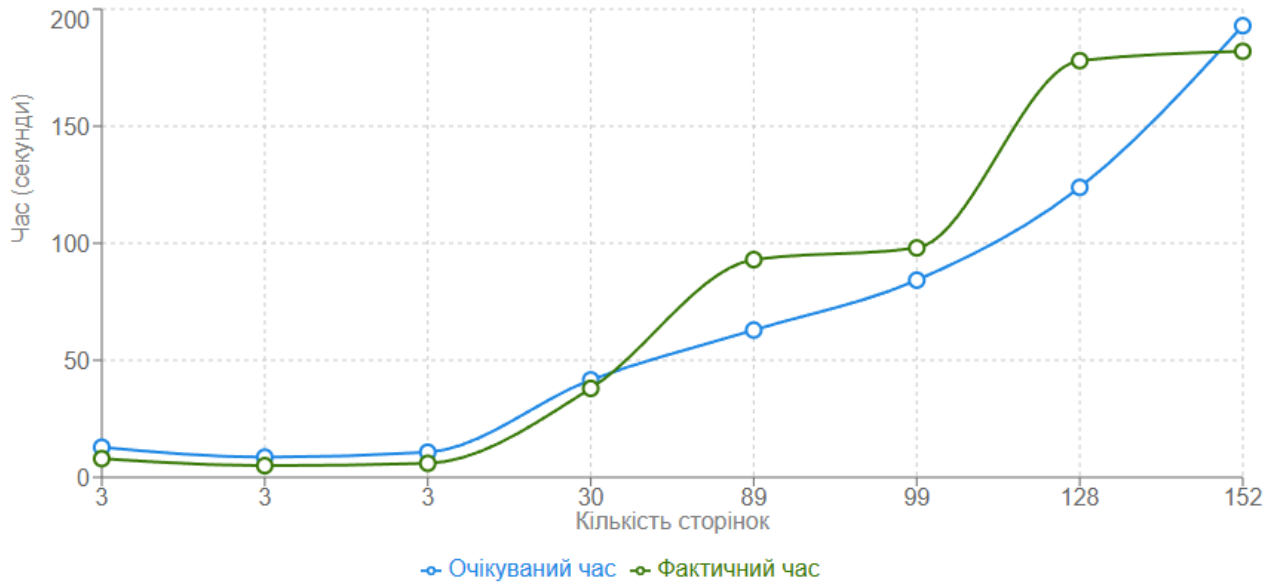


Рисунок 3.13 — Порівняння очікуваного та фактичного часу збору даних залежно від кількості сторінок після додавання коефіцієнта k

На рисунку 3.14 видно, що початкова модель (без коефіцієнта k) значно занижує прогнозований час, особливо при збільшенні кількості сторінок від 30 до 152, досягаючи відхилення у 2.5-3.97 разів. Оновлена модель з коефіцієнтом k демонструє набагато кращу апроксимацію реальних значень, особливо у діапазоні 89-152 сторінок.

Рисунок 3.15 кількісно підтверджує ефективність оптимізації: відносна похибка зменшилась у середньому з 60-75% до 10-30%. Найбільше покращення

спостерігається для середніх обсягів даних (30-99 сторінок). Проте для випадків із маленькою кількістю сторінок похибка навпаки збільшилась.

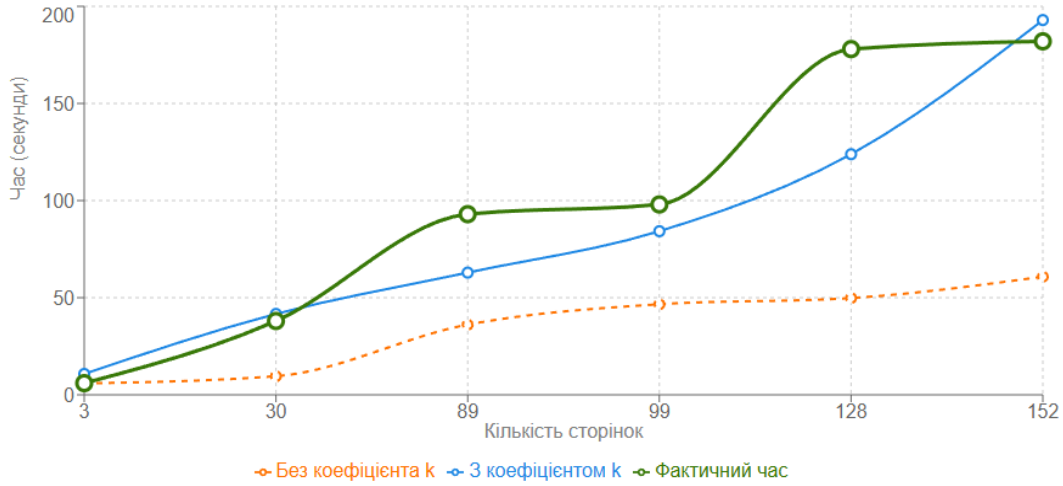


Рисунок 3.14 — Графік оцінки часу після додавання коефіцієнта k

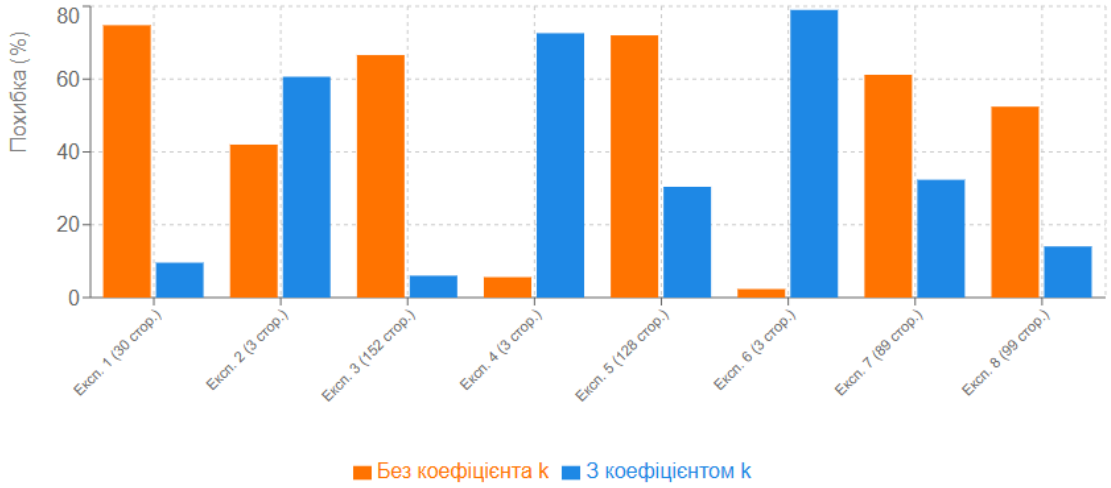


Рисунок 3.15 — Відносна похибка оцінки часу збору даних початкової та оновленої функцій

Отже, після додавання коефіцієнта k теоретичне обчислення часу виконання процесу збору даних стало ближчим до практичного, оскільки формула почала враховувати нерівномірність навантаження під час обробки вебсторінок, зміну швидкості мережі та збільшення часу обробки зі зростанням кількості сторінок. Проте похибка зберігається через те, що на фактичний час впливають додаткові

фактори, зокрема нестабільність роботи вебсерверів, затримки мережі, поведінка механізмів захисту від ботів тощо. Таким чином, попри покращення точності обчислення, повного усунення відхилень між теоретичним та фактичним значеннями досягти неможливо через динамічну природу вебскрепінгу.

Висновки до розділу 3

У цьому розділі було проведено технічний опис розробленої інформаційної системи для автоматизованого збору даних з інтернет-магазинів.

Реалізована архітектура забезпечує швидку обробку великих обсягів даних завдяки асинхронному підходу та механізму кешування. Гнучка система шаблонів надає можливість користувачам самостійно налаштовувати збір даних без необхідності програмування. Впровадження модуля прогнозування часу виконання дозволяє ефективно планувати процес збору даних, а автоматизація через заплановані завдання забезпечує постійну актуалізацію інформації.

4 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

У цьому розділі описано взаємодію користувача з програмною системою через її інтерфейс. Подано призначення та зміст основних сторінок — від головного вікна, реєстрації та входу до створення шаблонів, перегляду даних і експорту результатів. Матеріал демонструє послідовність роботи та зручність використання системи.

4.1 Опис головної сторінки

При вході до створеної вебсистеми, користувач потрапляє на головну сторінку, зображену на рисунку 4.1. Вгорі екрану закріплено елемент з кнопками навігації по системі, кожна з яких дає можливість перейти до відповідного розділу сайту після натискання на неї.

По центру екрану виведено кнопку, яка дає можливість користувачам перейти на сторінку створення шаблону.

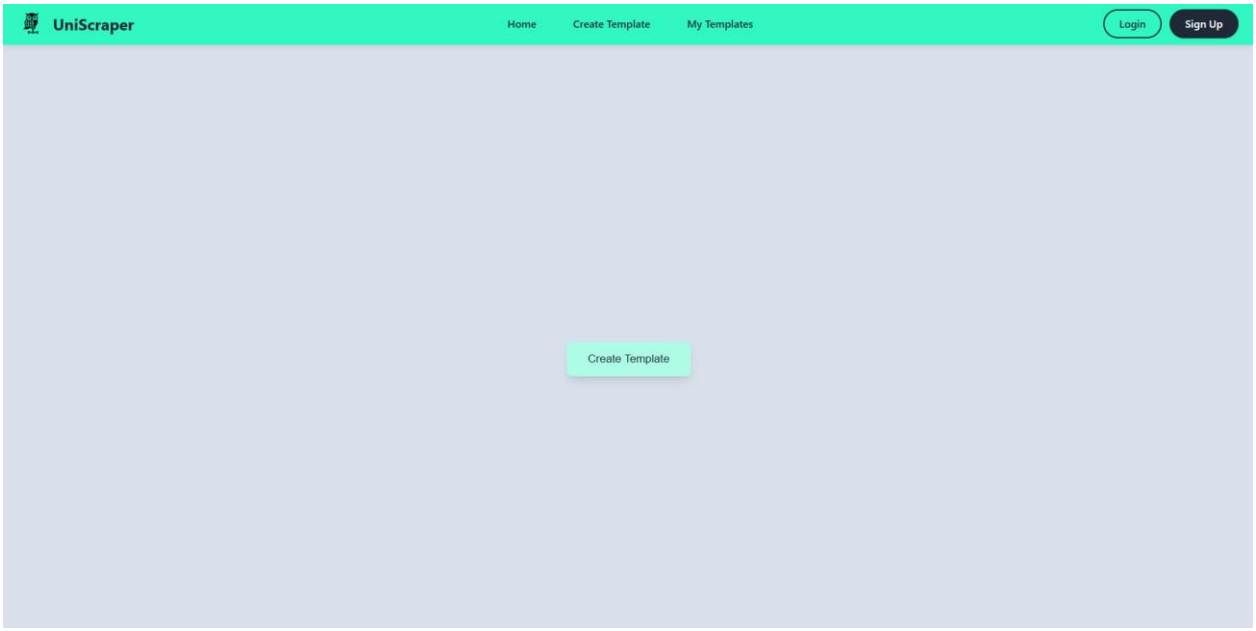


Рисунок 4.1 — Головний екран системи

4.2 Опис сторінки реєстрації

На рисунку 4.2 зображено початковий екран сторінки реєстрації нового користувача. Для переходу на неї необхідно натиснути на кнопку «Sign Up» на головному екрані, або сторінці авторизації користувача.

Ця сторінка містить поля для вводу електронної пошти користувача, імені, прізвища та пароля.

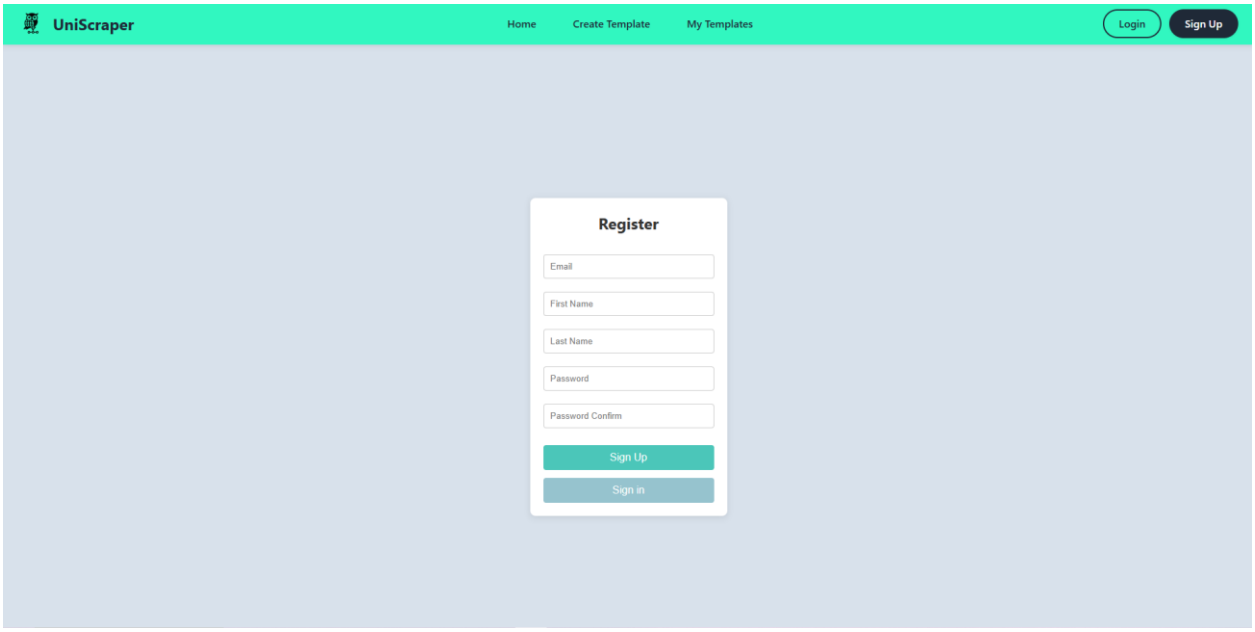


Рисунок 4.2 — Початковий екран сторінки реєстрації нового користувача

Заповнивши всі необхідні поля та натиснувши на кнопку «Submit», система виконує перевірку введених даних. У разі виявлення помилок, між полями форми та кнопками з'являється повідомлення про помилку червоного кольору, що інформує користувача про причину виникнення помилки.

Можливі варіанти помилок:

— якщо користувач не заповнив всі необхідні поля, відображається повідомлення «This field may not be blank.» (Це поле не може бути пустим);

— якщо електронна пошта вже зареєстрована в системі, відображається повідомлення «User with this email already exists.» (Користувач з такою електронною поштою вже існує);

— якщо пароль занадто короткий (менше 8 символів), система повідомляє «This password is too short. It must contain at least 8 characters.» (Цей пароль занадто короткий. Він повинен містити не менше восьми символів.);

— якщо пароль занадто простий або поширений, з'являється відповідне попередження «This password is too common.» (Цей пароль занадто поширений);

— якщо паролі в полях «Password» та «Password Confirm» не збігаються, користувач отримує відповідне повідомлення про помилку.

У разі успішної реєстрації користувача буде автоматично перенаправлено на сторінку авторизації для входу в систему з новоствореним обліковим записом.

Також внизу форми для реєстрації присутня кнопка «Sign In», яка дозволяє користувачам, які вже мають обліковий запис, швидко перейти на сторінку авторизації.

4.3 Опис сторінки входу в систему

На рисунку 4.3 зображено початковий екран сторінки реєстрації нового користувача. Для переходу на неї необхідно натиснути на кнопку «Sign Up» на головному екрані, або сторінці авторизації користувача. Ця сторінка містить поля для вводу електронної пошти користувача та пароля.

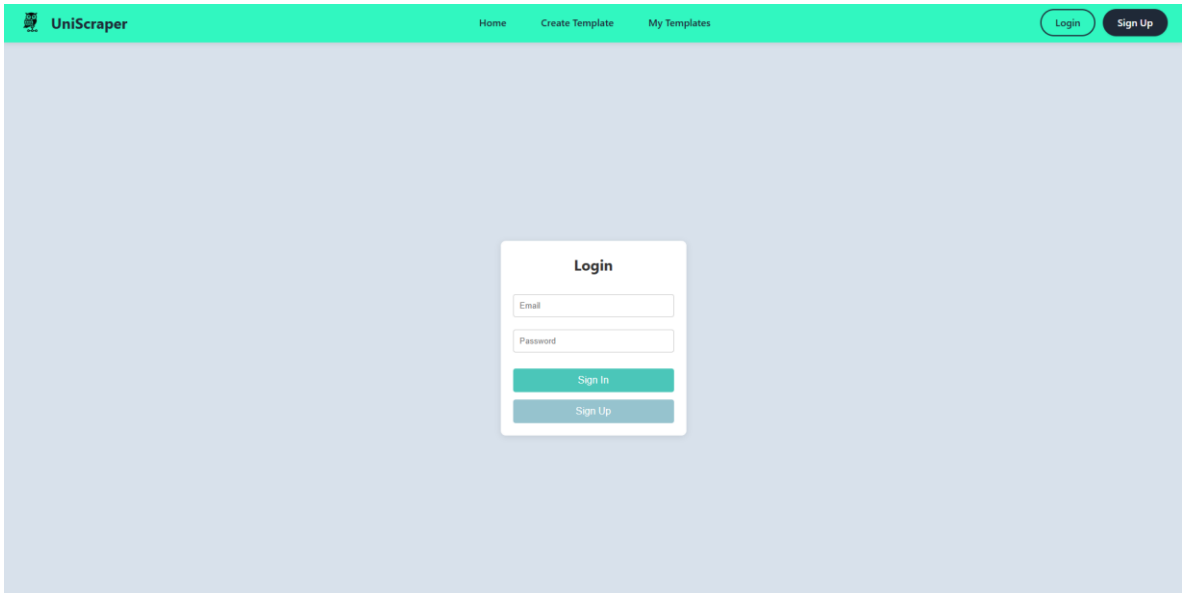


Рисунок 4.3 — Початковий екран сторінки авторизації нового користувача

Заповнивши всі необхідні поля та натиснувши на кнопку «Submit», система виконує перевірку введених даних. У разі виявлення помилок, між полями форми та кнопками з'являється повідомлення про помилку червоного кольору, що інформує користувача про причину виникнення помилки.

Можливі варіанти помилок:

— якщо користувач не заповнив всі необхідні поля, відображається повідомлення «This field may not be blank.» (Це поле не може бути пустим);

— якщо користувач ввів неправильну електронну пошту або пароль, відображається повідомлення «Incorrect email or password.» (Неправильна електронна пошта або пароль).

У разі успішної авторизації користувача буде автоматично перенаправлено на сторінку зі списком шаблонів для подальшої роботи з системою.

Також внизу форми для авторизації присутня кнопка «Sign Up», яка дозволяє користувачам, які ще не мають облікового запису, швидко перейти на сторінку реєстрації для створення нового профілю в системі.

4.4 Опис сторінки створення шаблону для збору даних

Натиснувши на кнопку «Create Template» в верхній частині екрану, користувача буде перенаправлено на сторінку створення шаблону (рис. 4.4).

У верхній частині сторінки розташоване поле для введення URL-адреси сторінки, з якої потрібно буде зібрати дані, та кнопка «Submit».

В нижній лівій частині сторінки знаходиться кнопка «Add Attribute», яка потрібна для додавання атрибута, який необхідно збирати зі сторінки. По центру розміщено форму для виставлення частоти здійснення процесу збору даних. В правій нижній частині знаходиться поле із запитанням про необхідність збору даних зі сторінок продуктів.

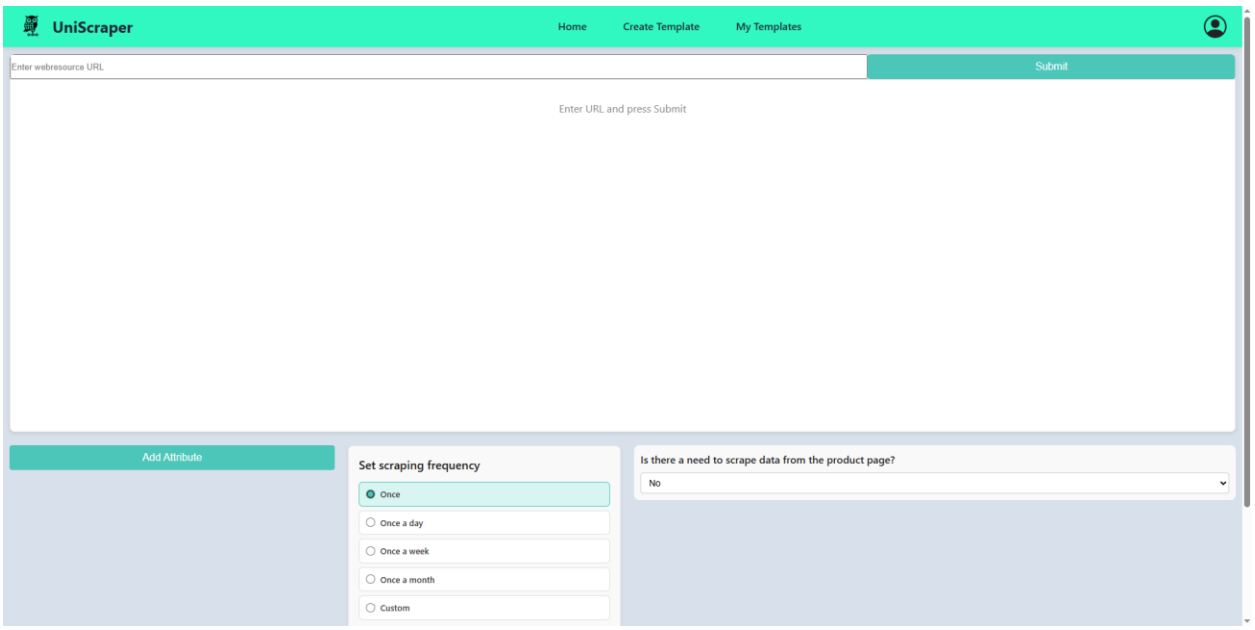


Рисунок 4.4 — Початковий екран сторінки створення шаблону

Після того, як користувач ввів посилання в поле та натиснув кнопку «Submit», система надсилає запит на сервер для отримання HTML-вмісту вказаної вебсторінки. Протягом обробки завантаження відповідної сторінки, у вікні користувачу відображатиметься напис «Loading…».

У випадку, якщо серверу не вдасться завантажити вміст сторінки, користувачу буде виведено повідомлення про помилку «Не вдалося завантажити сторінку: HTTP error! Status: 500».

Якщо вдалось успішно завантажити сторінку, то її буде відображено у відповідному вікні (рис. 4.5). Завантажена сторінка відображатиметься майже ідентично до оригіналу — якщо відкрити це саме посилання у звичайному браузері, вигляд сторінки буде практично однаковим. Це забезпечує зручність роботи, оскільки користувач бачить знайому структуру вебресурсу та може легко орієнтуватися при виборі елементів для збору.

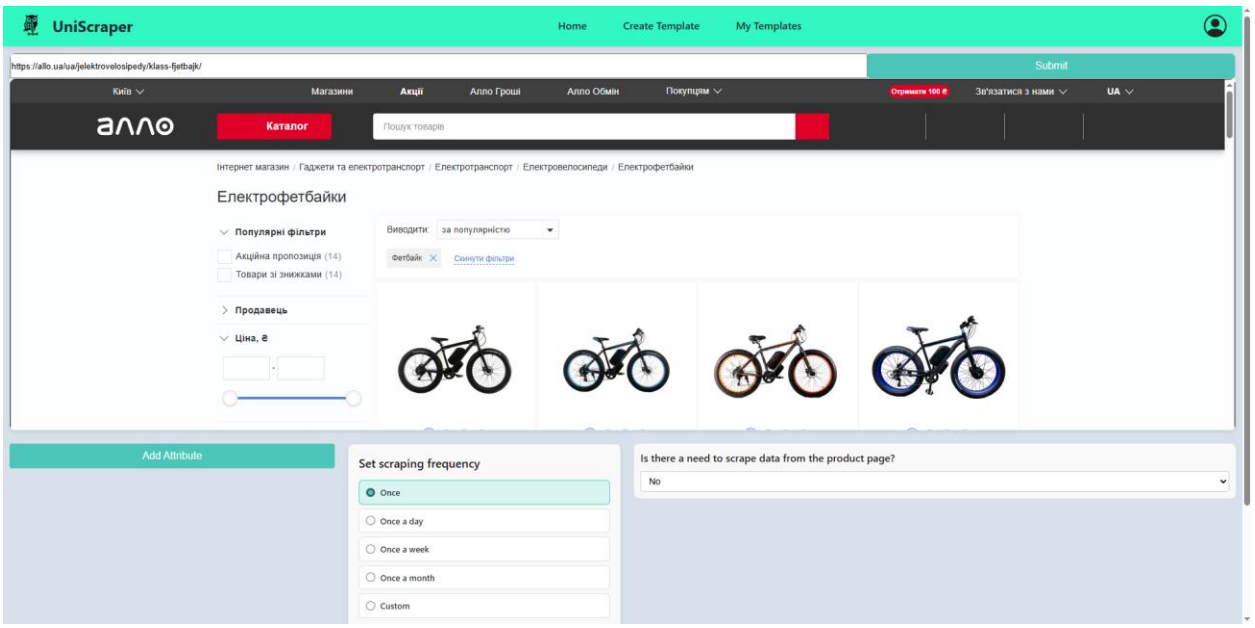


Рисунок 4.5 — Результат відображення цільової сторінки

Після того, як сторінку за посиланням було завантажено, користувач може додати атрибути, дані з яких потрібно зібрати. Щоб це зробити, користувачу потрібно натиснути на кнопку «Add Attribute», після чого на сторінці з’явиться форма для додавання атрибута.

Щоб додати новий атрибут, потрібно вказати назву атрибута, тип та обрати елемент цільової сторінки, значення якого потрібно зібрати. Для того, щоб обрати елемент, користувачу потрібно навестись на нього у вікні відображення цільової сторінки та натиснути лівою кнопкою миші. Після цього тег та атрибути цього тегу будуть записані в формі додавання нового атрибуту (рис. 4.6). Для збереження атрибута, потрібно натиснути на кнопку «Save».

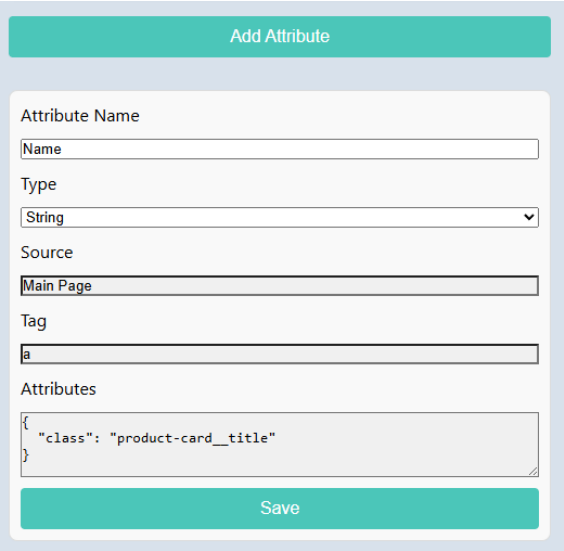


Рисунок 4.6 — Результат натискання на елемент цільової сторінки

Після збереження атрибута, на сторінці з’явиться кнопка «Save Template», за допомогою якою якої можна зберегти створений шаблон. Також в правій частині екрану з’явиться таблиця «Table of attributes» (рис. 4.7), в якій відображаються всі додані атрибути та дані про них. Видалити атрибут з таблиці можна, натиснувши на відповідну кнопку «Delete».

Після збереження атрибута внизу сторінки також відобразиться таблиця з прикладом даних, які будуть зібрані при поточному наборі атрибутів (рис. 4.7).

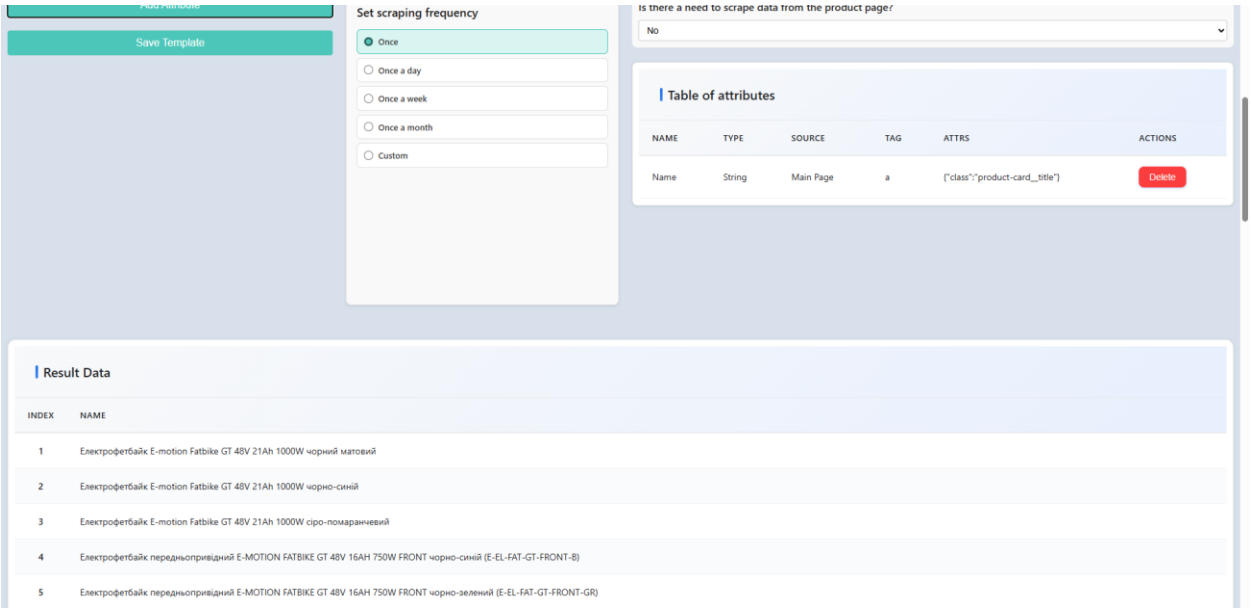


Рисунок 4.7 — Приклад зібраних даних, для поточних атрибутів

Користувач також має можливість зібрати дані зі сторінок продуктів. Для цього потрібно вибрати «Yes» для поля з написом “Is there a need to scrape data from the product page?” та ввести посилання на сторінку продукту у відповідне поле.

Після того, як користувач натиснув на кнопку «Submit», система надсилає запит на сервер для отримання HTML-вмісту вказаної вебсторінки, аналогічно до основної сторінки. Так сам, як і у випадку із головною сторінкою, під час завантаження сторінки товару, у користувача відображатиметься напис «Loading…», у вікні сторінки товару. Після завантаження HTML-вмісту сторінки товару, на екрані користувача буде виведено вміст сторінки товару. Відмальовану сторінку товару можна згорнути.

Система дає можливість збирати дані зі сторінок товарів, наприклад на рисунку 4.7 зображено два атрибути, які потрібно збирати зі сторінки товару (source = «Product Page»).

Також користувач має можливість виставити частоту виконання збору даних для шаблону. Вибравши «Once», збір даних відбудеться один раз. Також є варіанти збору даних раз на день, тиждень та місяць. Крім того, обравши варіант «Custom», можна задати власну частоту збору даних в діапазоні від однієї години до одного року.

Після того, як користувач натисне на кнопку «Save Template», по центру екрану з’явиться форма для введення назви шаблону (рис. 4.29). Ввівши назву шаблону та натиснувши на кнопку «Save», шаблон буде збережено та автоматично запущено процес збору даних, згідно із налаштуваннями шаблону, а користувача буде перенаправлено на сторінку новоствореного шаблону. Натиснувши на кнопку «Cancel» (рис. 4.8), форма закриється і користувач зможе продовжити налаштування поточного шаблону.

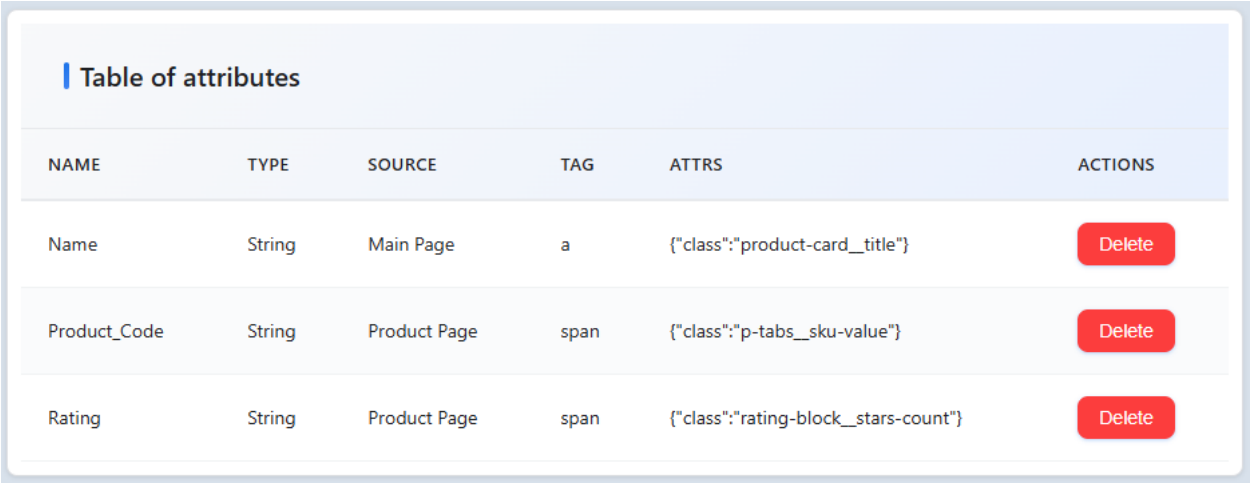


Рисунок 4.7 — Таблиця атрибутів із атрибутами зі сторінки товару

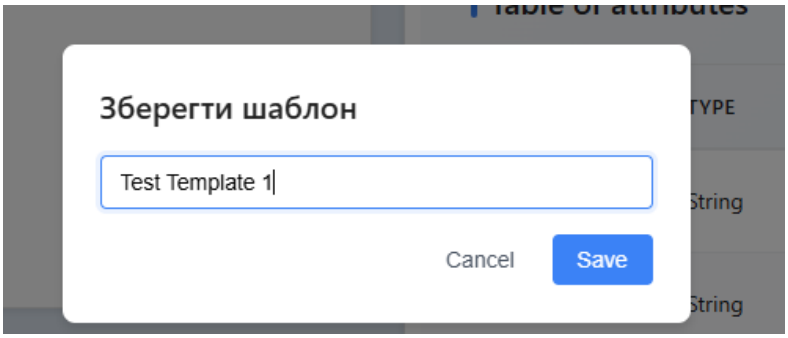


Рисунок 4.8 — Форма збереження шаблону

4.5 Опис сторінки списку шаблонів

Для того, щоб користувач потрапив на сторінку зі списком збережених шаблонів, йому необхідно натиснути на кнопку «My Templates» вгорі екрану, після чого на екрані користувача з’явиться список створених ним шаблонів (рис. 4.9).

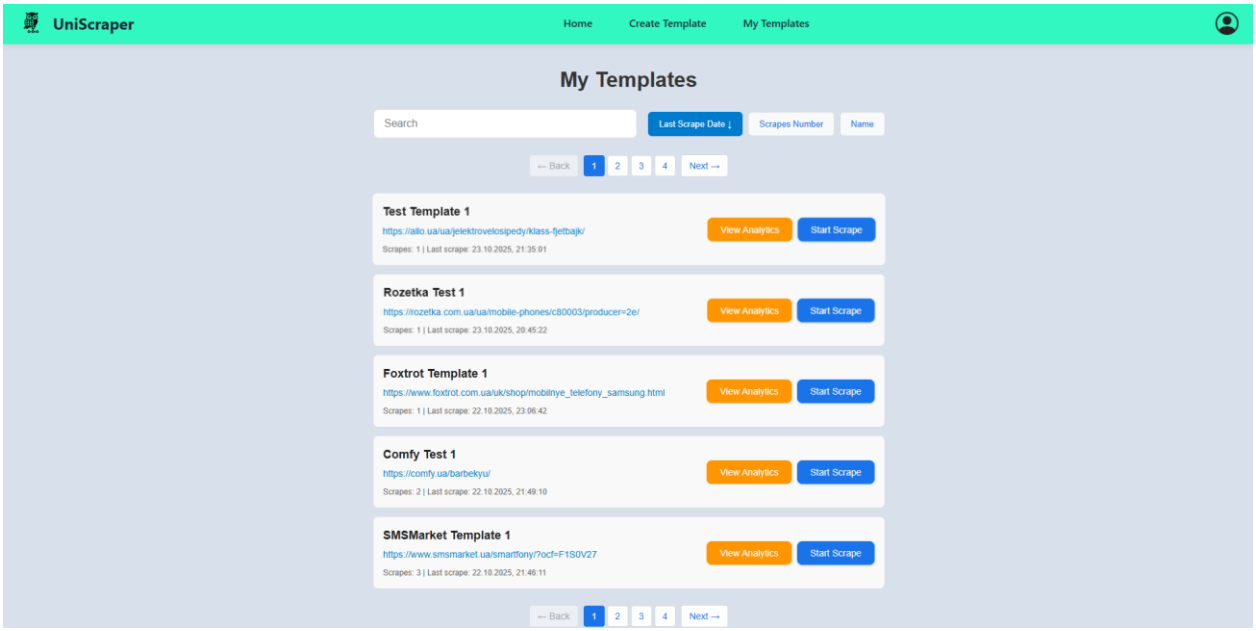


Рисунок 4.9 — Початковий вигляд сторінки шаблонів користувача

За замовчуванням список шаблонів відсортований від шаблону, в якому збір даних проводився останнім, до шаблону того, в якому збір даних проводився найдавніше. Також користувач має можливість відсортувати список шаблонів за кількістю проведених зборів даних та за іменем шаблону в алфавітному порядку.

Кожна сторінка може містити до 5-ти шаблонів, а переміщення між сторінками можна виконувати за допомогою кнопок «Next» та «Back», або натискаючи безпосередньо на номер потрібної сторінки.

Користувач також має можливість фільтрувати список шаблонів, використовуючи поле із написом «Search». Для здійснення фільтрування користувачу необхідно ввести текст, після чого буде відфільтровано шаблони, імена яких містять введений користувачем текст.

На кожному шаблоні на сторінці списку шаблонів, користувач може знайти ім’я шаблону, посилання, за яким збираються дані, кількість проведених процесів збору даних та дату останнього збору даних.

Також для кожного шаблону наявні дві кнопки: «View Analytics» та «Start Scrape». Натиснувши на кнопку «View Analytics», користувача буде перенаправлено на сторінку відповідного шаблону, а при натисканні кнопки «Start Scrape», буде розпочато процес збору даних для відповідного шаблону.

4.6 Опис сторінки шаблону

Якщо користувача було перенаправлено на сторінку шаблону одразу після його створення, то сторінка шаблону буде виглядати, як зображено на рисунку 4.10. Кнопка «Start Scrape» буде сірою та на неї не можна буде натиснути, оскільки процес збору даних було автоматично розпочато після створення шаблону. Новостворений шаблон також матиме порожні блоки зі статистикою та історією зібраних даних (рис. 4.11).

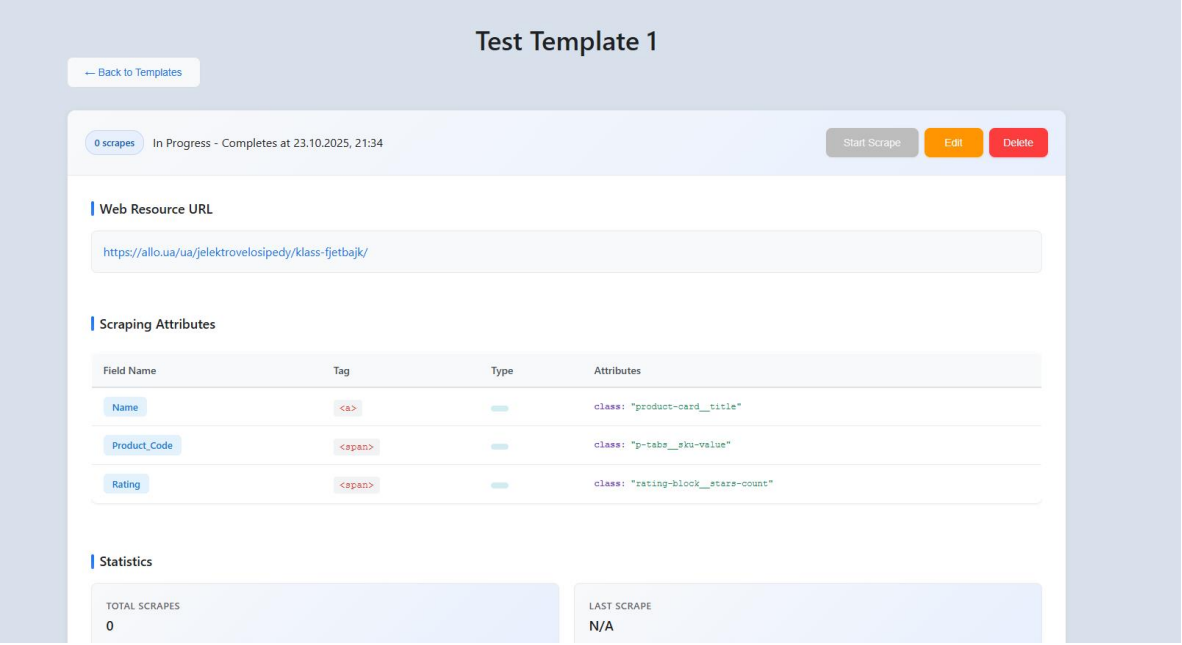


Рисунок 4.10 — Початкова сторінка шаблону одразу після його створення

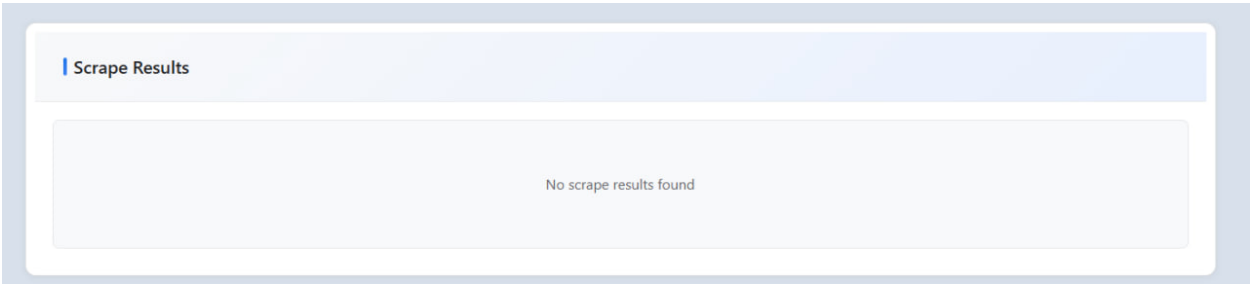


Рисунок 4.11 — Пустий блок з історією зібраних даних

Після того, як процес збору даних закінчився, кнопка «Start Scrape» стала зеленою, що означає, що можна знову почати процес збору даних (рис. 4.12).

Розділ статистики та історії зібраних даних також оновились, як зображено на рисунках 4.13 та 4.14 відповідно.

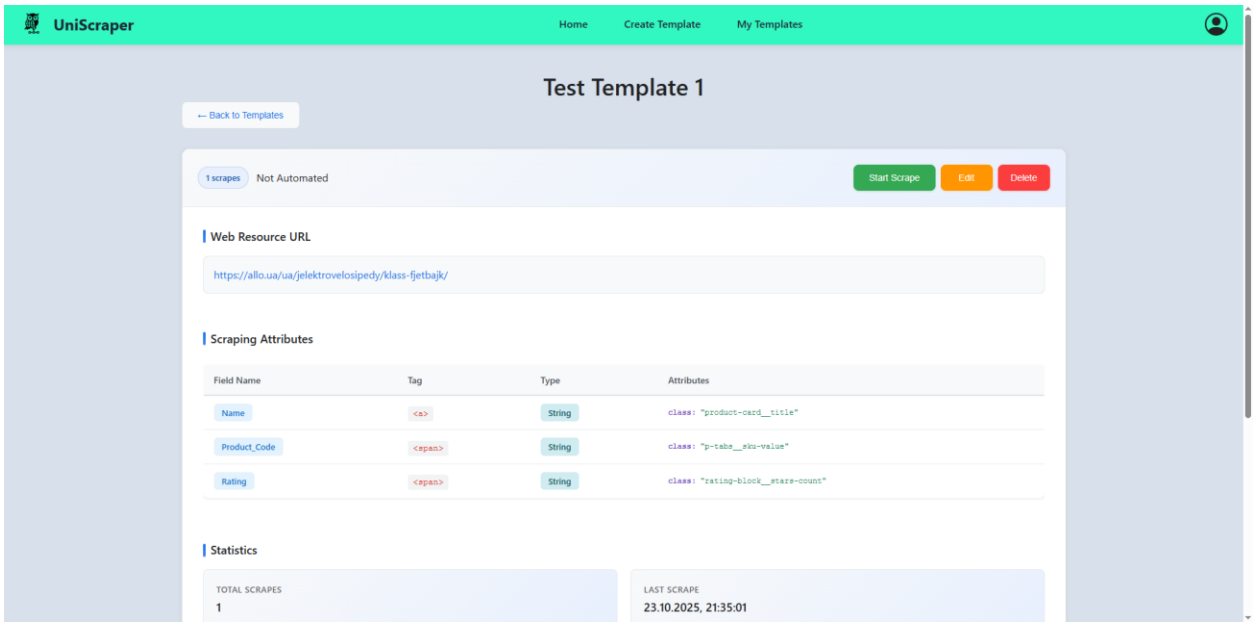


Рисунок 4.12 — сторінка шаблону після закінчення початкового збору даних

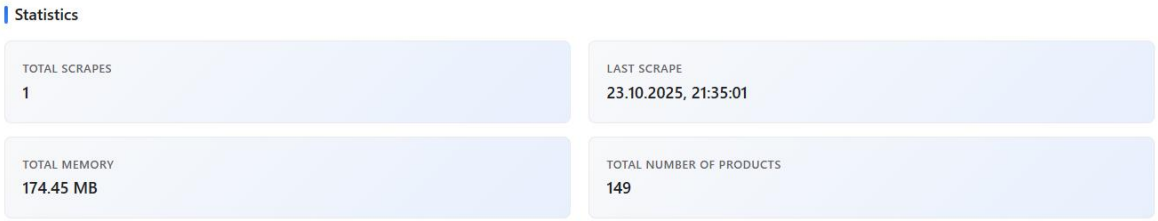


Рисунок 4.13 — блок статистики після закінчення початкового збору даних

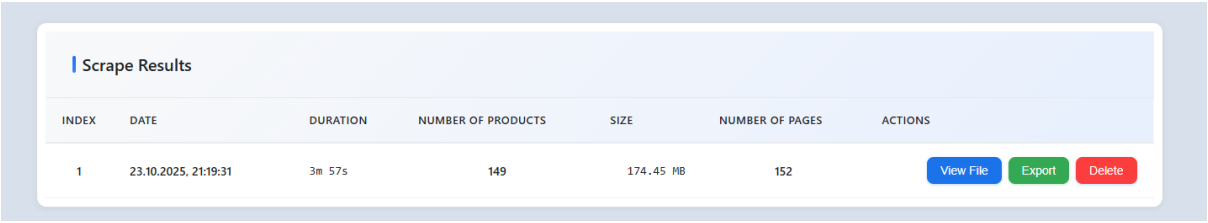


Рисунок 4.14 — блок історії зборів даних після закінчення початкового збору даних

Блок статистики містить наступні дані про шаблон: загальна кількість проведених зборів даних, дата та час завершення останнього збору даних, загальний об’єм використаної пам’яті та загальну кількість зібраних товарів.

Також, якщо для цього шаблону було проведено процес збору даних 2 і більше разів, тоді для нього також відображатиметься графік на основі історії проведених зборів даних для цього шаблону (рис. 4.15). Можна вивести графіки за витраченим часом, об’ємом витраченої пам’яті, кількістю продуктів та кількістю зібраних вебсторінок.

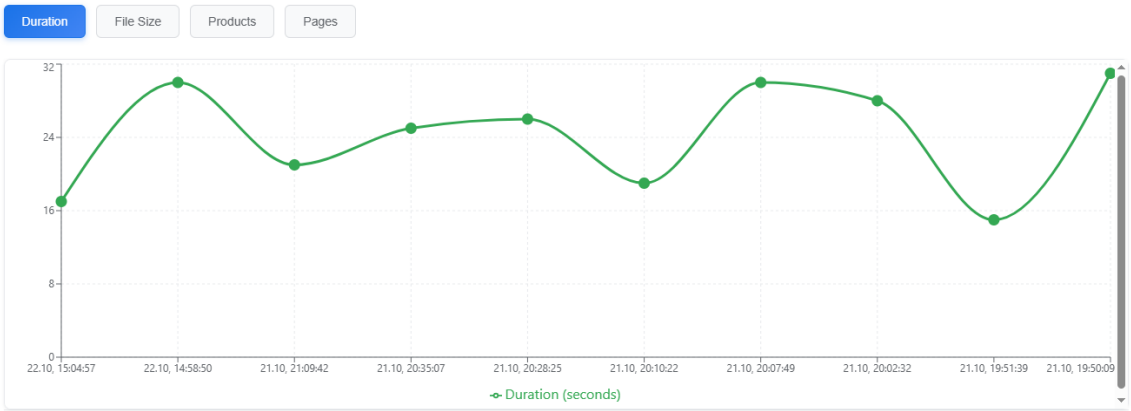


Рисунок 4.15 — Графік історії зборів даних за часом

Внизу сторінки знаходиться блок історії зборів даних для поточного шаблону. Оскільки збір для шаблону вище проводився тільки один раз, в таблиці є тільки один запис (рис. 4.16).

Кількість зібраних вебсторінок становить 152, а кількість зібраних продуктів 149. Це викликано необхідністю збору даних зі сторінок продуктів. Отже, необхідно було зібрати 149 сторінок продуктів та ще 3 сторінки, на яких розміщувались посилання на сторінки продуктів.

На рисунку 4.17, наприклад, зображено таблицю історії зборів даних, із значенням колонки «Number of pages» дорівнює лише 3, оскільки в цьому шаблоні дані не збираються зі сторінок продуктів.

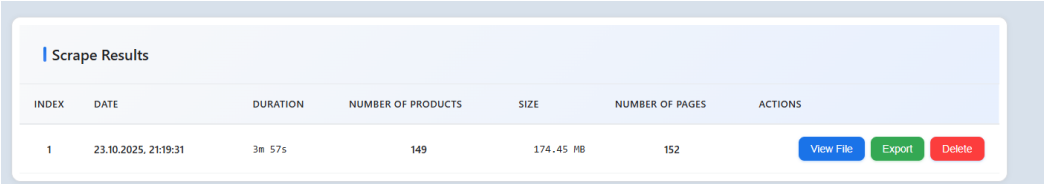


Рисунок 4.16 — Історія зборів даних для створеного шаблону

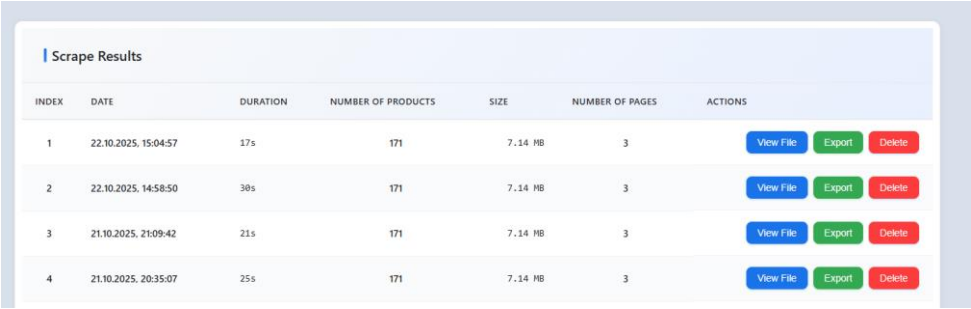


Рисунок 4.17 — Приклад історії зборів даних для іншого шаблону

Кожен рядок з блоку історії збору даних містить колонки індекс, дата завершення збору даних, тривалість, кількість зібраних продуктів, об’єм витраченої пам’яті та кількість зібраних вебсторінок. Також для кожного зібраного файлу є 3 кнопки. Натиснувши на кнопку «View File» користувача буде перенаправлено на сторінку перегляду відповідного файлу. Кнопка «Export» дає можливість експортувати зібрані дані в різних форматах. Після натискання на кнопку «Delete», відповідний файл буде видалено з історії зборів даних.

4.7 Сторінка перегляду файлу

Користувач може перейти на сторінку перегляду файлу, натиснувши на кнопку «View File» для рядка з таблиці історії зборів даних на сторінці перегляду шаблону.

На сторінці перегляду файлу користувач може побачити таблицю із зібраними даними. У верхній лівій частині таблиці написано дату та час закінчення збору даних для цього файлу (рис. 4.18). В правій верхній частині користувач може побачити загальну кількість продуктів в цьому файлі.

Також користувач може експортувати файл з даними в різних форматах, натиснувши на кнопку «Export», та видалити файл, натиснувши на кнопку «Delete».

За замовчуванням продукти невідсортовані, проте користувач має можливість провести сортування за чи проти алфавітного порядку, або за збільшенням чи зменшенням числа для кожної колонки в таблиці.

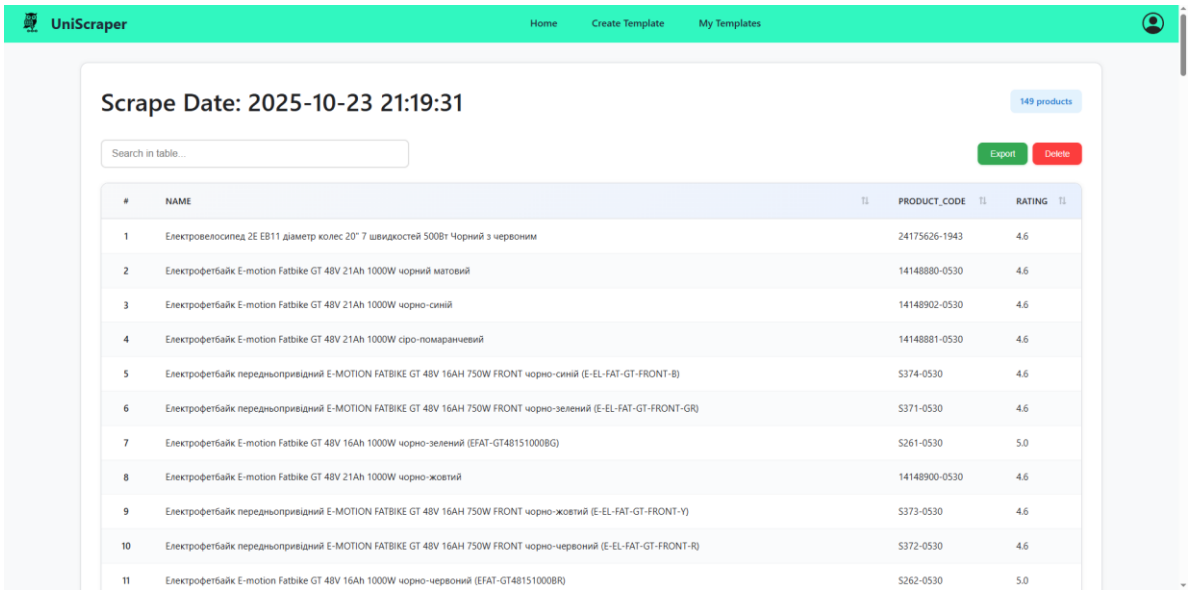


Рисунок 4.18 — початковий екран сторінки перегляду файлу

Також на сторінці є фільтр для пошуку необхідних рядків, який шукає збіг для кожного значення кожного рядка.

4.8 Вікно експорту даних

Відкрити вікно експорту файлу з даними можна натиснувши на кнопку «Export» на сторінках перегляду шаблону або файлу. Файл із зібраними даними можна експортувати у наступних форматах: Excel, CSV, JSON, Google Sheets, SQL, TXT та TSV (рис. 4.19). Для здійснення експорту достатньо натиснути на потрібний формат. Для того, щоб закрити вікно експорту, потрібно натиснути на кнопку «Cancel».

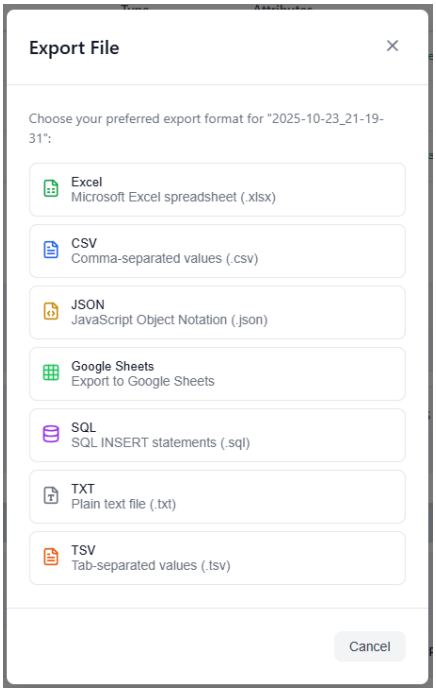


Рисунок 4.19 — Вікно експорту зібраних даних

Висновки до розділу 4

У цьому розділі було детально описано процес взаємодії користувача зі створеною програмною системою, що дозволяє на практиці оцінити зручність, логічність та послідовність роботи її інтерфейсу. Також було розглянуто структуру основних сторінок, починаючи від головної та сторінок авторизації, які забезпечують доступ до функціоналу платформи, і завершуючи сторінками створення, перегляду та керування шаблонами збору даних.

В додатку Д зображено загальний алгоритм роботи користувача із системою збору даних, а в додатку Ж – діаграму послідовності роботи користувача із системою збору даних.

5 РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

У цьому розділі розглядається процес розроблення стартап-проєкту, що включає формування його ідеї, аналіз технічної здійсненності та оцінку ринкових умов для впровадження. Також досліджуються можливості та загрози ринку, конкуренція, цільові групи споживачів і фактори конкурентоспроможності продукту. Визначаються базова стратегія розвитку, позиціонування та маркетингова програма, які забезпечують ефективний вихід проєкту на ринок.

5.1 Опис ідеї проєкту

Початковим етапом розроблення стартап-проєкту є формування чіткого та структурованого опису ідеї. Це включає визначення сутності продукту, аналіз можливих напрямків його використання та вигоди, яку отримає кінцевий користувач. Зведена інформація опису ідеї стартапу наведена в таблиці 5.1.

Таблиця 5.1 — Опис ідеї стартап-проєкту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Створення інформаційної системи, що автоматизовано збирає дані про товари з інтернет-магазинів.	Порівняльні сервіси, агрегатори даних	Можливість швидко порівняти ціни з десятків магазинів та вибрати найвигіднішу пропозицію.
	Моніторинг конкурентів для інтернет-магазинів	Оперативне відстеження змін цін і наявності товарів у конкурентів без ручної роботи.
	Маркетингові агентства	Отримання якісних даних для досліджень, аналітичних звітів і прогнозування ринку.
	Системи рекомендацій та AI-моделі	Живлення алгоритмів реальними даними для формування рекомендацій та прогнозів.

Зміст ідеї	Напрямки застосування	Вигоди для користувача
	Системи рекомендацій та AI-моделі	Живлення алгоритмів реальними даними для формування рекомендацій та прогнозів.
	Платформи автоматизації закупівель	Можливість швидко порівнювати постачальників і формувати найвигідніші закупівлі.
	Маркетплейси електронної комерції	Підтримка актуальності каталогу — автоматичне оновлення цін, фото, описів тощо.
	Дослідники ринку та стартапи	Швидкий доступ до великих наборів даних для перевірки гіпотез і побудови мінімально життєздатного продукту.

Отже, ідея стартап-проєкту полягає у створенні SaaS-платформи (Software as a Service) для автоматизованого збору та аналізу даних про товари з інтернет-магазинів. Система надає можливість користувачам без технічних навичок налаштовувати збір даних, отримувати структуровані дані про товари в інтернет-магазинах, а також експортувати їх.

У таблиці 5.2 подано порівняльний аналіз техніко-економічних характеристик проєкту та його конкурентів. Це дозволяє визначити сильні, слабкі та нейтральні сторони ідеї, а також оцінити її загальну конкурентоспроможність.

Таблиця 5.2 — Визначення сильних, слабких та нейтральних характеристик ідеї проєкту

Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
	Мій проєкт	ParseHub	Price2Spy			
Швидкість збору даних	Висока	Середня	Висока		+	
Актуальність та повнота даних	Висока	Середня	Середня			+
Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			W (слабка сторона)	N (нейтральна	S (сильна сторона)
	Мій	ParseHub	Price2Spy			

	проєкт				сторона)	
Гнучкість налаштування	Дуже висока	Середня	Низька			+
Захист від блокувань	Середній	Низький	Високий		+	
Можливість масштабування	Висока	Середня	Висока		+	
Інтеграції з зовнішніми системами	Середня	Середня	Висока	+		
Легкість реалізації алгоритмів обходу захистів сайтів	Середня	Низька	Висока		+	
Збереження історії змін	Є	Є	Є		+	
Зручність використання	Висока	Висока	Середня		+	
Вартість використання	Середня	Висока	Висока			+
Адаптованість до українського ринку	Висока	Низька	Низька			+

На основі проведеного порівняння можна зробити висновок, що ідея проєкту має чітко виражені сильні сторони, серед яких ключовими є висока гнучкість налаштування, можливість масштабування, глибина та повнота зібраних даних, а також оптимальна вартість використання. Важливою перевагою є також адаптованість системи до українського ринку, що дозволяє ефективно працювати з українськими інтернет-магазинами. Це забезпечує конкурентоспроможність системи порівняно як із універсальними безкодовими інструментами, так і з вузькоспеціалізованими платформами для моніторингу цін, які не орієнтовані на український сегмент.

Слабкі сторони проєкту головним чином пов’язані з обмеженим рівнем автоматичного обходу захистів сайтів та менш розвиненими функціональними можливостями інтеграцій, порівняно з професійними аналітичними сервісами. Проте такі недоліки не є критичними та можуть бути усунені на етапах подальшої розробки.

5.2 Технологічний аудит ідеї проєкту

Наступним етапом є проведення аналізу техніко-економічних характеристик ідеї стартап-проєкту, що дозволяє оцінити її конкурентоспроможність, технологічну здійсненність та потенціал подальшого розвитку. Результати проведеного аналізу наведені у таблиці 5.3.

Таблиця 5.3 — Технологічна здійсненність проєкту

№ п/п	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Створення REST API	Python, Django, DRF	Технології наявні	Технології доступні безкоштовно
2	Створення підсистеми збору даних	Python, requests, Selenium, BeautifulSoup	Технології наявні	Технології доступні безкоштовно
3	Побудова вебінтерфейсу користувача	React, TypeScript, HTML, CSS	Технології наявні	Технології доступні безкоштовно
4	Збереження даних платформи	SQLite, JSON, Redis	Технології наявні	Технології доступні безкоштовно
5	Забезпечення автентифікації та авторизації	OAuth, JWT	Технології наявні	Технології доступні безкоштовно
6	Автоматизоване виконання збору даних	DjangoQ, Redis	Технології наявні	Технології доступні безкоштовно
7	Забезпечення захисту від блокування	Proxy, User-Agent, Rate limiting	Технології наявні	Доступні частково — безкоштовні обмежено, платні за підпискою

Необхідні технології для реалізації проєкту наявні та є доступними авторам. Усі компоненти від серверної частини та вебінтерфейсу, до інструментів збору даних і захисту від блокувань, можуть бути впроваджені на основі доступних рішень. Отже, технологічна реалізація проєкту є можливою.

5.3 Аналіз ринкових можливостей запуску стартап-проєкту

Далі проводиться аналіз ринкового середовища проєкту, що включає оцінку можливостей і загроз, рівня конкуренції, темпів розвитку ринку та основних вимог до входження в галузь. Такий аналіз дозволяє визначити перспективність впровадження системи автоматизованого збору даних, її конкурентні переваги та потенційні ризики. Узагальнені результати наведено в таблицях 5.4–5.14, де таблиця 5.4 містить попередню характеристику цільового ринку.

Таблиця 5.4 — Попередня характеристика потенційного ринку стартап-проєкту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	8-10
2	Загальний обсяг продаж, грн/ум.од	$1.03 мільярда
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Середні: конкуренція, необхідність надійного анти-блокування, проксі, технічна інфраструктура
5	Специфічні вимоги до стандартизації та сертифікації	Відповідність GDPR, robots.txt, “Terms of Service” вебсайтів; ISO/IEC 27001
6	Середня норма рентабельності в галузі (або по ринку), %	25-40%

За результатами аналізу таблиці можна зробити висновок, що ринок є привабливим для входження за попереднім оцінюванням. Динаміка ринку демонструє стабільне зростання, середня норма рентабельності є високою, що підтверджує перспективність цього стартап-проєкту.

Хоча на ринку присутні від восьми до десяти великих конкурентів, більшість із них орієнтовані на міжнародний сегмент і не мають виразної присутності чи локалізації в українському сегменті, що створює вікно можливостей для виходу на місцевий ринок та формування конкурентної переваги за рахунок адаптації продукту під потреби українських користувачів.

У таблиці 5.5 визначено характеристику основних груп потенційних клієнтів стартап-проєкту, їх вимоги та ключові потреби.

Таблиця 5.5 — Характеристика потенційних клієнтів стартап-проєкту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Автоматизація збору даних про товари конкурентів для моніторингу цін	Малий та середній бізнес (інтернет-магазини, продавці на маркетплейсах)	Обмежений бюджет, потреба у простих рішеннях без технічних знань, фокус на швидкому впровадженні	Простота налаштування без програмування, доступна ціна, технічна підтримка
2	Збір даних для аналітики ринку та конкурентного аналізу	Маркетингові агенції, аналітичні відділи компаній	Потреба у великих обсягах даних та інтеграції з аналітичними інструментами	Можливість експорту в різних форматах, можливість інтеграції через API, обробка великих обсягів даних
3	Моніторинг наявності товарів та цін	Великі торговельні мережі, роздрібна торгівля	Високі вимоги до стабільності, корпоративні стандарти безпеки, регулярність збору	Висока надійність роботи, захист даних, підтримка роботи з великими каталогами
4	Дослідження ринку перед запуском нових продуктів	Стартапи, підприємці, менеджери продуктів	Епізодичність використання, обмежений бюджет, самостійне використання	Гнучке ціноутворення, зрозумілий інтерфейс, швидке отримання результатів
5	Агрегація даних для порівняльних сервісів	Сервіси порівняння цін, агрегатори товарів	Потреба в актуальності, безперервності й швидкості збору	Висока швидкість збору, автоматичне оновлення інформації, зберігання результатів збору в структурованому вигляді

Також було проведено аналіз ринкового середовища, зокрема визначено ключові загрози та можливості для реалізації проєкту. Результати відображено в таблицях 5.6 та 5.7.

Таблиця 5.6 — Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Посилення захисту вебсайтів від автоматизованого збору даних	Впровадження систем захисту від ботів, CAPTCHA, динамічних токенів, що ускладнює збір даних	Розробка та постійне оновлення методів обходу захисту, використання ротації проксі-серверів, імітація поведінки людини, застосування штучного інтелекту для адаптації
2	Зростання конкуренції	Поява нових конкурентів на ринку з дешевшими або безкоштовними рішеннями, розвиток відкритого програмного забезпечення	Фокус на унікальних перевагах (простота використання без програмування), покращення користувацького інтерфейсу, додаткові послуги (аналітика, інтеграції), програми лояльності
3	Технологічні зміни вебсайтів	Перехід на нові технології (SPA, PWA), часті зміни структури сайтів, що потребує постійного оновлення програм для збору даних	Розробка адаптивних алгоритмів розпізнавання структури сторінок, використання штучного інтелекту для автоматичного визначення змін, швидка реакція на зміни
4	Обмеження продуктивності та блокування IP	Блокування IP-адрес при перевищенні ліміту запитів, уповільнення роботи сервісу	Використання розподілених проксі-мереж, регулювання частоти запитів, інтелектуальне розподілення навантаження, партнерство з постачальниками проксі

Таблиця 5.7 — Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Цифровізація торгівлі та зростання електронної комерції	Стрімке зростання кількості інтернет-магазинів, збільшення обсягів онлайн-торгівлі на 20-30% щорічно в Україні	Активний маркетинг серед інтернет-магазинів, розробка галузевих рішень (одяг, електроніка, продукти), партнерство з платформами для онлайн продажу
2	Зростання попиту на рішення, засновані на даних	Компанії все більше покладаються на дані для прийняття рішень, зростання інвестицій у бізнес-аналітику	Позиціонування як платформи для аналітики даних, інтеграція з інструментами аналізу, розробка аналітичних модулів
3	Недостатність технічних навичок у цільової аудиторії	Більшість власників малого бізнесу не мають навичок програмування, але потребують автоматизації	Акцент на підході без програмування (no-code) в маркетингу, розробка візуального конструктора, навчальні матеріали, персоналізована підтримка
4	Розвиток штучного інтелекту та машинного навчання	Можливість використання штучного інтелекту для автоматичного розпізнавання структури даних, покращення якості збору інформації	Впровадження модулів штучного інтелекту для автоматичного збору даних, прогнозної аналітики, обробки тексту для розпізнавання товарів
5	Глобалізація та міжнародна торгівля	Зростання потреби у моніторингу цін на міжнародних ринках, розвиток міжнародної електронної комерції	Розширення підтримки міжнародних сайтів, мультивалютність, локалізація інтерфейсу
6	Тренд на автоматизацію бізнес-процесів	Компанії активно шукають рішення для автоматизації рутинних завдань, високий рівень повернення інвестицій	Розробка інтеграцій з популярними системами управління, API для автоматизації, розширення функціональних можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
7	Вузькоспеціалізовані ринки та галузеві рішення	Можливість спеціалізації на конкретних галузях з унікальними потребами	Розробка галузевих шаблонів, партнерство з галузевими асоціаціями
8	Відсутність аналогічних систем в Україні	Наявність незаповненого ринку для подібних рішень	Просування продукту як першого в Україні, формування ринку та навчання користувачів

Крім того, було проведено ступеневий аналіз конкуренції на ринку, результати якого наведені у таблиці 5.8.

Таблиця 5.8 — Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Тип конкуренції — монополістична	На ринку присутня велика кількість компаній (ParseHub, Import.io, Apify), які пропонують різні за функціональністю та ціною продукти	Вирізнення завдяки простоті використання без необхідності програмування, фокус на український ринок та локалізацію, конкурентне ціноутворення, унікальний користувацький інтерфейс
2. За рівнем конкурентної боротьби — міжнародна	Конкуренція відбувається на глобальному рівні, оскільки більшість рішень доступні онлайн з будь-якої точки світу	Початковий фокус на український ринок з подальшою міжнародною експансією, локалізація інтерфейсу, підтримка локальних платіжних систем, конкурентні ціни для розвинутих ринків
3. За галузевою ознакою — внутрішньогалузева	Конкуренція в межах галузі автоматизованого збору даних та вебскрепінгу	Постійне вдосконалення технологій збору даних, слідкування за галузевими трендами, інновації в AI/ML, активна участь у професійних спільнотах

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
4. Конкуренція за видами товарів товарно-родова та товарно-видова	Конкуренція як між різними підходами до збору даних (кодові та безкодові рішення), так і між схожими безкодовими рішеннями	Чітке позиціонування як безкодового рішення для користувачів без технічних навичок, розвиток унікальних функцій (візуальний конструктор, AI-асистент), баланс простоти та потужності
5. За характером конкурентних переваг — нецінова / цінова	На етапі виходу на ринок конкуренція через унікальність UX, простоту, якість підтримки. При зростанні — також через ціну	Інвестиції в дослідження та розвиток для покращення продукту, створення найпростішого у використанні рішення на ринку, оптимізація витрат для конкурентного ціноутворення, використання безкоштовного доступу
6. За інтенсивністю — марочна	Важливість репутації бренду, відгуків користувачів, присутності в галузевих рейтингах	Активна робота над брендом, збір та публікація відгуків клієнтів, участь у галузевих виставках та конференціях, контент-маркетинг, демонстрація прикладів успішного використання

Далі було проведено аналіз конкуренції за моделлю М. Портера, що дозволяє виявити основні ринкові сили та визначити напрями реалізації стратегічних заходів. Результати відображено в таблиці 5.9.

Таблиця 5.9 — Аналіз конкуренції в галузі за М. Портером

	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Складові аналізу	ParseHub, Import.io, Scrapy Cloud, Apify, Mozenda, Diffbot	Великі tech-компанії (Google, Microsoft), українські IT-стартапи	Хмарні провайдери, провайдери проксі-серверів, сервіси розпізнавання CAPTCHA	Інтернет-магазини, продавці на маркетплейсах, роздрібна торгівля, сервіси порівняння цін, агрегатори товарів, стартапи, маркетингові агенції, аналітичні компанії	Ручний збір даних, віртуальні асистенти, розробка власних рішень для автоматизації збору даних
Висновки:	Високий рівень конкуренції на глобальному ринку через наявність великих конкурентів із значними бюджетами та клієнтською базою, однак майже немає конкурентів на українському ринку.	Середній ризик появи нових конкурентів. Бар'єри входження існують, але не є критичними. . Строк появи потенційних конкурентів: 6-12 місяців	Низька загроза з боку постачальникі в через наявність великої кількості альтернатив.	Клієнти мають значну силу через низькі витрати на перехід до конкурентів та велику кількість альтернатив. Ключові фактори: якість досвіду використання, технічна підтримка та швидкість інтеграції.	Середня загроза з боку замінників. Для простих завдань є дешевші альтернативи (наприклад, ручний збір), а для складних завдань — розробка власних рішень. Обмеження для роботи на ринку помірні

Отже, аналіз конкуренції за моделлю М. Портера показує, що глобальний ринок автоматизованого збору даних характеризується високим рівнем конкуренції через присутність великих міжнародних гравців та безкодових

рішень, таких як ParseHub та Octoparse. Проте на українському ринку конкуренція знаходиться на низькому рівні, що створює сприятливі умови для входження стартапу.

Далі необхідно визначити та обґрунтувати фактори конкурентоспроможності стартап-проєкту, що дозволить окреслити основні напрями підвищення його ефективності на ринку (див. табл. 5.10).

Таблиця 5.10 — Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Простота використання (безкодовий підхід)	Система має зрозумілий користувацький інтерфейс, що знижує час, необхідний для налаштування збору даних та отримання результату для користувачів без технічних знань.
2	Швидкість налаштування збору даних	Час налаштування збору даних становить 10-15 хвилин без попереднього навчання.
3	Якість та повнота зібраних даних	Забезпечується точність розпізнавання полів товарів, відсутність дублікатів, коректна обробка динамічного контенту вебсторінок.
4	Надійність та стабільність роботи	Система забезпечує безперебійність роботи сервісу, здатність обходити системи захисту від автоматизованого збору та стабільність виконання запланованих зборів даних.
5	Ціна та гнучкість тарифних планів	Наявність безкоштовного тарифного плану для тестування, прозоре ціноутворення без прихованих платежів, можливість оплати за фактичне використання.
6	Якість підтримки користувачів	Забезпечення україномовної підтримки, наявність докладної документації та навчальних матеріалів.
7	Різноманітність форматів експорту	Наявність можливості експорту даних в популярні формати (CSV, Excel, JSON, Google Sheets, SQL тощо) для подальшого використання для потреб клієнтів.

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
8	Автоматизація та періодичний збір	Можливість налаштування автоматичного розкладу для регулярного оновлення даних без ручного втручання, для забезпечення постійного моніторингу змін.

Далі було проаналізовано сильні та слабкі сторони стартап-проєкту у порівнянні з основними конкурентами за ключовими факторами конкурентоспроможності (див. табл. 5.11).

Таблиця 5.11 — Порівняльний аналіз сильних та слабких сторін «UniScraper»

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з … (назва підприємства)						
			–3	–2	–1	0	+1	+2	+3
1	Простота використання (UX/UI)	18				+			
2	Час налаштування першого збору	15				+			
3	Якість зібраних даних (точність)	17				+			
4	Ціна	20			+				
5	Підтримка динамічного вмісту сторінок	17					+		
6	Різноманітність форматів експорту даних	14				+			
7	Автоматизація періодичного збору даних	19				+			
8	Швидкість збору великих обсягів даних	13					+		
9	Гнучкість налаштування збору даних	16					+		
10	Можливість збору даних зі сторінок товарів	16				+			

Результати ринкового дослідження та оцінки внутрішніх ресурсів проєкту узагальнено у формі SWOT-аналізу (див. табл. 5.12), який визначає стратегічну позицію стартапу та окреслює ключові вектори його розвитку.

Таблиця 5.12 — SWOT-аналіз стартап-проєкту

Сильні сторони: • Найпростіший інтерфейс на ринку без потреби програмування • Український продукт з повною локалізацією та місцевою підтримкою • Гнучка модель підписки (безоплатна базова версія + оплата за використання) • Висока швидкість налаштування збору • Збір даних зі статичних та динамічних сторінок • Конкурентне ціноутворення для малого та середнього бізнесу • Фокус на недостатньо обслуговуваний ринок (Україна та Східна Європа) • Різноманітність форматів експорту даних • Автоматизація періодичного збору	Слабкі сторони: • Відсутність репутації та історії на ринку • Обмежений стартовий бюджет • Мала команда • Відсутність клієнтської бази • Обмежені можливості для використання масштабованої хмарної інфраструктури • Слабка присутність на міжнародних ринках • Залежність від зовнішніх сервісів (проксі, хмарні рішення) • Відсутність підтримки багатокористувацької роботи на початковому етапі • Обмежені можливості технічної підтримки через розмір команди
Можливості: • Швидке зростання електронної комерції в Україні • Збільшення попиту на автоматизацію • Потреба локалізованих рішень • Можливість спеціалізації за галузями • Розвиток штучного інтелекту для покращення розпізнавання даних • Партнерства з українськими платформами електронної комерції • Інтеграції з популярними інструментами • Створення каталогу шаблонів • Розробка мобільного застосунку • Залучення інвестицій	Загрози: • Висока конкуренція від міжнародних платформ • Посилення технологій захисту від автоматизованого збору даних • Правові ризики • Можливість появи серед конкурентів великих технологічних компаній • Ризик блокування служб проксі-серверів • Швидкі технологічні зміни структури вебсайтів • Залежність від кількох ключових клієнтів на початковому етапі • Зміни в законодавстві щодо захисту персональних даних • Технічні проблеми при масштабуванні

Наступним етапом створення стартап-проєкту є розробка та аналіз альтернативних варіантів ринкової поведінки з урахуванням ймовірності отримання необхідних ресурсів та строків реалізації. Результати аналізу можливих альтернатив ринкового впровадження представлено в таблиці 5.13.

Таблиця 5.13 — Альтернативи ринкового впровадження стартап-проєкту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Поступовий вихід через безоплатну версію	85%	6-9 місяців
2	Партнерська модель з платформами електронної комерції	60%	9-12 місяців
3	Галузева спеціалізація	80%	4-6 місяців
4	Корпоративна модель з великими клієнтами	40%	12-18 місяців
5	Модель "продукт дня" з швидким залученням інвестицій	55%	3-6 місяців

Розглянуті альтернативи ринкового впровадження проєкту UniScraper демонструють різні підходи до виходу на ринок залежно від доступних ресурсів, рівня ризику та бажаної швидкості розвитку.

Далі необхідно вибрати цільові групи потенційних споживачів продукту, що наведено в таблиці 5.14.

Таблиця 5.14 — Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Власники малих та середніх інтернет-магазинів	Висока	Високий	Низька	Висока
2	Маркетологи та аналітики у сфері торгівлі через інтернет	Висока	Високий	Середня	Середня

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
3	Фріланс-спеціалісти у сфері онлайн-торгівлі (збір даних, аналітика, налаштування реклами)	Висока	Середній	Середня	Середня
4	Маркетингові та аналітичні агенції	Середня	Середній	Середня	Середня
5	Великі рітейлери та маркетплейси	Низька	Низький	Висока	Низька
Які цільові групи обрано: власники малих та середніх інтернет-магазинів і аналітики у сфері торгівлі через інтернет					

Отже, виконаний аналіз підтвердив перспективність запуску стартап на українському ринку та дозволив обґрунтувати стратегічні напрямки підвищення ефективності і конкурентоспроможності проєкту.

5.4 Розроблення ринкової стратегії проєкту

Розроблення ринкової стратегії проєкту є критичним етапом планування, який передбачає послідовне формування узгодженої системи стратегічних рішень: від вибору цільових сегментів споживачів до визначення базової стратегії розвитку, конкурентної поведінки та позиціонування продукту на ринку. В таблиці 5.15 визначено базову стратегію розвитку стартап-проєкту.

Таблиця 5.15 — Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
1	Запуск платформи автоматизованого збору даних для українських інтернет-магазинів	Концентрований маркетинг з подальшим переходом до диференційованого	Простий інтерфейс, доступна ціна, локалізація, готові шаблони сайтів, швидке впровадження, якісна підтримка	Стратегія спеціалізації (фокус на ніші українського малого та середнього бізнесу електронної комерції)

В таблиці 5.16 було визначено базову стратегію конкурентної поведінки створюваного сатртап-проєкту, а в таблиці 5.17 — стратегію позиціонування.

Таблиця 5.16 — Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
1	Ні, на ринку існують систему збору даних та сервіси моніторингу, але немає локальних рішень, націлених на простоту та автоматизацію	Компанія планує переманювати існуючих користувачів, пропонуючи простіше, швидше та доступніше рішення, а також залучати нових клієнтів завдяки безкоштовному базовому пакету	Частково так: можуть бути відтворені основні функціональні можливості, проте інтерфейс, автоматизація та підтримка будуть суттєво вдосконалені	Стратегія наслідування лідера з покращенням функціональності та зайняттям конкурентної ніші

Таблиця 5.17 — Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1	Простота налаштування без програмування, доступна ціна, швидке впровадження, технічна підтримка українською мовою	Стратегія спеціалізації (фокус на ніші українського малого та середнього бізнесу електронної комерції)	Інтерфейс без потреби програмування, налаштування збору за 10-15 хвилин, гнучка модель підписки з безоплатною базовою версією, повна україномовна локалізація та підтримка	Простота, швидкість, доступність
2	Можливість експорту в різних форматах, інтеграція через програмний інтерфейс, обробка великих обсягів даних, збереження історії змін	Стратегія спеціалізації	Різноманітність форматів експорту, автоматизація періодичного збору без втручання користувача, структуроване зберігання з повною історією змін, гібридний підхід до збору статичних та динамічних даних	Автоматизація, надійність, професійність
3	Актуальність даних, висока точність збору, стабільність роботи, адаптація до українського ринку	Стратегія спеціалізації	Готові шаблони для популярних українських інтернет-магазинів, висока точність розпізнавання полів товарів, регулярне автоматичне оновлення інформації	Точність, актуальність, локальна адаптованість

За результатами розроблення ринкової стратегії стартап-проєкту було сформовано узгоджену систему стратегічних рішень, визначено пріоритетні цільові групи споживачів — власників малих та середніх інтернет-магазинів і аналітиків у сфері електронної комерції. Також було обрано стратегію спеціалізації з фокусом на українському ринку, стратегію наслідування лідера з покращенням функціональності та позиціонування на основі простоти, швидкості та доступності.

5.5 Розроблення маркетингової програми стартап-проєкту

Завершальним етапом підготовки до запуску стартапу є розробка маркетингової стратегії, яка визначатиме спосіб представлення продукту цільовій аудиторії.

Для формування маркетингової концепції товару необхідно визначити ключові вигоди, які отримає споживач, та конкурентні переваги системи. У таблиці 5.18 подано перелік основних потреб цільової аудиторії, вигоду від використання системи та ключових переваг порівняно з існуючими рішеннями на ринку.

Таблиця 5.18 — Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1	Автоматизація збору даних про товари конкурентів для моніторингу цін	Швидке отримання структурованої інформації про ціни та наявність товарів без ручної роботи	Найпростіший інтерфейс без потреби програмування, налаштування за 10-15 хвилин проти годин у конкурентів, готові шаблони для українських інтернет-магазинів
2	Збір даних для аналітики ринку та конкурентного аналізу	Отримання якісних даних для досліджень, аналітичних звітів і прогнозування ринку	Автоматичне збереження історії змін, експорт у різних форматах, підтримка великих обсягів даних, можливість інтеграції через програмний інтерфейс

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
3	Регулярний моніторинг наявності товарів та актуальних цін	Автоматичне оновлення інформації без втручання користувача	Гнучке налаштування періодичності збору, автоматизація без додаткової плати, надійність виконання запланованих завдань
4	Доступ до технології збору даних без технічних навичок	Можливість користуватися професійним інструментом без знання програмування	Візуальний конструктор шаблонів, інтуїтивний інтерфейс, повна україномовна локалізація, якісна технічна підтримка
5	Економія коштів на автоматизацію збору даних	Доступна ціна порівняно з конкурентами та економія на зарплаті фахівців	Безоплатна базова версія, гнучке ціноутворення за фактичне використання, відсутність прихованих платежів, найнижча ціна серед конкурентів

Наступним кроком є розроблення трирівневої маркетингової моделі товару, яка дозволяє деталізувати ідею продукту від базової концепції до конкретних характеристик та додаткових послуг. У таблиці 5.19 наведено опис системи на всіх трьох рівнях: задум товару, реальне виконання з технічними характеристиками та елементи підкріплення до і після продажу.

Таблиця 5.19 — Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
I. Товар за задумом	Платформа для автоматизованого збору та структурування інформації про товари з інтернет-магазинів, що задовольняє потребу бізнесу в отриманні актуальних даних про ціни, наявність та характеристики продукції конкурентів без необхідності технічних навичок та значних фінансових витрат

Рівні товару	Сутність та складові		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1. Швидкість налаштування збору даних	Нм	Тх
	2. Точність збору інформації	Нм	Е
	3. Автоматизація періодичного оновлення	Нм	Тл
	4. Робота зі статичними та динамічними сторінками	Нм	Тх
	5. Інтерфейс без потреби програмування	М	Ор
	6. Експорт даних у різних форматах	М	Тл
	7. Збереження історії всіх зборів	М	Вр
	Якість: систему розроблено відповідно до сучасних стандартів веброзробки, забезпечено точність збору та згідно із поставленими задачами		
	Відсутнє, оскільки продукт є цифровим		
	Марка: UniScraper — універсальна платформа для збору даних з інтернет-магазинів		
III. Товар із підкріпленням	До продажу: безоплатний пробний період, готові шаблони, документація українською мовою		
	Після продажу: технічна підтримка українською мовою, регулярні оновлення, допомога у налаштуванні		
За рахунок чого потенційний товар буде захищено від копіювання: за рахунок унікального поєднання простоти інтерфейсу, локалізації для українського ринку, накопиченої бази шаблонів та ліцензійної угоди користувача, яка забороняє копіювання, модифікацію та розповсюдження програмного коду платформи. Додатковий захист забезпечується через реєстрацію торговельної марки, авторське право на програмне забезпечення та угоду про нерозголошення конфіденційної інформації для корпоративних клієнтів			

Визначення цінових меж є критичним етапом для формування конкурентоспроможної цінової стратегії стартапу. У таблиці 5.20 проведено аналіз цін на товари-замінники та аналоги, рівня доходів цільової аудиторії, що дозволяє встановити оптимальний діапазон цін для різних сегментів споживачів.

Таблиця 5.20 — Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	15000-30000 грн/місяць (ручний збір даних)	0 – 37 000 грн/місяць	10 000 – 150 000 грн/місяць	0 - 15000 грн/місяць

Отже, середній рівень цін на замінники та аналоги суттєво перевищує запропонований ціновий діапазон, що дозволяє сформувати відчутну цінову перевагу. Діапазон доходів цільової аудиторії охоплює запропоновану вартість, роблячи продукт доступним. Установлення ціни на рівні до 15 000 грн забезпечує конкурентність на старті. Такий підхід відповідає стратегії зниження ціни на 40–60% для швидкого входу на ринок.

Наступним після визначення ціни етапом є формування ефективної системи збуту. У таблиці 5.21 подано особливості поведінки цільових клієнтів і визначено оптимальні канали та інструменти продажу, що забезпечують зручний доступ до продукту та підтримку під час його впровадження.

Таблиця 5.21 — Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Малий та середній бізнес шукає рішення онлайн, порівнює пропозиції, читає відгуки, тестує безоплатні версії перед прийняттям рішення про купівлю, цінує швидкість впровадження та простоту використання	Доступ через вебінтерфейс, безоплатний пробний період, техпідтримка, допомога у налаштуванні, інтеграція з українськими платіжними системами	Канал нульового рівня (прямий продаж без посередників)	Прямий онлайн-збут через вебплатформу: реєстрація та доступ через сайт, автоматичний доступ до безоплатної версії, система онлайн оплати з підтримкою карток та електронних гаманців, чат підтримка та система заявок

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
2	Маркетингові агенції та професійні аналітики шукають рішення через професійні спільноти, потребують детальної технічної документації, цінують можливість інтеграції з існуючими інструментами, приймають рішення на основі технічних характеристик	Надання програмного інтерфейсу для інтеграції, детальна технічна документація, консультації щодо впровадження, допомога у налаштуванні складних сценаріїв збору, індивідуальні умови для корпоративних клієнтів	Канал нульового рівня з елементами технічної підтримки	Пряма модель збуту з посиленою технічною підтримкою: особистий менеджер для корпоративних клієнтів, технічні консультації, допомога у інтеграції, гнучкі умови оплати для довгострокових контрактів

Завершальним етапом є розробка маркетингових комунікацій (таблиця 5.22), що враховують поведінку цільових клієнтів і канали їх взаємодії.

Таблиця 5.22 — Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Малий та середній бізнес активно використовує соціальні мережі для пошуку рішень, читає тематичні блоги про електронну комерцію, шукає відгуки користувачів	Соціальні мережі, тематичні блоги про електронну комерцію, форуми підприємців, YouTube-канали про автоматизацію бізнесу, месенджери для спілкування у бізнес-спільнотах	Простота налаштування без програмування, швидкість впровадження за 15 хвилин, безкоштовний старт	Сформувати впізнаваність бренду як найпростішого рішення для збору даних, продемонструвати легкість використання, знизити бар'єр входу через безкоштовну версію	«Почніть збирати дані про конкурентів за 15 хвилин. Без програмування, без складних налаштувань. Перший місяць безкоштовно. UniScraper — автоматизація для вашого бізнесу»
2	Маркетингові агенції та аналітики користуються професійними платформами, технічними оглядами та галузевими конференціями, цінують детальні технічні характеристики.	Професійні платформи для технічних фахівців, галузеві конференції з електронної комерції, спеціалізовані вебінари, технічні блоги, професійні спільноти аналітиків	Якість та повнота зібраних даних, автоматизація без втручання, збереження історії змін	Позиціонувати систему як професійний інструмент для аналітики, підкреслити технічні переваги, продемонструвати можливості інтеграції	«Професійна аналітика потребує якісних даних. UniScraper забезпечує точність понад 95%, автоматичне оновлення та повну історію змін. Для тих, хто приймає рішення на основі фактів»

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
3	Стартапи та дослідники ринку шукають огляди та порівняння інструментів в мережі, цінують співвідношення ціни та якості, потребують швидкого доступу до даних	Пошукові системи, платформи для стартапів, спільноти підприємців, тематичні телеграм-канали про технології та бізнес	Доступність для обмежених бюджетів, швидкість отримання результатів, локальна адаптованість	Залучити увагу молодих компаній, підкреслити вигідність цінової політики, показати швидкість досягнення результату	«Дослідіть ринок швидше за конкурентів. UniScraper дає дані про тисячі товарів без великих витрат. Український сервіс для українського бізнесу. Розпочніть сьогодні безкоштовно»

За підсумками розробки маркетингової програми стартапу сформовано комплекс стратегічних рішень щодо позиціонування продукту, визначено основні переваги для цільової аудиторії та обрано ефективні канали збуту й комунікацій для просування на ринку.

Висновки до розділу 5

У цьому розділі було розроблено та описано концепцію стартап-проєкту, визначено його призначення та проаналізовано технічні й ринкові умови запуску. Проведене порівняння з конкурентами дозволило виокремити ключові сильні сторони проєкту — простоту використання, локальну адаптованість та конкурентну вартість послуг. Технологічний аудит підтвердив наявність і

доступність усіх необхідних інструментів для реалізації системи автоматизованого збору даних.

Аналіз ринкового середовища та SWOT-структура засвідчили перспективність запуску проєкту на українському ринку, де рівень конкуренції є відносно низьким, а попит на автоматизацію навпаки — високим. На основі цих результатів було визначено цільові сегменти користувачів, сформовано стратегію спеціалізації та обрано позиціонування, орієнтоване на простоту, швидкість та доступність.

Розроблена маркетингова програма встановила ключові переваги продукту, оптимальний діапазон цін та канали збуту. Загалом проведений аналіз підтвердив, що проєкт має реальні перспективи виходу на ринок і здатний забезпечити конкурентні позиції за умови поетапного вдосконалення та активної ринкової діяльності.

ВИСНОВКИ

В цій роботі було розроблено інформаційну систему автоматизованого збору даних про товари в інтернет-магазинах, яка поєднує сучасні методи вебскрепінгу, актуальні технології обробки даних та зручні засоби взаємодії користувача з програмною інфраструктурою. У процесі дослідження було проведено детальний аналіз предметної області, визначено ключові вимоги до системи та обґрунтовано вибір архітектурних підходів, мов програмування, бібліотек і технологій, що забезпечують надійність та ефективність розробленого програмного рішення.

Проаналізовано різні методи отримання даних, включаючи використання публічних API, збір даних зі статичного HTML, обробку динамічних сторінок за допомогою браузерів без графічного інтерфейсу, а також гібридний підхід, що дозволило визначити оптимальне поєднання інструментів для реалізації поставлених задач. Для програмної реалізації серверної частини системи було обрано мову програмування Python, а також відповідні бібліотеки, такі як aiohttp, Selenium та BeautifulSoup4, для створення підсистеми збору даних, що забезпечило необхідну продуктивність та гнучкість під час розробки та роботи створеного проєкту.

Створена архітектура системи забезпечує чітку взаємодію між клієнтською та серверною частинами, підтримуючи стабільний збір, обробку та збереження даних у форматах JSON та CSV з використанням SQLite і Redis. У роботі було розроблено інформаційну модель, механізм виконання процесу збору даних та модулі, що дозволяють здійснювати планування, запуск, моніторинг і оцінку часу виконання відповідних операцій.

Для реалізації модуля оцінки часу виконання процесу збору даних у рамках цієї роботи була розроблена відповідна математична модель, яка поєднує обчислення часу скрапінгу та парсингу даних. Хоча теоретичні та фактичні результати дещо відрізняються, ця різниця є незначною. Наявність похибки обумовлена значною кількістю непередбачуваних факторів, які впливають на

фактичний час збору даних, таких як затримки на стороні серверів, швидкість інтернет-з’єднання, обсяг та складність оброблюваної інформації, а також обмеження частоти запитів.

В рамках системи було також введено поняття «шаблон», що слугує для опису набору інструкцій до збору даних з певного інтернет-магазину. Також реалізовано інтерфейси, які забезпечують створення, редагування та перегляд шаблонів збору даних, а також перегляд та можливість експорту результатів зібраної інформації, що робить систему зручною для користувача.

У рамках даної роботи було також описано концепцію стартап-проєкту, яка включає опис ідеї, аналіз технологічних можливостей її реалізації, оцінку ринкового потенціалу та формування стратегії розвитку і маркетингової програми. Це підтвердило перспективність впровадження створеної інформаційної системи на ринку інструментів для автоматизованого збору даних та аналітики у сфері електронної комерції.

Загалом результати роботи демонструють, що розроблена система є ефективним, гнучким і перспективним рішенням для автоматизації процесів збору та подальшого аналізу даних про товари з інтернет-магазинів. Вона може бути розширена, адаптована до нових джерел інформації та інтегрована з аналітичними або бізнес-платформами, що визначає її значний потенціал для подальшого розвитку та комерційного застосування.

Посилання на програний код створеної в даній роботі системі можна знайти за посиланням, який знаходиться в додатку А.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Big data statistics: How much data is there in the world?. URL: https://rivery.io/blog/big-data-statistics-how-much-data-is-there-in-the-world/ (дата звернення: 28.11.2025).

2. Selenium-webdriver. URL: https://www.npmjs.com/package/selenium-webdriver (дата звернення: 28.11.2025).

3. Playwright (software). URL: https://en.wikipedia.org/wiki/Playwright_(software) (дата звернення: 28.11.2025).

4. Tutorial: How To Use a Headless Browser for Web Scraping. URL: https://www.nimbleway.com/blog/headless-browser-scraping-guide (дата звернення: 28.11.2025).

5. A free web scraper that is easy to use. URL: https://www.parsehub.com/ (дата звернення: 28.11.2025).

6. Scalable cloud hosting for your Scrapy spiders. URL: https://www.zyte.com/scrapy-cloud/ (дата звернення: 28.11.2025).

7. AI - Data Extraction. URL: https://www.import.io/ (дата звернення: 28.11.2025).

8. Client-server architecture. URL: https://www.britannica.com/technology/client-server-architecture (дата звернення: 28.11.2025).

9. Understanding the Client-Server Model. URL: https://systemdesignschool.io/blog/client-server (дата звернення: 28.11.2025).

10. Microservices vs. monolithic architecture. URL: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith (дата звернення: 28.11.2025).

11. What is Microservices Architecture?. URL: https://cloud.google.com/learn/what-is-microservices-architecture (дата звернення: 28.11.2025).

12. Python (programming language). URL: https://en.wikipedia.org/wiki/Python_(programming_language) (дата звернення: 28.11.2025).

13. Applications for Python. URL: https://www.python.org/about/apps/ (дата звернення: 28.11.2025).

14. TypeScript. URL: https://en.wikipedia.org/wiki/TypeScript (дата звернення: 28.11.2025).

15. TypeScript: An Introduction to the Statically Typed Superset of JavaScript. URL: https://dev.to/matheusgomes062/typescript-an-introduction-to-the-statically-typed-superset-of-javascript-4hgo (дата звернення: 28.11.2025).

16. TypeScript. URL: https://reintech.io/terms/tech/typescript-superset-javascript-static-typing (дата звернення: 28.11.2025).

17. Programming language TypeScript: definition, advantages, and disadvantages. URL: https://www.mytaskpanel.com/programming-language-typescript/ (дата звернення: 28.11.2025).

18. Benefits of TypeScript: Why You Should Choose It?. URL: https://www.codewalnut.com/learn/benefits-of-typescript (дата звернення: 28.11.2025).

19. Advantages and Disadvantages of TypeScript over JavaScript. URL: https://www.geeksforgeeks.org/javascript/advantages-and-disadvantages-of-typescript-over-javascript/ (дата звернення: 28.11.2025).

20. TypeScript, What Good For?. URL: https://medium.com/@chaewonkong/typescript-what-good-for-8240dc9173c7 (дата звернення: 28.11.2025).

21. What are disadvantages of TypeScript?. URL: https://fatcatremote.com/it-glossary/typescript/disadvantages-of-typescript (дата звернення: 28.11.2025).

22. The Good and the Bad of TypeScript. URL: https://www.altexsoft.com/blog/typescript-pros-and-cons/ (дата звернення: 28.11.2025).

23. Корнієнко Б.Я., Галата Л.П. Дослідження імітаційного полігону захисту критичних інформаційних ресурсів методом IRISK // *Моделювання та інформаційні технології*. 2018. Вип. 83. С. 34-42.

24. Korniyenko B., Galata L. Implementation of the information resources protection based on the CentOS operating system // Conference Proceedings of 2019 IEEE 98 2nd Ukraine Conference on Electrical and Computer Engineering (UKRCON - 2019) July 2 – 6, 2019, Lviv, Ukraine. P. 1007-1011.

25. Korniyenko B., Yudin A., Galata L. Risk estimation of information system. // *Wschodnioeuropejskie Czasopismo Naukowe*. 2016. № 5. P. 35-40.

26. Корнієнко Б.Я. Побудова та тестування імітаційного полігону захисту критичних інформаційних ресурсів // *Наукоємні технології*. 2017. № 4 (36). С. 316 - 322.

27. Корнієнко Б.Я., Юдін О.К., Снігур О.С. Безпека аутентифікації у web-ресурсах // *Захист інформації*. 2012. № 1 (54). С. 20 -25. DOI: 10.18372/2410-7840.14.2056 (ukr).

28. Корнієнко Б.Я. Безпека інформаційно-комунікаційних систем та мереж // Навчальний посібник для студентів спеціальності 125 «Кібербезпека». – К.: НАУ, 2018. 226 с.

29. Корнієнко Б.Я., Максімов Ю.О., Марутовська Н.М. Прикладні програми управління інформаційними ризиками // *Захист інформації*. 2012. № 4 (57). С. 60 – 64. DOI: 10.18372/2410-7840.14.3493 (ukr).

30. Корнієнко Б.Я., Галата Л.П. Оптимізація системи захисту інформації корпоративної мережі // *Математичне та комп'ютерне моделювання. Серія: Технічні науки*. 2019. Випуск 19. С. 56-62.

31. MVC Framework Introduction - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/mvc-framework-introduction/ (дата звернення: 28.11.2025).

32. Django Architecture - Detailed Explanation. InterviewBit. URL: https://www.interviewbit.com/blog/django-architecture/ (дата звернення: 28.11.2025).

33. Корнієнко Б. Я. Дослідження моделі взаємодії відкритих систем з погляду інформаційної безпеки // Наукоємні технології. 2012. № 3 (15). С. 83–89, doi.org/10.18372/2310- 5461.15.5120 (ukr).

34. DJango Rest Framework. URL: https://medium.com/@david.christianto05/django-rest-framework-6466ffb09b78 (дата звернення: 28.11.2025).

35. Tyagi A. Learn React – A Guide to the Key Concepts. freeCodeCamp.org. URL: https://www.freecodecamp.org/news/learn-react-key-concepts/#key-concepts-to understand-in-react (дата звернення: 28.11.2025).

36. React JSX. URL: https://www.geeksforgeeks.org/reactjs/reactjs-jsx-introduction/ (дата звернення: 28.11.2025).

37. Aiohttp. URL: https://webscraping.fyi/lib/python/aiohttp/ (дата звернення: 28.11.2025).

38. What Is Selenium Python?. URL: https://www.coursera.org/articles/what-is-selenium-python (дата звернення: 28.11.2025).

39. BeautifulSoup4 Module - Python. URL: https://www.geeksforgeeks.org/python/beautifulsoup4-module-python/ (дата звернення: 28.11.2025).

40. Welcome to Django Q. URL: https://django-q.readthedocs.io/en/latest/ (дата звернення: 28.11.2025).

41. Task Wars: DjangoQ vs Celery, Redis vs RabbitMQ, and the Battle of Cloud vs Self-Hosted. URL: https://medium.com/@s.m.shahinul.islam/task-wars-djangoq-vs-celery-redis-vs-rabbitmq-and-the-battle-of-cloud-vs-self-hosted-bc5b14e577cf (дата звернення: 28.11.2025).

42. Asynchronous tasks in Django with Django Q. URL: https://www.valentinog.com/blog/django-q/ (дата звернення: 28.11.2025).

43. About SQLite. URL: https://sqlite.org/draft/about.html (дата звернення: 28.11.2025).

44. Галата Л.П., Корнієнко Б.Я., Заболотний В.В. Математична модель протидії загрозам у системі захисту критичних інформаційних ресурсів // *Наукоємні технології*. 2019. Том 43. № 3. С. 300 – 306.

45. Korniyenko B.Y., Galata L.P. Design and research of mathematical model for information security system in computer network // *Наукоємні технології*. 2017. № 2 (34). С. 114-118.

46. Korniyenko, B. Modeling of information security system in computer network // *Безпека інформаційних систем і технологій*. 2019. Том №1 (1). С.36-41.

47. Korniyenko B., Galata L., Ladieva L. Research of Information Protection System of Corporate Network Based on GNS3 // Conference Proceedings of 2019 IEEE Inernational Conference on Advanced Trends in Information Theory (IEEE ATIT - 2019) Desember 18 – 20, 2019, Kyiv, Ukraine. P. 244-248.

48. Korniyenko B., Galata L., Kozuberda O. Modeling of security and risk assessment in information and communication system // *Sciences of Europe*. 2016. V. 2. No 2 (2). P. 61 -63.

49. What is Redis? | IBM. IBM in Deutschland, Österreich und der Schweiz. URL: https://www.ibm.com/topics/redis (дата звернення: 25.05.2024).

50. Introducing JSON. URL: https://www.json.org/json-en.html (дата звернення: 28.11.2025).

51. What Is Asynchronous Programming? (And When To Use It). URL: https://www.indeed.com/career-advice/career-development/asynchronous-programming (дата звернення: 28.11.2025).