ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ІНТЕЛЕКТУАЛЬНА СИСТЕМА МОНІТОРИНГУ ТОВАРІВ НА E-COMMERCE ПЛАТФОРМАХ З ВИКОРИСТАННЯМ WEB-СКРЕПІНГУ ТА ВІЗУАЛІЗАЦІЇ»

на здобуття освітнього ступеня магістр
за спеціальності 124 Системний аналіз
(код,найменування спеціальності)
освітньо-професійної програми Інтелектуальні системи управління
(назва)

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

_____ Роман ТОЛМАЧЕВ
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти
група САДМ-61 Роман ТОЛМАЧЕВ
(ім'я, ПРІЗВИЩЕ)
Керівник Ігор Патракеєв
доцент *(ім'я, ПРІЗВИЩЕ)*

Рецензент: _____
(ім'я, ПРІЗВИЩЕ)

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти магістр
Спеціальність 124 Системний аналіз
Освітньо-професійна прорама Інтелектуальні системи управління

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ
_____ Каміла СТОРЧАК
"____" _____2025 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Толмачева Романа Вікторовича
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Інтелектуальна система моніторингу товарів на e-commerce платформах з використанням web-скрепінгу та візуалізації
керівник кваліфікаційної роботи: Ігор ПАТРАКЕЄВ, доцент
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від "_25_" жовтня 2025 р. № _467___

2. Строк подання кваліфікаційної роботи «26» грудня 2025 р.

3. Вихідні дані кваліфікаційної роботи:
   1. Завдання на кваліфікаційну роботу студента.
   2. Наукові статті.
   3. Науково-технічна література.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
   1. Аналіз та огляд предметної області.
   2. Постановка задачі та методологія дослідження.
   3. Аналіз результатів впровадження системи.

5.Перелік ілюстративного матеріалу: презентація на слайдах

6. Дата видачі завдання «_30_» __жовтня__ 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи	30.10.2025	Викон.
2.	Написання 1 розділу кваліфікаційної роботи	10.11.2025	Викон.
3.	Написання 2 розділу кваліфікаційної роботи	20.11.2025	Викон.
4.	Написання 3 розділу кваліфікаційної роботи	28.11.2025	Викон.
5.	Висновки по роботі, вступ, реферат	29.11.2025	Викон.
6.	Розробка демонстраційних матеріалів, доповідь.	01.12.2025	Викон.
7.	Оформлення магістерської роботи	11.12.2025	Викон.

Здобувач вищої освіти _____ Роман ТОЛМАЧЕВ
(підпис) *(ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Ігор Патракеєв
(підпис) *(ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття ступня магістр: 73 стор., 33 рис., 7 табл., 28 джерел.

Мета роботи – розробка автоматизованої системи парсингу даних з веб-ресурсів.

Об'єкт дослідження – розробка моделі парсеру для аналізу веб-сторінок, пошуку і збереження не-обхідних даних..

Предмет дослідження – вивчення можливостей інструментів, аналіз існуючих методів та алгоритмів для створення моделі парсеру.

Короткий зміст роботи. У першому розділі магістерської роботи виконано аналіз особливостей використання існуючих методів для реалізації парсеру та застосування цих методів для розробки моделі. Проаналізовано функціонування з іншими технологіями, у різних галузях застосування. Розглянуто тенденції розвитку, які відображають не лише поточний стан ринку, але й дають уявлення про майбутнє зростання та інновації.

Виконано огляд досліджень у сфері систем збору і агрегації даних з Інтернету та, зокрема, в контексті e-commerce платформ. Проаналізовано різні методи, які використовуються для досягнення основних результатів

КЛЮЧОВІ СЛОВА: ІНТЕРНЕТ РЕЧЕЙ, ВИЯВЛЕННЯ АНАОМАЛІЙ, МОДЕЛЬ, МЕТОТ, АЛГОРИМ, МАШИННЕ НАВЧАННЯ, КЛАСТЕРИЗАЦІЯ, РЕРЕВО РІШЕНЬ, МІЖМАШИННА ВЗАЄМОДІЯ.

ABSTRACT

The text part of the qualifying work for obtaining a bachelor's degree: 73 pp., xx fig., 7 tables, 28 sources.

The purpose of the work is to study methods for detecting anomalies in IoT systems using machine learning algorithms.

The object of the study is the process of detecting anomalies in IoT systems.

The subject of the study is anomalies in IoT systems.

Summary of the work. The first section of the master's thesis analyzes the features of the use of IoT in various industries. The functioning with other technologies is analyzed. Development trends are considered, which reflect not only the current state of the market, but also give an idea of future growth and innovation.

A review of research in the field of intrusion detection systems in the context of the Internet and, in particular, in the context of the Internet of Things is performed. Various methods used to achieve the main results are analyzed.

The third section describes the methodology, the way in which the anomaly detection model in IoT uses a combination of two algorithms. The results of running the proposed model for anomaly detection in IoT are presented.

KEYWORDS: INTERNET OF THINGS, ANOMALY DETECTION, MODEL, METHOD, ALGORITHM, MACHINE LEARNING, CLUSTERIZATION, SOLUTION POOL, INTER-MACHINE INTERACTION.

ЗМІСТ

ВСТУП

Стрімкий розвиток електронної комерції є однією з ключових тенденцій сучасної цифрової економіки. Онлайн-торгівля поступово трансформується з допоміжного каналу збуту у домінуючий формат взаємодії між продавцями та споживачами. В умовах високої конкуренції на e-commerce платформах ціна, наявність товару, репутація продавця та динаміка ринкових змін стають критично важливими факторами прийняття управлінських рішень як для бізнесу, так і для кінцевих споживачів.

Сучасні інтернет-магазини та маркетплейси характеризуються високою динамічністю даних: ціни можуть змінюватися кілька разів на добу, асортимент постійно оновлюється, а відгуки користувачів формують репутаційний фон товарів і брендів у реальному часі. У таких умовах ручний моніторинг ринкової інформації є неефективним і практично неможливим, що зумовлює необхідність використання автоматизованих інтелектуальних систем збору та аналізу даних.

Одним із найбільш ефективних підходів до отримання актуальної інформації з відкритих веб-ресурсів є застосування технологій web-скрепінгу, які дозволяють автоматично збирати структуровані та напівструктуровані дані з веб-сторінок. Поєднання web-скрепінгу з методами інтелектуального аналізу даних, зокрема обробки часових рядів та аналізу тональності текстових відгуків, відкриває можливості для побудови комплексних систем моніторингу ринкових тенденцій у сфері e-commerce.

Незважаючи на наявність комерційних сервісів моніторингу цін і товарів, більшість із них або орієнтовані на великі корпоративні клієнти та мають високу вартість, або надають обмежений функціонал без глибокої аналітики. Крім того, такі рішення часто є закритими, що ускладнює їх адаптацію під специфічні вимоги малого та середнього бізнесу. У зв’язку з цим актуальним є завдання розробки власної інтелектуальної системи моніторингу, яка поєднує

автоматизований збір даних, аналітичну обробку та зручну візуалізацію результатів.

Тому на сьогодні є актуальність розробки інтелектуальної системи моніторингу товарів, цін та ринкових тенденцій на e-commerce платформах з використанням технологій web-скрепінгу, аналітичної обробки даних та елементів штучного інтелекту. Для досягнення поставленої мети у роботі необхідно вирішити низку завдань, пов’язаних з аналізом предметної області, вибором архітектурних рішень, реалізацією програмних модулів збору й обробки даних, а також оцінкою ефективності розробленої системи.

Об’єктом дослідження є процеси автоматизованого збору та аналізу інформації з e-commerce платформ.

Предметом дослідження є методи, моделі та програмні засоби побудови інтелектуальних систем моніторингу з використанням web-скрепінгу, аналітичних алгоритмів та мікросервісної архітектури.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ МОНІТОРИНГУ ДАНИХ У СФЕРІ E-COMMERCE

В наш час цифровізація економічного простору призвела до того, що електронна комерція трансформувалась з допоміжного інструменту торгівлі до фундаментальної складової економічної системи. Сучасні умови торгівельних процесів висувають нові вимоги до технічного забезпечення торгівлі, що призвело до розповсюдження та впровадження спеціалізованих e-commerce платформ. Моніторинг платформ e-commerce на ранніх етапах розвитку інтернет-торгівлі обмежувався перевіркою технічної доступності веб-ресурсів, але на сьогодні доцільно впроваджувати моніторинг зміни цін, тенденції популярності груп товарів, контроль якості обслуговування та забезпечення відповідності нормам та правовим вимогам.

1.1. Сутність, структура та функціональні особливості e-commerce платформ як об’єкта моніторингу

Сучасний е-бізнес це використання якісних сучасних технологій, для досягнення конкурентної переваги завдяки поліпшенню обслуговування своїх клієнтів та оптимізації бізнес-стосунків з партнерами. Використання сучасних інтернет-технології є однією з основ для досягнення переваг над конкурентами, але не єдиним ключовим аспектом в е-бізнесі.

Електронний бізнес це вид економічної діяльності суб’єктів господарювання через комп’ютерні мережі (тобто використання інтернету), з метою отримання прибутку. Тобто це електронна економічна діяльність, яка здійснюється за допомогою інформаційно-комунікаційних технологій (ІКТ) з метою отримання прибутків.

Е-бізнес це більш ніж просто продаж товарів або придбання товарів електронна покупка або продаж товарів. Е-бізнес потребує використання

сучасних Інтернет-технологій для проведення дій з метою отримання прибутків всередині та поза підприємством. Процес створення е-бізнесу наведено на рисунку 1.1.

Рисунок 1.1 – Процес створення е-бізнесу

В сучасності технології е-бізнесу є один із важливих інструментів сучасної конкурентної боротьби. Електронний бізнес змінив всі форми діяльності від малих до великих підприємств, від розробки продукту до продажу товарів. На рисунку 1.2. приведені технології на яких будується е-бізнес.

Е-бізнес ґрунтується на таких технологіях:
• мережні технології; • корпоративні; • Інтернет-технології; • виробничі IT; • система підтримки ухвалення рішень.

Рисунок 1.2 – Технології е-бізнесу

Головною перевагою на ринку є втілення організаційних структур дослідження та ринкові корпорації які створюють нові IT напрямки для дослідження ринку та його стану на поточний момент та на прогнозування на майбутнє.

Е-комерція є складовою е-бізнес, оскільки є одним зі способів його здійснення. Електронна комерція (е-комерція) це вид електронної комерційної діяльності з використанням інформаційно комунікаційних систем (тобто

використання інтернету). На рисунку 1.3 приведено основні напрямки на які орієнтується е-комерція.

Рисунку 1.3 – Основні напрямки орієнтація е-комерції

Е-комерція – це широкий комплекс методів ведення діяльності з наданням споживачам товарів або послуг. Тобто під е-комерцією розуміють будь-яку форму ділових операцій, де сторони взаємодіють між собою віддалено за допомогою електронних технологій без фізичного обміну або контакту.

Якщо узагальнюючи, е-комерція це застосування цифрових технологій для побудови та розвитку економічних відносин як між самими організаціями, так і між бізнесом та споживачем. Фактично, це ведення бізнесу в онлайн-режимі, що охоплює чотири основні сфери. На рисунку 1.4 приведено основні сфери комерції в онлайн режимі.

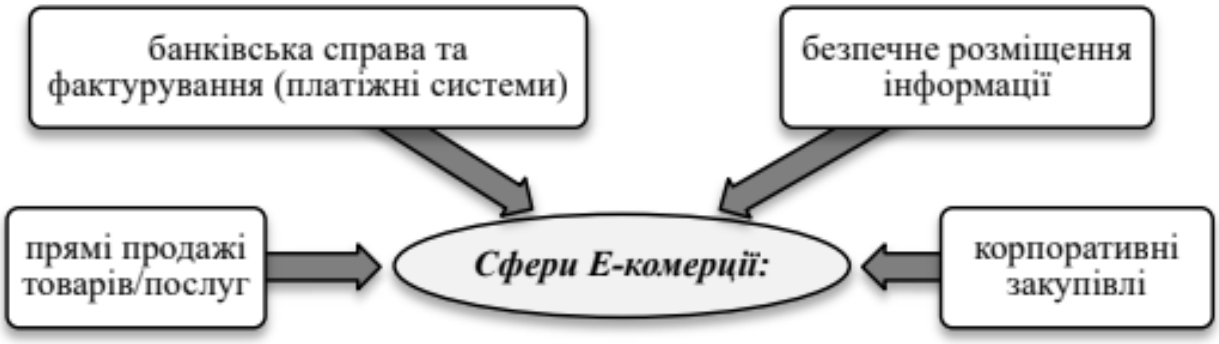


Рисунок 1.4 – Основні сфери комерції в онлайн режимі

Важливим аспектом класифікації об'єктів моніторингу є модель взаємодії учасників ринку, оскільки залежно від того, хто виступає продавцем та покупцем, змінюється специфіка роботи системи моніторингу.

Виділяють наступні модулі в е-комерції в сучасності, які наведені на рисунку 1.5.

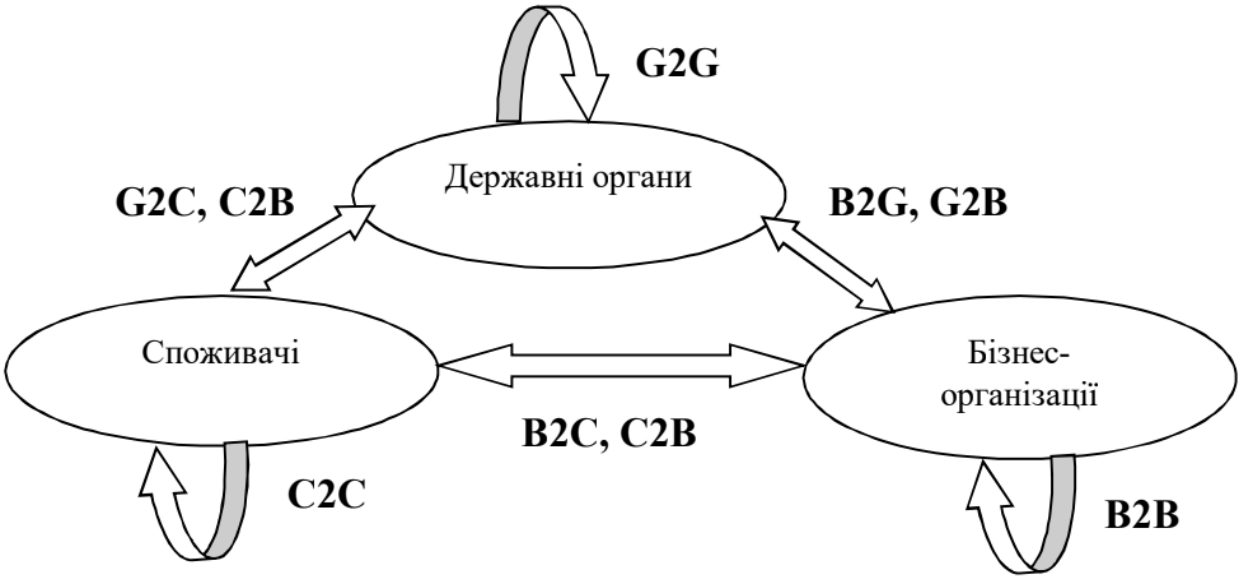


Рисунок 1.5 – Моделі е-комерції

Відповідно до рисунка 1.5 існують наступні моделі комерції:

– B2B – (Business-to-Business, бізнес для бізнесу) модель взаємодії між юридичними особами та організаціями. Тобто, охоплюються всі електронні транзакції товарів або послуг між двома підприємствами, а не між підприємством і споживачем. Як приклад це взаємодія між виробником та оптовим посередником;

– B2C – (Business-to-Consumer, бізнес для споживача) модель взаємодії між юридичними та фізичними особами. Дана модель відноситься більше до створення ділових відносин між підприємством та кінцевим споживачем. Тобто це роздрібна торгівля, яка в останній час набрала великих обертів через створення великої кількості онлайн магазинів;

– B2G – (Business-to-Government, бізнес для уряду) модель взаємодії між юридичними особами та державними організаціями. До даної категорії відносяться фіскальні послуги, державні закупівлі, соціальне забезпечення;

– C2C – (Consumer-to-Consumer, споживач для споживача) – модель взаємодії між фізичними особами. Як правило через третю особу, третьою стороною може виступати онлайн платформа. Прикладом може бути онлайн

аукціони, де людина пропонує свій товар, а інші користувачі встановлюють ціну, яку вони готові віддати за нього, наприклад ebay, olx. Однорангові системи (протокол для обміну файлами між користувачами) та обмін грошей, а також класифіковані оголошення на сайтах;

– G2C (Governmentto-Consumer, уряд для споживача) – модель взаємодії між державними організаціями та фізичними особами. Як приклад пенсійний фонд, взаємодія держави з громадянами;

– C2G (Consumer-to-Government, споживач для уряду) – модель взаємодії фізичної особи з державними органами. Це може стосуватись соціальних виплат, дистанційне навчання, податкові виплати або декларації.

Моделі відносин між учасниками процесу електронної комерції виглядають наступним чином.

B2B – (бізнес – бізнесу) охоплює:

– торгово-закупівельні майданчики;

– електронні вітрини та каталоги;

– електронні торгові ряди;

– електронні магазини;

– електронні біржі;

– електронні аукціони;

– галузеві торгові майданчики;

– системи повного циклу супроводу постачальників (SCM);

– системи управління розподілом;

– системи повного циклу супроводу клієнтів (CRM);

– аутсорсинг;

– електронні платіжні системи;

– віртуальні підприємства;

– системи інтернет-трейдінгу;

– інтернет-інкубатори;

– інтернет-реклама;

– системи мобільної комерції;

– системи страхування і перестрахування.

B2C – (бізнес – споживачам) охоплює:

– торгові ряди;

– електронні вітрини і каталоги;

– електронні магазини;

– електронні аукціони;

– інтернет-трейдінг;

– електронні платіжні системи;

– інтернет-страхування;

– системи телероботи;

– інтернет-реклама;

– спонсорські програми;

– дистанційна освіта;

– інтерактивне телебачення;

– електронні ЗМІ;

– туристичні послуги.

B2G (бізнес уряду) охоплює:

– участь в електронних торгах з закупівлі продукції для державних потреб;

– виконання державних замовлень;

– надання податкової, статистичної, митної та іншої звітності.

C2B (споживачі бізнесу) охоплює:

– приватні послуги;

– участь в опитуваннях та інших рекламних акціях;

– участь у партнерських і спонсорських програмах.

C2C(споживачі споживачам) охоплює:

– дошки оголошень;

– інтернет-аукціони;

– системи B2B;

– системи вірусного маркетингу.

C2G (споживачі владі) охоплює:

– участь у виборах;

– сплата податків, зборів, штрафів;

– участь в опитуваннях громадської думки;

– надання заявок, скарг, звернень громадян.

G2B (влада бізнесу) охоплює:

– системи розподілу державних замовлень;

– забезпечення контакту з податковими, митними органами, органами державної сертифікації та ліцензування, адміністраціями тощо;

– юридичні та інформаційно-довідкові служби, зокрема геоінформаційні системи.

G2C (влада споживачам) охоплює:

– системи соціального обслуговування (виплати, допомоги, пільги тощо);

– системи комунального обслуговування;

– юридичні та інформаційно-довідкові служби.

G2G (влада владі) охоплює:

– автоматизовані системи співпраці з митницею, податковою, правоохоронною сферами тощо;

– інформаційно-довідкові служби.

E-commerce в Україні має власний розвиток, який відрізняється від європейської та американської моделей. На вітчизняному ринку лідерами стали локальні маркетплейси та омніканальні рітейлери, які адаптувались до умов сучасності та складної логістики.

Компанія Kantar Ukraine та асоціації рітейлерів, робили дослідження ринку України. За їх даними беззаперечним лідером з охоплення аудиторії є маркетплейс Rozetka. За період 2023-2024 роки, до п'ятірки найпопулярніших e-commerce платформ також входять OLX, Prom.ua, AliExpress та Epicentrk.ua.

Дані від Kantar CMeter за 2023–2024 рр. рейтинг найбільш відвідуваних e-commerce в Україні є наступні сайти:

1. Rozetka.com.ua;
2. OLX.ua;

3. Prom.ua;
4. AliExpress.com;
5. Epicentrk.ua;
6. Comfy.ua;
7. Allo.ua;
8. Ria.com (включаючи Auto.ria);
9. Makeup.com.ua;
10. Tabletki.ua (що свідчить про стрімкий ріст сегмента e-pharma).

На рисунку 1.6 наведена діаграма частки відвідувань за платформами відповідно даним Kantar CMeter за 2023–2024 рр..

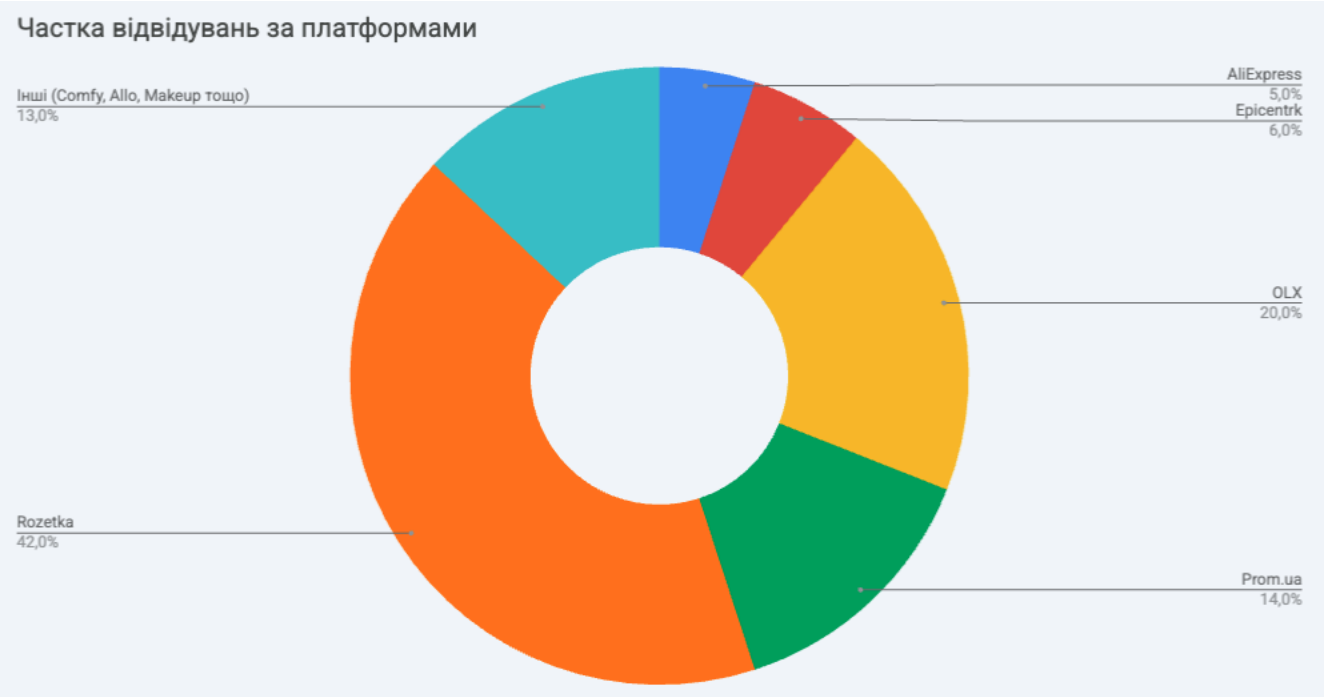


Рисунок 1.6 – Частка відвідувань за платформами

Український ринок e-commerce демонструє відновлення та зростання. За даними аналітиків групи Promodo, у 2023 році обсяг ринку електронної комерції в Україні зріс на 17% у валютному еквіваленті порівняно з 2022 роком, а гривневий виторг рітейлерів суттєво перевищив довоєнні показники 2021 року. Загальний обсяг ринку оцінюється у понад 151 млрд грн.

Сьогодні спостерігається значний зріст категорій товару таких як «Товари для дому», «Зоотовари», «Дитячі товари» та «Військове спорядження», у

порівнянні з попередніми роками де абсолютним лідером категорій товарів була електроніка та одяг.

На рисунку 1.7 наведено діаграму усереднених даних від Promodo/Bank data частки ринку відповідно категорії товарів.

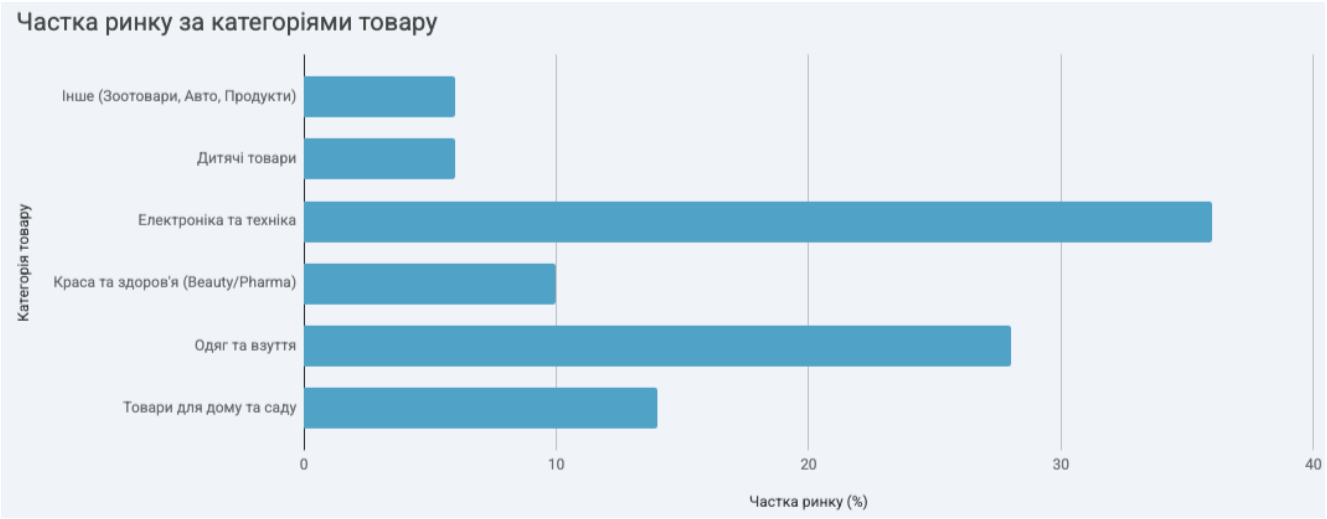


Рисунок 1.7 – Діаграма частки товарів залежно від категорії

З рисунка 1.7 бачимо, що категорії розподілились наступним чином:

– Електроніка та побутова техніка — залишається лідером за грошовим обігом (високий середній чек);

– Одяг та взуття — лідер за кількістю транзакцій

– Краса та догляд (Beauty-сегмент), демонструє стабільний ріст (лідер Makeup).

Компанія «Нова Пошта» за 2023 рік збільшила кількість посилок на 30% у порівнянні з попереднім роком, що свідчить про те, що українці переходять на модель онлайн-замовлення товарів все більше.

Середній чек, в категорії електроніка складає близько 200-250 доларів США, в категорії одяг складає в середньому 40-50 доларів США.

Важливою тенденцією стало те, що український покупець став більш вимогливим до сервісу: наявність товару, швидкість відправлення товару «день у день» та безкоштовне доставлення товару стають головними факторами вибору продавця.

Будь-яка сучасна e-commerce платформа складається з трьох основних рівнів:

– Frontend (Клієнтська частина): Візуальний інтерфейс (HTML/CSS);

– Backend (Серверна частина): Логіка роботи сайту;

– База даних: Сховище інформації.

Сучасні e-commerce платформи окрім базових рівнів таких як клієнтська частина, серверна частина та база даних, також входять додаткові модулі, що забезпечують збирання, оновлення, пошук, аналіз та захист даних. Таким чином такі платформи мають функцію не лише як інструмент онлайн-торгівлі, а повноцінну екосистему, в якій здійснюється обробка даних.

Деякі великі платформи по типу Amazon, ebay для своїх клієнтів мають API (Application Programming Interface), який забезпечує взаємодію між внутрішніми сервісами та зовнішніми клієнтами. Більшість платформ використовує REST або GraphQL API для отримання структурованих даних про товари, виконання пошукових запитів, фільтрації, керування кешованих даних, взаємодії з особистим кабінетом користувача. При наявності API на платформі, забезпечує оптимізацію передачі даних, зменшує навантаження на фронтенд та забезпечує можливість інтеграції з мобільними застосунками або з CRM-системою та інструментами аналітики. Для систем моніторингу API є важливим інструментом для доступу до інформації, але дуже часто даний інструмент відсутній, або обмежений корпоративною політикою, чи авторизаційним механізмом.

Ключовою складовою будь-якої e-commerce платформи є наявність каталогу товарів та модулі керування товарними позиціями, який включає механізм класифікації, категоризації, прив’язка товарів до груп, визначення атрибутів, додаткові опціональні варіації (наприклад, розмір або колір) та присвоєння унікального артикулу товару (SKU Stock Keeping Unit). У каталозі також може міститись метадані про рейтинги, кількість відгуків, популярність товару. Каталог як правило виступає центральним джерелом даних для всіх внутрішніх модулів, його структура забезпечує зручність індексації, пошуку та подальшої аналітики.

Також важливий елементом є система ціноутворення, яка забезпечує відображення актуальних цін, акційних пропозицій, програм лояльності. Деякі великі маркетплейси використовують алгоритми динамічного ціноутворення, тобто реагують динамічно на попит товару, поведінку покупців та дії конкурентів. Тому інформація на ціни на товар є однією з найбільш важкодоступною для моніторингу.

Як правило на платформах присутній механізм антибот, що використовується для запобігання автоматизованого збору даних. Один з найпоширенішими засобами є CAPTCHA, обмеження частоти запитів (rate limiting), визначення аномальної поведінки користувача, перевірка User-Agent та IP-адреси.

E-commerce платформах у межах функціонування формується широкий спектр даних, які можна розділити на:

- структуровані;
- напівструктуровані;
- неструктуровані.

До структурованих даних відносяться відомості про товарні позиції, ціни, залишки на складі, транзакції та інформація про продавця, що зберігається в базі даних в табличному вигляді.

До напівструктурованих даних відносяться дані представлені у вигляді HTML-коду сторінки, JSON-відповідями API, XHR-запитами та іншими форматами, які формально мають певну структуру, але не відповідають схемі реляційних баз даних.

До неструктурованих даних належать текстовий опис товару, відгуки, коментарії, мультимедійні матеріали (зображення, відео), що може створювати додаткові труднощі для автоматизованого аналізу.

Комплексність взаємодії модулів вказує на те що сучасні e-commerce платформи є надзвичайно складними об’єктами для моніторингу. Висока динамічність даних, часті зміни цін та наявність товарів, велика кількість продавців, різноманітність форматів подання даних та їх залежність від

локалізації, персоналізації суттєво ускладнюють отримання даних для моніторингу. Додатковим фактором складності є періодична зміна структури HTML, використання механізмів антибот та значні обсяги інформації, які оновлюються в режимі реального часу. Це все потребує використання спеціалізованих методів автоматичного збору, попередньої обробки та аналізу даних.

1.2. Методи збору, попередньої обробки та аналітичного опрацювання даних у середовищі електронної комерції

Сучасні e-commerce платформи містять значні обсяги даних, що включають інформацію про асортимент, зміни цін, акційні пропозиції, динаміку попиту та інші бізнес-процеси, які впливають на стан ринку. Отримання таких даних є ключовою складовою роботи інтелектуальної системи моніторингу e-commerce платформ, що здійснює аналіз ринкових тенденцій, визначає динаміку цін, наявність товарів та інші показники, ці дані є важливими для покупців та бізнесу. З огляду на це виникає необхідно розглянути методи збору, попередньої обробки та аналізу даних, для системи моніторингу товарів.

Процес збору даних в e-commerce ґрунтується на використання кількох підходів, що визначаються технічними особливостями платформи, політикою доступу до даних, обмеженнями у вигляді захисту від автоматизованих запитів та характером інформації, що підлягає моніторингу.

В більшості випадках використовується два основні методи для отримання даних з e-commerce платформи:

- звернення до офіційного прикладного програмного інтерфейсу (у випадку наявності такого), тобто використання API від платформи;
- використання автоматизованого зчитування HTML-коду вебсторінок (web-scraping).

Метод використання API платформи є найбільш структурований та надійний для отримання даних, але велика частина e-commerce платформ та

маркетплейсів не надають відкритий доступ до API, або ж є суттєві квоти на кількість запитів та доступність параметрів.

Метод web-scraping є більш універсальним, оскільки дозволяє отримувати дані у випадку за відсутності API платформи, але даний метод потребує значно складніший механізм отримання та обробки даних.

В сучасних системах моніторингу ці два підходи часто комбінуються при можливості для досягнення максимальної повноти даних. На рисунку 1.8 приведена узагальнена структурна схема збору даних з e-commerce платформ.

Рисунок 1.8 – Узагальнена структурна схема збору даних з e-commerce платформ

Під час web-scraping важливим етапом є коректний збір HTML-структури, сторінки, що відображає товар, ціну, параметри та інші дані про товар. Але на сьогодні більшість платформ використовують JavaScript, який формує контент

динамічно, тому використання просто HTTP-запитів недостатньо в таких випадках. При використанні JavaScript використовують headless-браузери, що дозволяє емулювати поведінку користувача та завантажити динамічний контент.

Після отримання даних необхідно попередньо обробити, провести нормалізацію та привести до структури, для подальшого аналізу. Оскільки попередні дані як правило містять велику кількість зайвої інформації яка не використовується для подальшого аналізу, як приклад дані рекламного блоку, елементи навігації, службові метадані та фрагменти коду, то необхідно відфільтрувати ці дані для отримання чистого набору атрибутів які будемо зберігати та аналізувати.

Оскільки інформацію можемо отримувати з різних платформ, це призводить до того, що дані відрізняються за форматом залежно від платформи, як приклад різні продавці можуть використовувати різні валютні позначки, формат відображення цін, метрики та інше. Тому для використання даних в подальшому аналізі та порівнянні необхідно їх трансформувати до узгодженого формату, тобто провести нормалізацію даних.

Аналітичне опрацювання даних в тематиці e-commerce, може охоплювати широкий спектр методів, таких як статистичні методи для визначення середніх цін, сезонні коливання та інших показників для розуміння ситуації на ринку. Часові ряди дозволяють ідентифікувати тренди змін цін, та можливість прогнозування цін у майбутньому, визначення підвищення попиту, що може бути пов’язаний з маркетинговими компаніями або святковими періодами чи іншими зовнішніми чинниками.

Основні методи аналітичного опрацювання даних:

1. Статистичний аналіз дозволяє визначити середні ціни, амплітуду їх коливань та сезонність.
2. Аналіз часових рядів (Time Series Analysis) використовується для ідентифікації трендів та прогнозування майбутніх цін.
3. Зіставлення товарів (Product Matching): Одним із найскладніших завдань аналізу є ідентифікація ідентичних товарів на різних сайтах

(наприклад, зіставлення "iPhone 15" та "Apple iPhone 15 128GB Black"). Для цього використовуються методи нечіткого пошуку та машинного навчання.

4. Обробка відгуків (NLP): Для аналізу репутації застосовуються методи обробки природної мови, що дозволяють оцінити тональність відгуків покупців.

Також використовуються агреговані дані, які використовуються для побудови звітів, візуалізації, рекомендаційні моделі. Важливо також забезпечувати швидке оновлення даних, оскільки в наш час на e-commerce платформах дані змінюються дуже швидко.

Отже, методи збору, попередньої обробки та аналітичного опрацювання даних в галузі e-commerce формує багаторівневу систему, яка містить автоматизований доступ до даних e-commerce платформ або маркетплейсів, виконує нормалізацію даних, застосовує статичні та аналітичні методи та формує структуризований результат. Висока складність внутрішніх процесів e-commerce платформ, швидка зміна інформації та значна варіативність даних зумовлюють необхідність використання адаптивних, масштабованих та інтелектуальних підходів, які здатні забезпечити стабільне функціонування систем моніторингу в умовах динамічного ринкового середовища.

1.3. Технології web-скрепінгу та інтелектуального аналізу інформації: сучасний стан, тенденції та перспективи розвитку

Технології web-скрепінгу бере свій початок від перших років Інтернету, коли вебресурси були статичні, а їх структура була мінімально складною. Це обумовлено тим, що в той час вебсторінки створювались статичним HTML-документом, що робило процес отримання даних відносно простим та легко автоматизувався процес парсингу. В той час основними інструментами розробників для створення парсерів використовувались мови програмування Perl, PHP та ранні версії Python. Тому web-скрепінг в той час здійснювався шляхом написанням спеціалізованих скриптів, що орієнтувались на конкретні сайти.

Розроблявся алгоритм під конкретний сайт для аналізу HTML-коду, знаходження необхідних тегів та вилучання текстових даних. Даний підхід на той час демонстрував ефективність для невеликого обсягу інформації, але не даний підхід майже не піддавався масштабуванню, що призвело до обмеження використання у великих аналітичних проєктах.

З розвитком вебтехнологій все більше створювались вебресурси з динамічним інтерфейсом, що використовували JavaScript-генерації контенту та асинхронні HTTP-запити (AJAX, Fetch API). Це призвело до того, що скриптові методи парсингу втратили ефективність. Разом з цим з’явилися перші інструменти для структурованого парсингу HTML-документів, на той час дуже широко використовувалась бібліотека BeautifulSoup. Дана бібліотека використовувала високоабстрагований інтерфейс для навігації DOM-дерева та суттєво зменшила складність вилучення даних зі сторінок навіть у випадку некоректною або неповною HTML-структурою. Бібліотека BeautifulSoup стала важливим етапом розвитку до напівавтоматизованого, структурованого парсингу.

Розглянемо основні бібліотеки для роботи HTML-структурами:

- BeautifulSoup, бібліотека для мови програмування Python. Вона дозволяє спростити процес парсингу HTML-документів. Дозволяє швидко створити скрипти для вилучення інформації, обробляючи DOM-структуру сторінок, та автоматизувати процес.
- Scrapy, фреймворк розроблений для розв'язання задач, які пов’язані з масовим збором даних. Даний фреймворк забезпечує не тільки створення парсерів, а й підтримує асинхронну обробку запитів, що забезпечує ефективність роботи з великим обсягом інформації.
- Selenium, інструмент який дозволяє працювати з динамічним контентом. Selenium забезпечує автоматизовану роботу з браузерами та взаємодію з вебсторінками імітуючи справжнього користувача.

PHP також використовується для web-скрепінгу. Класичний PHP скрипт виконується синхронно, це означає, що коли скрипт відправляє запит на сайт то скрип стає на паузу доки не отримає відповідь. До недоліків можна віднести

низьку швидкість при обробці великою кількістю сторінок, оскільки у випадку якщо сайт відповідає 2 секунди, то парсинг 10000 товарів займе понад 30 хвилин. Сучасні бібліотеки такі як Goutte, Symfony Panther, намагаються прискорити роботу, але архітектурно PHP залишається орієнтованим на короткі вебзапити, а не на довготривалі фонові процеси моніторингу.

Java сформувала окремий напрямок комерційних рішень для інтеграції, масштабованого збору та подальшої обробки вебданих. В порівнянні з інтерпретованими мовами програмування Java має наступні переваги: типізація, висока продуктивність в багатопотокових системах, розвинену екосистему бібліотек та фреймворків для роботи з мережевими протоколами.

Для роботи зі статичними вебсторінками використовується бібліотека Jsoup, яка забезпечує синтаксичний аналіз HTML-документів, навігацію DOM-структурою, очищення HTML та вилучання необхідних атрибутів. Використання Jsoup є оптимальне рішення для парсингу статичних сайтів.

Для роботи з динамічним контентом використовується Selenium WebDriver, що дозволяє завантажувати повноцінні вебінтерфейси, які використовують JavaScript та дозволяє взаємодіяти з елементами DOM.

При роботі на Java окрім використання Selenium, також використовується HtmlUnit. HtmlUnit це легкий headless-браузер, який імітує поведінку браузера без графічного інтерфейсу, що значно знижує навантаження на систему.

WebMagic це повноцінний Java-фреймворк для web-скрепінгу, що є аналогічним по функціональності Python-фреймворку Scrapy. Фреймворк WebMagic містить модулі планування, завантаження, парсингу та має розширювану архітектуру, що забезпечує легкість інтеграції.

Розглянувши різні мови програмування, фреймворки, бібліотеки та інструменти зробимо порівняння основних характеристик. В таблиці 1.1 наведенні порівнянні характеристики технологій web-скрепінгу.

Сучасна система моніторингу e-commerce це вже не просто скрипт, а розподілена програмна архітектура, що включає модулі планування (Scheduling), завантаження (Downloading), парсингу (Parsing) та очищення даних (Cleaning).

Таблиця 1.1

Порівняльні характеристики технологій web-скрепінгу

Характеристики	Python (Scrapy)	PHP (Panther)	Java (Jsoup/Selenium)
Архітектура	Асинхронна	Синхронна	Багатопотокова
Типізація	Динамічна	Динамічна	Статична (Надійна)
Складність підтримки	Середня	Висока	Низька (завдяки структурі)
Продуктивність	Висока	Низька	Дуже висока

Зростає роль headless-браузерів та систем керування чергами завдань, які забезпечують масштабованість та стабільність процесу збору даних. Окремою тенденцією є перехід від неструктурованого HTML-скрепінгу до використання офіційних API, що підвищує надійність та коректність отримання даних.

При розробці сучасної інтелектуальної системи моніторингу e-commerce необхідно використовувати Java, оскільки система буде надійніше та ефективнішою внаслідок багатопотоковості.

Java дозволяє створити справжні паралельні потоки на відміну від синхронного PHP, або Python де асинхронність часто обмежена одним ядром.

Архітектура типового Java-скрапера будується на базі Thread Pool (Пул потоків):

- Створюється ExecutorService з фіксованою кількістю потоків.
- Задачі додаються в чергу.
- Кожен потік забирає URL, завантажує сторінку через Jsoup (для статики) або Selenium WebDriver (для динаміки).

Разом із технічним розвитком технологій web-скрепінгу зростає актуальність питань, пов’язаних з етичними та правовими обмеженнями збору інформації. Багато вебресурсів регламентують доступ до своїх даних за допомогою файлів robots.txt або умов користування, які обмежують частоту запитів чи забороняють автоматизований збір інформації. Ігнорування таких обмежень може призвести до блокування IP-адрес, спотворення результатів

моніторингу або юридичних наслідків. У зв’язку з цим сучасні системи збору даних повинні враховувати механізми контролю навантаження, дотримуватися принципів відповідального використання ресурсів та забезпечувати відповідність чинним нормативним вимогам.

Разом із розвитком технологій збору даних розвивались методи інтелектуального аналізу інформації. Застосування алгоритмів машинного навчання надало змогу значно підвищити якість класифікації, виявлення закономірностей, прогнозування та оцінки поведінкових характеристик користувачів. У сфері електронної комерції машинне навчання дає можливість автоматично ідентифікувати дублікати товарів, зіставляти аналогічні позиції між різними платформами, оцінювати ціноутворення, виявляти аномальні зміни цін або підозрілу активність продавців. Це стало особливо важливим у контексті розширення кількості продавців на маркетплейсах та зростання конкуренції, що супроводжується появою недобросовісних практик, маніпуляцій із цінами або штучного завищення рейтингів.

Особливо важливу роль відіграють технології обробки природної мови (NLP), які дають змогу працювати з великими масивами текстових даних: відгуками користувачів, описами товарів, характеристиками, рекламними повідомленнями та інструкціями продавців. Застосування моделей класифікації тексту, аналізу тональності та тематичного моделювання дозволяє формувати загальні оцінки якості товарів, визначати найчастіше згадувані проблеми, виявляти потенційні фальсифікації та навіть прогнозувати рівень попиту на підставі змін у структури відгуків. Такі підходи суттєво підсилюють можливості систем моніторингу, оскільки дозволяють аналізувати не лише формальні ознаки товару, а й поведінковий контекст, який відображає реальну оцінку споживача.

Перспективи розвитку технологій web-скрепінгу тісно пов’язані з подальшою інтеграцією методів інтелектуального аналізу даних та машинного навчання. У майбутньому очікується активне застосування моделей глибокого навчання для автоматичної адаптації алгоритмів збору даних до змін структури вебсторінок, а також для семантичного аналізу контенту без жорсткої прив’язки

до HTML-розмітки. Поєднання web-скрепінгу з технологіями обробки природної мови, кластеризації та виявлення аномалій дозволяє створювати інтелектуальні системи моніторингу, здатні не лише збирати інформацію, та формувати аналітичні висновки в режимі реального часу. Це робить такі системи перспективним інструментом у сфері електронної комерції.

В даному розділі було розглянуто теоретичні основи функціонування електронної комерції та сучасні підходи до моніторингу даних в сфері e-commerce. Розглянуто структуру та функціональні особливості e-commerce платформ, визначено основні моделі взаємодії учасників ринку (B2B, B2C, C2C, B2G, G2C тощо) та охарактеризовано типи даних, що формуються в процесі їх функціонування. Обґрунтовано, що e-commerce платформи є складним об'єктом моніторингу через високу динамічність інформації, наявність механізмів антибот-захисту, часті зміни структури вебсторінок та значні обсяги структурованих, напівструктурованих і неструктурованих даних. Такод проаналізовано основні методи збору, попередньої обробки та аналітичного опрацювання даних у середовищі електронної комерції. Визначено, що найбільш поширеними підходами до отримання даних є використання офіційних API та технологій web-скрепінгу, які часто застосовуються у комбінованому вигляді. Розглянуто етапи фільтрації, нормалізації та трансформації даних, а також методи їх подальшого аналізу, зокрема статистичний аналіз, аналіз часових рядів, зіставлення товарних позицій та обробку текстових даних з використанням методів машинного навчання та обробки природної мови. Проаналізовано сучасний стан, тенденції та перспективи розвитку технологій web-скрепінгу та інтелектуального аналізу інформації. Проведено порівняльний аналіз інструментів і мов програмування, що використовуються для збору даних, та обґрунтовано доцільність використання мови програмування Java для побудови масштабованих і багатопотокових систем моніторингу e-commerce платформ. Окрему увагу приділено питанням етичних і правових обмежень збору даних, а також перспективам інтеграції web-скрепінгу з методами глибокого навчання та інтелектуального аналізу.

РОЗДІЛ 2. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПРОЄКТУВАННЯ АРХІТЕКТУРИ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ МОНІТОРИНГУ

2.1. Аналіз існуючих систем моніторингу цін, товарів і ринкових тенденцій на e-commerce платформах

Сучасний розвиток електронної комерції становить складний та висококонкурентний ринок, в межах якого ціна, асортимент товарів, умови доставки та маркетингові стратегії змінюється швидко. Для ефективного функціонування та конкурентної здатності бізнесу необхідні інструменти, що дозволяють оперативно отримувати актуальну інформацію про стан ринку. Також споживачі зацікавлені в отримані актуальної інформації про ціну на товар, наявності та інше. Системи моніторингу e-commerce платформ виконують збір, аналіз та узагальнення даних щодо товарів, цін, продавців та поведінку споживачів.

В цілому ринок програмних рішень для моніторингу e-commerce є достатньо насиченим та включає в себе широкий спектр інструментів, які відрізняються за функціональним призначенням, джерелами даних, цільовою аудиторією та рівнем аналітичної глибини. Тому для формування вимог до інтелектуальної системи моніторингу яку ми проектуємо необхідно провести аналіз існуючих рішень, виявити їхні переваги та недоліки.

У загальному випадку всі існуючі рішення у сфері моніторингу e-commerce можна умовно поділити на декілька великих класів, залежно від їхнього призначення та способу отримання даних. Найбільш поширеними є B2C-агрегатори, орієнтовані на кінцевих покупців, професійні B2B SaaS-рішення для бізнесу, а також вузькоспеціалізовані сервіси, що працюють з окремими маркетплейсами або обмеженим набором товарних категорій. На рисунку 2.1 приведено класифікацію систем моніторингу за типом цільової аудиторії (кінцеві споживачі, бізнес-клієнти, спеціалізовані сервіси).

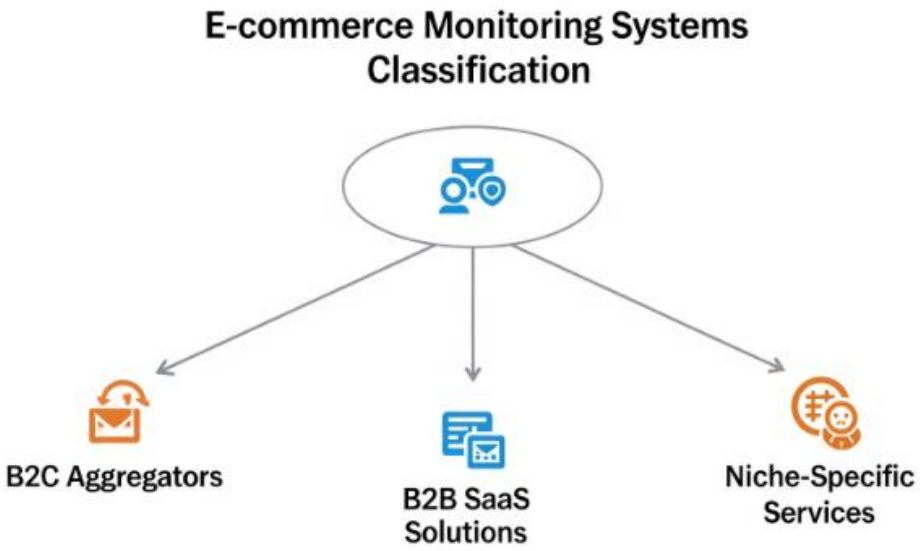


Рисунок 2.1 – Класифікація систем моніторингу за типом цільової аудиторії

Отже ми розділили на три групи системи моніторингу e-commerce, які орієнтовані на різці цільові аудиторії.

До першою групи віднесли B2C-агрегатори, до них відносяться прайс-агрегатори та маркетплейс-агрегатори, прикладом таких сервісів є як Hotline.ua, Price.ua або Google Shopping. Основне призначення подібних платформ є надання кінцевому користувачу зручного інтерфейсу для порівняння ціни товару в різних інтернет-магазинах. Принцип роботи даних систем базується на використанні XML-фідів, які інтернет магазини самостійно формують та надсилають на платформу агрегатора. XML-фіди містять структуровану інформацію про назви товарів, ціни, наявність та базові характеристики.

До переваг систем прайс-агрегаторів та маркетплейс-агрегаторів можна віднести мінімальні вимоги до обчислювальних ресурсів агрегатора та відносну простоту інтеграції для інтернет-магазинів.

До недоліків можна віднести те, що актуальність даних повністю залежить від частоти оновлення XML-фідів з боку інтернет-магазина, тобто система залежить повністю від даних продавця. В деяких випадках ситуація може бути такою що ціни на товар на стороні агрегатору може буди не актуальною, тобто ціна на сайті продавця змінилась а на стороні агрегатора містить застарілі значення. Крім того, агрегатори практично не здійснюють перевірку достовірності

наданої інформації та не забезпечують глибокої аналітики, зокрема аналізу історії цін, репутації продавців або поведінкових характеристик споживачів.

На рисунку 2.2 приведена спрощена структурна схема роботи прайс-агрегатора на основі роботи з XML-фідами, які приймають від інтернет-магазинів.

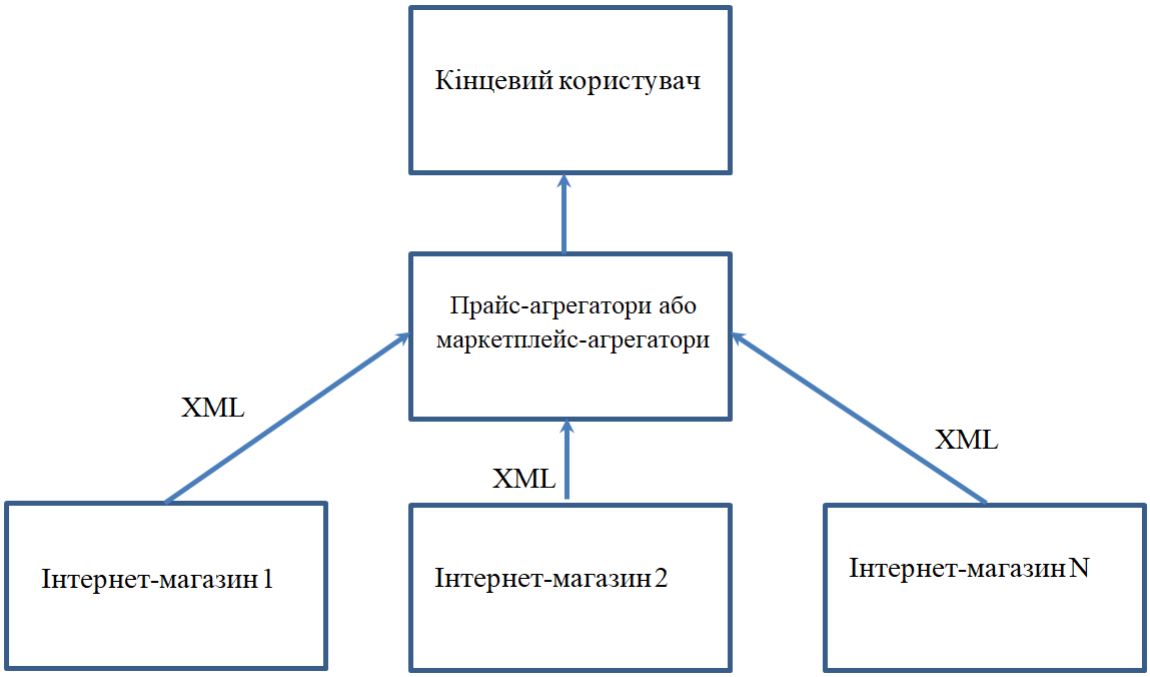


Рисунок 2.2 – Структурна схема роботи прайс-агрегатора на основі XML-фідів

Другий тип систем які націлені на роботу B2B SaaS-рішення, тобто орієнтовані на потреби середнього та великого бізнесу. До таких систем що відносяться сервіси Competera, Price2Spy, Prisync, PriceIndex. Дані системи мають значно ширший функціонал в порівнянні з B2C-агрегаторами, вони можуть містити автоматичне виявлення нових товарів, порівняння цін товарів між різними магазинами, можуть створювати історію цін на товар, а також інструменти динамічного ціноутворення.

Дані системи мають складнішу архітектуру у порівнянні з попереднь описаною системою яка працює тільки з XML-фідами. Дані системи мають декілька модулів які відповідають за різні задачі. Типова архітектура містить наступні модулі: модуль парсингу або комбінований з API, аналітичний модуль який поєднує в собі обробку даних, модуль інтерфейс користувача. На рисунку 2.3 приведено узагальнену структурну схему архітектури B2B SaaS-системи

моніторингу. Користувач може отримувати доступ до результатів за допомогою вебінтерфейсу або API.

Дані системи мають гарну функціональність але мають деякі обмеження та недоліки. До недоліків можна віднести високу вартість підписки, складність налаштування та інтеграції, закритість внутрішніх алгоритмів аналізу, що унеможливлює глибоке налаштування під специфічні бізнес-процеси.

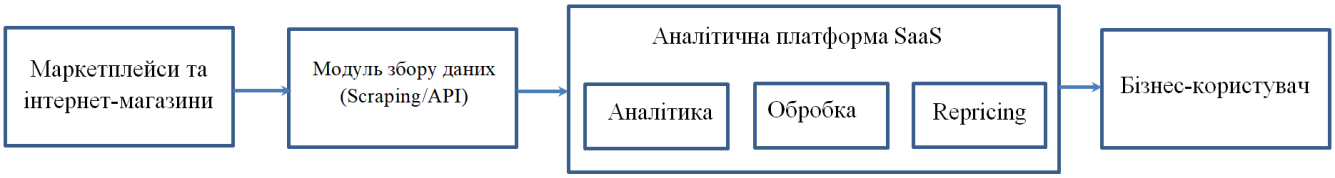


Рисунок 2.3 – Узагальнена схема архітектури B2B SaaS-системи моніторингу

Третій тип це вузькоспеціалізовані сервіси моніторингу, які як правило працюють з однією конкретною платформою, та можуть працювати з обмеженим набором ресурсів. Приклад такою вузькоспеціалізованої системи є сервіс CamelCamelCamel, який орієнтований на моніторинг цін товарів на платформі Amazon. Подібні системи забезпечують достатньо високу точність збору даних та детальну історію змін цін. До недоліків даної системи можна віднести жорстко прив'язану архітектуру та алгоритми оскільки є прив'язка до конкретної платформи маркетплейсу або інтернет-магазину. Це робить даний тип системи непридатної до масштабування або роботи з іншою платформою, як приклад неможливість CamelCamelCamel працювати з Rozetka або Allo.

На рисунку 2.4 приведено спрощену архітектуру вузькоспеціалізованого сервісу моніторингу.

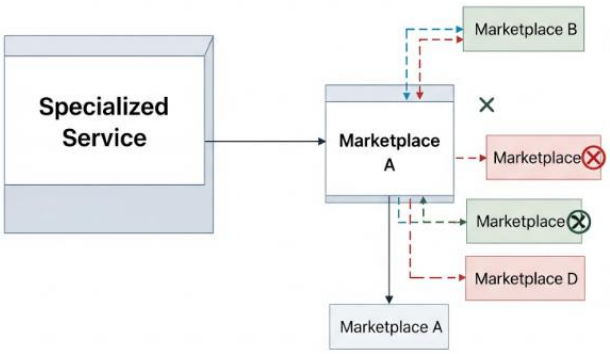


Рисунок 2.4 – Архітектура вузькоспеціалізованого сервісу моніторингу

Проаналізувавши існуючі системи доцільно звернути увагу також на тип даних що використовують системи для аналізу, та результати аналізу. Більшість B2C-агрегаторів обмежуються лише поточними цінами та базовими характеристиками товарів. B2B SaaS-рішення можуть надавати історію цін, інформацію про конкурентів. Водночас жодна з розглянутих категорій не забезпечує повноцінного інтелектуального аналізу текстових даних, таких як відгуки користувачів, опис товарів або поведінкові сигнали споживачів, у поєднанні з гнучким і масштабованим механізмом збору інформації.

Виходячи з цього було створено узагальнений порівняльний аналіз існуючих рішень та проектовану систему яка усуває недоліки існуючих, порівняльний аналіз наведено в таблиці 2.1. З даної таблиці бачимо, що відсутні універсальні системи, здатні одночасно забезпечити глибоку аналітику, гнучкість налаштування, актуальність даних і доступну вартість для малого та середнього бізнесу.

Таблиця 2.1

Порівняльний аналіз існуючих систем

Критерій	B2C-агрегатори (Hotline, Price.ua, Google Shopping)	B2B SaaS-рішення (Competera, Price2Spy, Prisync)	Проєктована система
Цільова аудиторія	Кінцеві споживачі (покупці)	Середній та великий бізнес	Малий та середній бізнес
Основне призначення	Порівняння поточних цін	Моніторинг конкурентів, оптимізація ціноутворення	Комплексний моніторинг ринку та аналітика
Джерела даних	XML-фіди, надані магазинами	Прямий web-скрепінг, API	Прямий web-скрепінг, API
Актуальність даних	Середня, залежить від оновлення фідів	Висока	Висока (контрольована системою)
Історія змін цін	Обмежена або відсутня	Повна	Повна
Аналіз відгуків користувачів	Поверхневий або відсутній	Частково (у розширених тарифах)	Інтелектуальний (NLP, аналіз тональності)
Аналіз ринкових тенденцій	Обмежений	Розширений	Розширений з використанням ML
Гнучкість налаштування	Низька	Середня	Висока (модульна архітектура)
Масштабованість	Обмежена	Висока	Висока

Продовження таблиці 2.1

Критерій	B2C-агрегатори (Hotline, Price.ua, Google Shopping)	B2B SaaS-рішення (Competera, Price2Spy, Prisync)	Проєктована система
Прозорість алгоритмів	Відсутня	Закрита	Повна (власна реалізація)
Вартість використання	Безкоштовно або CPA-модель	Висока підписка	Низька (власна розробка)
Можливість адаптації під локальні маркетплейси	Обмежена	Обмежена	Повна
Ціль використання	Інформаційна підтримка покупця	Оптимізація бізнес-рішень	Підтримка управлінських рішень

На основі проведеного аналізу можна зробити висновок, що більшість існуючих рішень або орієнтовані на кінцевого користувача та не надають достатньої аналітичної глибини, або є комерційними продуктами з високою вартістю та обмеженою можливістю адаптації. Це обґрунтовує доцільність розробки власної інтелектуальної системи моніторингу e-commerce, побудованої на основі технологій web-скрепінгу, багатопотокової архітектури Java та методів інтелектуального аналізу даних, що детально розглядається у наступних підрозділах.

2.2. Проєктування архітектури інтелектуальної системи з використанням технологій web-скрепінгу та автоматизованого збору даних

Проєктування архітектури програмного забезпечення є одним із ключових етапів життєвого циклу розробки, оскільки саме на цьому етапі визначаються структурні рішення, які безпосередньо впливають на надійність, продуктивність, масштабованість та подальшу підтримуваність системи. Для інтелектуальних систем моніторингу e-commerce платформ ця задача ускладнюється специфікою предметної області, що характеризується високою динамікою даних, неоднорідністю джерел інформації та наявністю технічних обмежень з боку вебресурсів. На відміну від класичних інформаційних систем, де дані надходять із

контрольованих джерел, системи моніторингу e-commerce працюють з відкритими вебплатформами, структура яких може змінюватися без попередження. Крім того, такі платформи часто застосовують механізми захисту від автоматизованого доступу, що вимагає від архітектури системи гнучкості та адаптивності. У зв'язку з цим архітектурні рішення повинні забезпечувати не лише коректний збір даних, але й стійкість до помилок, можливість повторного виконання операцій та ефективне використання обчислювальних ресурсів.

Архітектура інтелектуальної системи моніторингу базується на модульному підході (рисунок 2.5), що дозволяє забезпечити масштабованість та гнучкість у процесі її функціонування. Першим ключовим елементом є модуль web-скрепінгу, який відповідає за автоматизоване вилучення даних із веб-сторінок та API інтернет-магазинів. Для реалізації цього модуля застосовуються сучасні інструменти та бібліотеки, що підтримують роботу з HTML-структурами, а також механізми обходу обмежень, пов'язаних із захистом від автоматизованих запитів. Важливим аспектом є використання планувальників завдань, які забезпечують регулярність та актуальність отриманих даних. Особливістю проєктованої системи є необхідність паралельної обробки великої кількості вебсторінок. Операції завантаження сторінок з мережі мають значні затримки, тому послідовна обробка запитів призводить до неприйнятного часу виконання. Це обумовлює потребу у використанні багатопотокової архітектури, здатної одночасно обробляти десятки або сотні завдань збору даних.

Модуль ETL (Extract–Transform–Load) це другий компонент ахітектури, він виконує функції вилучення, трансформації та завантаження даних у сховище. На етапі вилучення інформація надходить із різних джерел, після чого здійснюється її очищення від шумів, нормалізація та уніфікація форматів. Завантаження даних у сховище забезпечує їх централізоване зберігання та доступність для подальшої обробки. Для цього можуть використовуватися як реляційні бази даних, що забезпечують структуроване зберігання, так і NoSQL-рішення, орієнтовані на роботу з великими обсягами неструктурованої інформації.

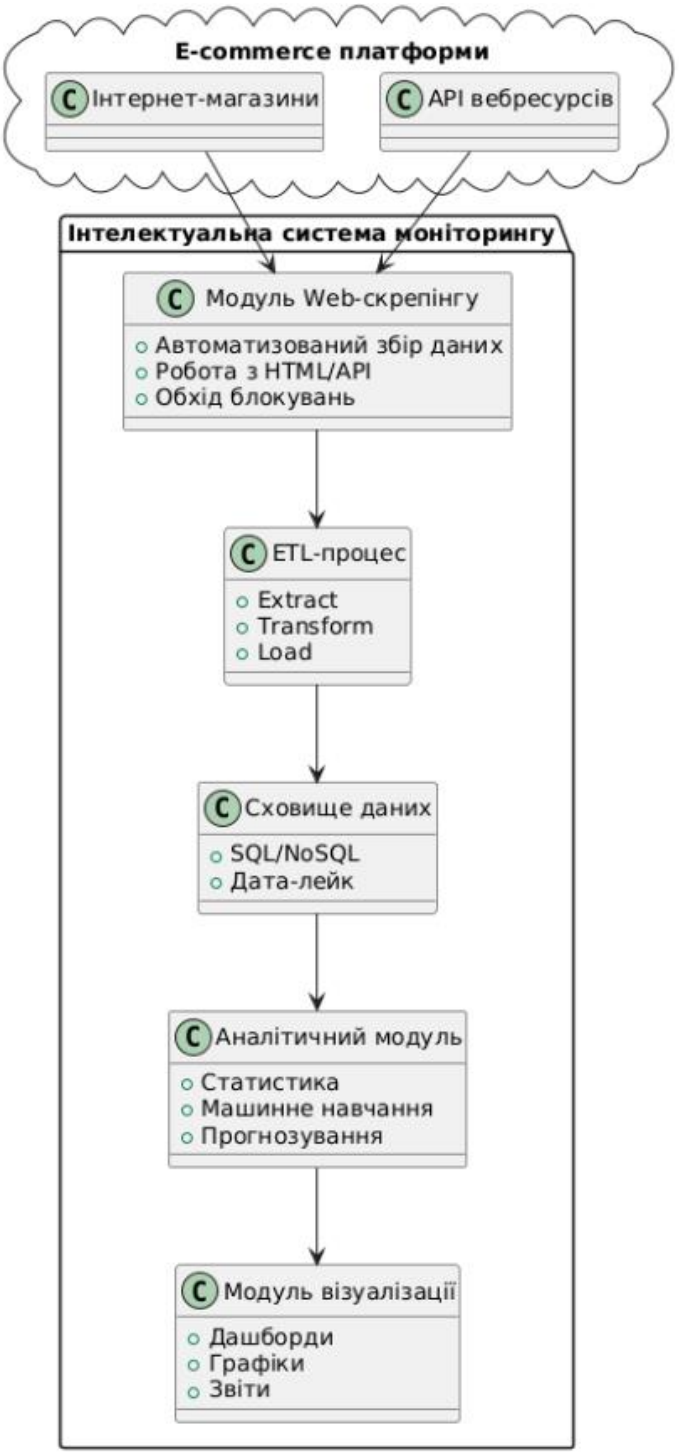


Рисунок 2.5 – Структурна схема архітектури інтелектуальної системи моніторингу e-commerce

Система повинна містити також аналітичний модуль, він інтегрує методи статистичного аналізу та алгоритми машинного навчання. Його завдання полягає у виявленні закономірностей, прогнозуванні цінових тенденцій та класифікації товарів за визначеними ознаками. Використання інтелектуальних алгоритмів

дозволяє підвищити точність прогнозів та забезпечити глибше розуміння ринкових процесів. Аналітичний модуль тісно пов’язаний із модулем візуалізації, який відповідає за представлення результатів у вигляді інтерактивних графіків, дашбордів та звітів, що значно полегшує інтерпретацію даних користувачами.

З урахуванням аналізу предметної області та існуючих рішень, проведеного вище, до архітектури інтелектуальної системи моніторингу були сформульовані узагальнені вимоги, що визначають її функціональні та технічні характеристики. Система повинна забезпечувати можливість інтеграції з різними e-commerce платформами без необхідності модифікації ядра програмного забезпечення. Це означає, що логіка роботи з конкретним вебресурсом має бути ізольованою та легко замінною, що дозволяє швидко адаптувати систему до змін у структурі сайтів та появи нових джерел даних.

Архітектура повинна підтримувати масштабування як за кількістю джерел інформації, так і за обсягами даних, що обробляються. При цьому важливою є не лише горизонтальна масштабованість, яка забезпечує можливість паралельної обробки великої кількості вебсторінок, але й поступове розширення функціоналу системи, зокрема шляхом додавання нових аналітичних модулів. Такий підхід гарантує довготривалу підтримуваність та розвиток системи відповідно до потреб користувачів і динаміки ринку.

Окрему увагу необхідно приділити надійності та відмовостійкості. Помилки мережевого з’єднання, тимчасова недоступність сайтів або зміни у структурі вебсторінок не повинні призводити до зупинки всієї системи. Архітектура має передбачати механізми обробки винятків, повторних спроб та журналювання подій, що забезпечує стійкість до помилок та можливість своєчасного відновлення процесів збору даних.

Крім того, система повинна відповідати вимогам продуктивності та оптимального використання ресурсів. Оскільки операції завантаження вебсторінок супроводжуються значними затримками, архітектура має підтримувати багатопотокову або асинхронну обробку, що дозволяє одночасно

виконувати десятки чи сотні завдань збору даних. Це суттєво скорочує час виконання та підвищує ефективність роботи системи.

При розробці потрібно враховувати вимогу гнучкості та адаптивності. Структура e-commerce платформ може змінюватися без попередження, тому архітектура повинна передбачати можливість швидкої модифікації алгоритмів збору та обробки даних. Використання конфігураційних файлів, параметризованих налаштувань та модульної структури дозволяє адаптувати систему до нових умов без значних витрат часу.

Підводячі підсумки, загальні вимоги до архітектури інтелектуальної системи моніторингу в e-commerce охоплюють інтеграційні можливості, масштабованість, надійність, продуктивність, гнучкість та безпеку. Виконання цих вимог забезпечує створення ефективного та стійкого інструменту, здатного працювати в умовах високої динаміки та складності електронної комерції.

2.2.1. Обґрунтування вибору технологічної платформи Java

У процесі розробки інтелектуальної системи моніторингу даних для e-commerce платформ було обрано технологічну платформу Java як основне середовище реалізації програмного забезпечення. Такий вибір зумовлений низкою технічних, функціональних та експлуатаційних переваг, які відповідають вимогам до масштабованості, надійності та кросплатформенності системи.

Java є однією з найпоширеніших мов програмування загального призначення, яка підтримує об’єктно-орієнтований підхід, що дозволяє ефективно моделювати складні предметні області. Завдяки архітектурній незалежності (платформа Java Virtual Machine), Java забезпечує можливість запуску програмного забезпечення на різних операційних системах без необхідності модифікації коду, що є критично важливим для систем, які можуть розгортатися у хмарному середовищі або на локальних серверах.

Ми будемо використовувати Java SE (Standard Edition), яка є основною платформою для розробки широкого спектра додатків, включаючи настільні,

серверні та корпоративні системи. Java SE пропонує багатий набір стандартних бібліотек і API, що охоплюють різні аспекти програмування, такі як робота з мережею, багатопоточність, обробка колекцій даних, безпека, обробка винятків та управління пам'яттю. Це дозволяє розробникам створювати високонадійні та ефективні додатки без необхідності залучення сторонніх інструментів або фреймворків, що підвищує рівень контролю та безпеку проекту.

Наш вибір обумовлений тим що Java SE забезпечує високу стабільність та сумісність із сучасними технологіями та фреймворками, що робить її ідеальною платформою для побудови масштабованих систем. Вона підтримує модульну архітектуру, що дозволяє легко розширювати функціональність та інтегрувати нові компоненти без порушення існуючої системи. Це особливо важливо для систем моніторингу, де необхідна надійність та швидка адаптація до змін у вимогах.

Завдяки своїй універсальності та широкому спектру можливостей, Java SE є оптимальним вибором для розробки різноманітних додатків, від простих інструментів до складних корпоративних систем, що вимагають високої продуктивності та безпеки. Вона також підтримує сучасні стандарти та технології, що забезпечує довгострокову перспективу розвитку та інтеграцію з іншими системами та платформами..

Однією з ключових переваг Java є її розвинута екосистема бібліотек та фреймворків, зокрема для роботи з вебресурсами, обробки даних, побудови багатопотокових додатків та інтеграції з базами даних. Для реалізації модулів web-скрепінгу можуть бути використані бібліотеки типу Jsoup або Selenium WebDriver, які мають стабільну підтримку та добре документовані API. Для обробки великих обсягів даних Java SE пропонує ефективні засоби роботи з потоками, що дозволяє реалізувати паралельну обробку запитів до вебресурсів, зменшуючи час виконання та підвищуючи продуктивність системи.

З точки зору безпеки, Java має вбудовані механізми контролю доступу, управління пам’яттю та обробки винятків, що дозволяє створювати надійні та захищені додатки. Крім того, платформа активно підтримується спільнотою

розробників та має регулярні оновлення, що забезпечує її актуальність та відповідність сучасним стандартам програмування.

Можливість інтеграції Java-додатків з іншими технологіями додає даній мові більше функціональності, зокрема RESTful API, системами управління базами даних (MySQL, PostgreSQL), аналітичними платформами (Apache Spark, Hadoop) та засобами візуалізації (наприклад, інтеграція з JavaScript-фреймворками через вебінтерфейси). Це дозволяє побудувати гнучку, розширювану архітектуру, яка відповідає вимогам до інтелектуальної системи моніторингу.

Тож вибір технологічної платформи Java, зокрема її стандартного середовища Java SE, є обґрунтованим з точки зору технічної доцільності, відповідності вимогам до системи, а також перспективи її масштабування, підтримки та інтеграції з сучасними інструментами обробки й аналізу даних.

2.2.2. Загальна архітектурна концепція системи

Архітектурний стиль проєктованої системи визначено як модульний моноліт із багаторівневою організацією. Такий підхід поєднує переваги монолітних систем, зокрема простоту розгортання та налагодження, з модульністю, що забезпечує чітке розмежування відповідальності між компонентами.

Система поділяється на логічні рівні, кожен з яких виконує окремий клас функцій та взаємодіє з іншими через визначені інтерфейси. Подібна архітектура дозволяє мінімізувати зв’язність між модулями та спростити супровід системи у процесі її експлуатації (рисунок 2.6).

Верхній рівень - шар представлення. Він відповідає за взаємодію користувача із системою та включає веб-інтерфейс і модулі візуалізації результатів моніторингу у вигляді дашбордів, графіків та звітів. Основна мета цього шару полягає у забезпеченні зручного доступу до даних та аналітичних результатів, а також у формуванні інтуїтивно зрозумілого інтерфейсу для кінцевих користувачів.

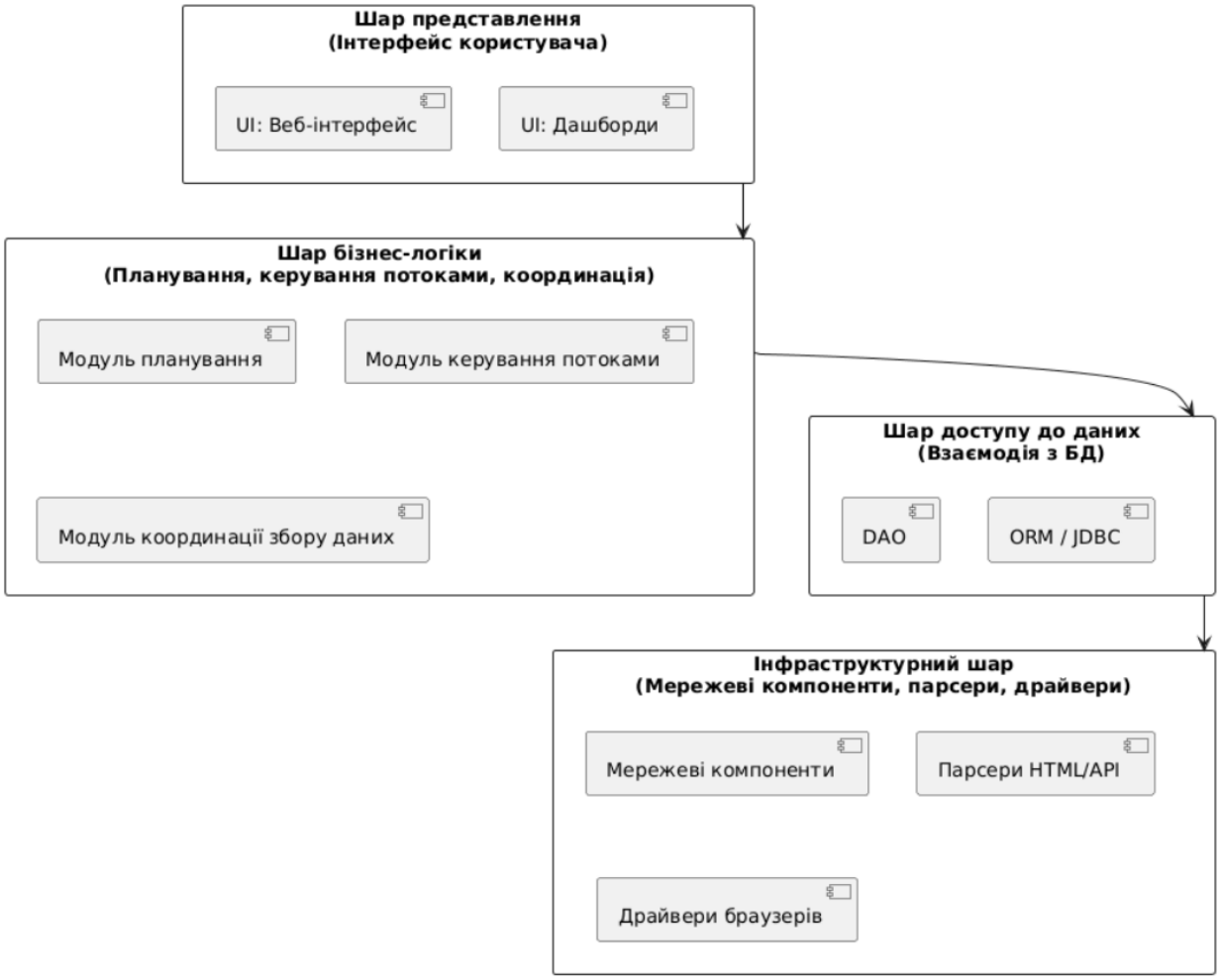


Рисунок 2.6 – Загальна архітектура інтелектуальної системи моніторингу e-commerce платформ.

Другий рівень - шар бізнес-логіки. Він реалізує ключові функції системи, зокрема планування процесів збору даних, керування потоками та координацію взаємодії між різними модулями. Саме тут визначаються правила та алгоритми роботи системи, що забезпечують узгодженість і ефективність виконання завдань. Бізнес-логіка виступає центральним ядром, яке поєднує інтерфейс користувача з механізмами доступу до даних та інфраструктурними компонентами.

Третій рівень - шар доступу до даних. Його завдання полягає у взаємодії з базою даних та іншими сховищами інформації. На цьому рівні реалізуються механізми DAO, ORM або JDBC, що забезпечують прозорий доступ до даних, їх збереження, оновлення та вибірку. Шар доступу до даних гарантує узгодженість

інформації та ізолює бізнес-логіку від технічних деталей роботи з конкретними системами зберігання.

Нижній рівень - інфраструктурний шар. Він включає мережеві компоненти, парсери та драйвери браузерів, які забезпечують технічну основу для збору даних із зовнішніх джерел. Саме цей шар відповідає за взаємодію з e-commerce платформами, обробку веб-сторінок та API, а також за забезпечення стабільності та продуктивності процесів збору інформації.

2.2.3. Проєктування підсистеми збору даних (Scraping Engine)

Підсистема збору даних є ключовим компонентом інтелектуальної системи моніторингу e-commerce платформ, оскільки саме на цьому етапі здійснюється взаємодія з зовнішніми джерелами та формується первинний масив інформації для подальшої аналітики. Її специфіка полягає у роботі з великою кількістю гетерогенних вебресурсів, що відрізняються структурою HTML, динамічним рендерингом контенту, формами пагінації та механізмами протидії автоматизованому доступу. Додаткові виклики створюють відмінності у представленні даних навіть в межах одного маркетплейсу (категорія товару, продавець, шаблон сторінки), що обумовлює потребу в ізоляції логіки парсингу та її незалежній еволюції.

Основною архітектурною вимогою виступає стандартизація точки входу до Scraping Engine при збереженні гнучкості реалізацій. Для цього застосовано комбінацію поведінкового патерна Strategy та породжувального патерна Factory Method. Strategy забезпечує підстановність алгоритмів вилучення даних без зміни клієнтського коду, тоді як Factory Method інкапсулює процес вибору конкретної стратегії на основі доменного імені або сигнатури URL. Така комбінація формує єдиний контракт взаємодії з підсистемою та відповідає принципу відкритості/закритості, мінімізуючи вплив змін джерел на ядро системи.

У центрі Scraping Engine знаходиться абстракція скрапера, яка визначає чіткий контракт для всіх реалізацій парсерів: встановлення мережевої сесії,

завантаження сторінок (у тому числі з підтримкою динамічного контенту), вилучення семантично значущих атрибутів (назва, ціна, наявність, рейтинг, відгуки), а також нормалізацію та валідацію даних перед передачею у шар доступу до даних. Конкретні реалізації інкапсулюють роботу з DOM та, за потреби, використовують headless-браузери або інтеграції з API для обходу клієнтського рендерингу. Це дозволяє відокремити особливості верстки та поведінки окремих сайтів від узагальнених механізмів планування, логування та контролю потоку.

Вибір реалізації скрапера делегується фабричному компоненту, який аналізує URL, підтримує реєстр доступних провайдерів та повертає відповідний екземпляр класу. Додавання нового джерела зводиться до імплементації нового скрапера та його реєстрації у фабриці без модифікації існуючого коду. Така організація не лише дотримується принципу відкритості/закритості, але й підвищує тестованість: кожна реалізація може покриватися модульними тестами з фікстурами HTML/API, а фабрика — контрактними тестами на коректний роутинг.

Суттєвим аспектом є експлуатаційна надійність та продуктивність підсистеми. Підтримуються асинхронна або багатопотокова обробка запитів, керування швидкістю звернень (rate limiting), ротація user-agent та проксі, повторні спроби з експоненціальним бекофом, а також детальне журналювання для трасування помилок. Вбудовані механізми обробки виключень та ізоляція збоїв на рівні окремих завдань запобігають каскадним відмовам. Конфігурація (частота опитування, таймаути, правила парсингу) винесена у зовнішні профілі, що спрощує адаптацію до змін на платформах без перезбирання системи.

Узгодженість даних забезпечується введенням проміжних DTO-структур і схем нормалізації, які вирівнюють гетерогенність полів різних джерел та гарантують стабільні контракти на межі з шаром доступу до даних. Разом ці рішення формують підсистему збору, що є гнучкою, розширюваною та стійкою до змін e-commerce середовища, забезпечуючи якісну та придатну до аналітики

інформаційну базу для всієї системи. На рисунку 2.7 номер зображено UML-діаграму класів підсистеми збору даних.

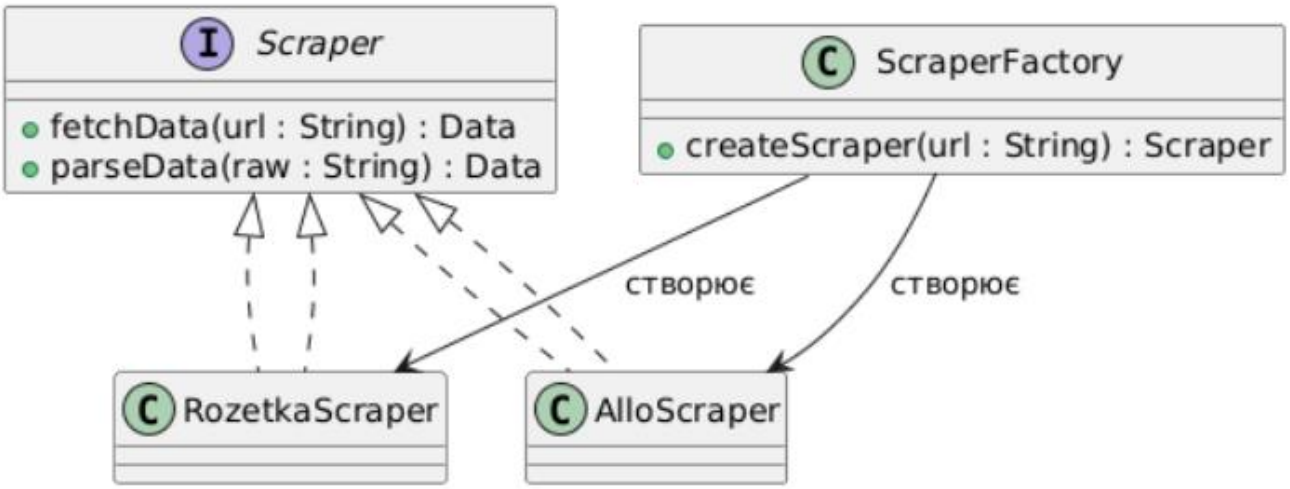


Рисунок 2.7 – UML-діаграма класів підсистеми збору даних

Діаграма ілюструє застосування патернів Strategy та Factory Method, які забезпечують гнучкість, розширюваність та ізоляцію логіки парсингу для різних джерел.

У верхній частині діаграми розміщено інтерфейс Scraper, який визначає загальний контракт для всіх реалізацій скреперів. Він містить два абстрактні методи:

- fetchData(url: String): Data — відповідає за завантаження HTML-сторінки або API-відповіді з вказаного URL;

- parseData(raw: String): Data — виконує вилучення структурованих даних із сирого HTML або JSON.

Інтерфейс реалізується конкретними класами RozetkaScraper та AlloScraper, які інкапсулюють специфічну логіку парсингу для відповідних платформ. Зв’язки типу <<implements>> вказують на те, що ці класи реалізують методи, визначені в інтерфейсі Scraper. Такий підхід дозволяє легко додавати нові реалізації без зміни існуючого коду, що відповідає принципу відкритості/закритості.

Клас ScraperFactory виконує роль фабрики, яка створює відповідний екземпляр скрапера на основі вхідного URL. Метод createScraper(url: String):

Scraper аналізує доменне ім’я або структуру URL і повертає реалізацію, яка найкраще підходить для обробки даних з цього джерела. Асоціативні зв’язки між ScraperFactory та конкретними реалізаціями (RozetkaScraper, AlloScraper) позначені як "створює", що відображає породжувальний характер взаємодії.

Загалом діаграма демонструє модульну та розширювану структуру підсистеми збору даних, де кожен компонент виконує чітко визначену роль, а взаємодія між ними реалізована через абстракції та фабричні механізми. Такий підхід забезпечує легкість підтримки, тестування та масштабування системи при додаванні нових джерел інформації.

2.2.4. Логіка конвеєра збору даних (Scraping Pipeline)

Ось покращений академічний варіант тексту з більш чіткою структурою, логічними зв’язками та стилістичною узгодженістю:

Процес web-скрепінгу в межах інтелектуальної системи моніторингу e-commerce платформ доцільно розглядати як багатоступеневий конвеєр обробки даних, де кожен етап виконує чітко визначену функцію. Такий підхід дозволяє формалізувати логіку збору інформації, забезпечити її узгодженість та створити основу для масштабування системи відповідно до зростання обсягів даних і кількості джерел.

На початковому етапі формується список цільових URL-адрес, що підлягають моніторингу. Джерелом таких адрес можуть бути конфігураційні файли, записи у базі даних або автоматично згенеровані посилання на основі структури каталогу товарів. Сформовані URL передаються до підсистеми завантаження, яка відповідає за встановлення HTTP-з’єднання, обробку запитів та отримання HTML-контенту відповідних сторінок.

Наступним етапом є аналіз отриманого контенту. Для обробки статичних сторінок застосовується бібліотека Jsoup, яка дозволяє побудувати DOM-дерево документа та здійснювати навігацію по його елементах. У випадку динамічного контенту, що генерується на стороні клієнта за допомогою JavaScript,

використовується Selenium WebDriver або headless-браузери, які емулюють поведінку користувача та забезпечують доступ до повністю згенерованої версії сторінки.

Після завантаження контенту виконується етап вилучення даних. На цьому рівні конкретна реалізація скрапера здійснює пошук цільових елементів у DOM-структурі та формує структурований об’єкт, який містить ключові атрибути товару — назву, ціну, наявність, рейтинг, відгуки тощо. Отримані дані передаються до модуля попередньої обробки, де проходять етапи очищення від шумів, нормалізації форматів та валідації на предмет коректності. Лише після цього інформація зберігається у базі даних для подальшої аналітики. Узагальнена схема конвеєра web-скрепінгу зображена на рисунку 2.8.

Рисунок 2.8 – Узагальнена схема конвеєра web-скрепінгу

2.2.5. Проєктування механізму багатопотокової обробки

Висока динаміка розвитку e-commerce платформ та значна кількість об’єктів моніторингу зумовлюють потребу у використанні багатопотокової моделі виконання. Основна мета впровадження паралельної обробки полягає у мінімізації загального часу збору даних шляхом одночасного виконання незалежних завдань, що дозволяє системі масштабуватися відповідно до зростання обсягів інформації.

У проєктованій системі багатопотоковість реалізується на основі стандартних засобів Java Concurrency API, які забезпечують високий рівень абстракції та безпечну роботу з потоками. Центральним компонентом є менеджер завдань, що використовує пул потоків із фіксованою кількістю робочих потоків. Такий підхід дозволяє:

- контролювати навантаження на систему;
- уникати надмірного споживання ресурсів;
- забезпечувати стабільність роботи при використанні ресурсомістких інструментів, таких як headless-браузери.

Логіка роботи організована за принципом черги завдань. Усі URL-адреси, які підлягають обробці, розміщуються у потокобезпечній структурі даних. Кожен вільний потік отримує адресу з черги, визначає відповідну реалізацію скрапера та виконує повний цикл збору даних: завантаження сторінки, парсинг контенту, вилучення ключових атрибутів. Після завершення обробки результат передається до підсистеми збереження, а потік повертається до пулу для виконання наступного завдання. Така організація забезпечує безперервність процесу та оптимальне використання доступних ресурсів.

Використання керованого пулу потоків має низку додаткових переваг:

- продуктивність: значний приріст швидкості у порівнянні з послідовною обробкою завдяки паралельному виконанню.

- стійкість до блокувань: система може обмежувати кількість одночасних запитів, знижуючи ризик блокування з боку цільових сайтів.
- гнучкість конфігурації: кількість потоків у пулі може змінюватися залежно від поточних умов — продуктивності сервера, швидкості мережі чи політики доступу до ресурсів.
- масштабованість: додавання нових джерел даних не потребує зміни архітектури, оскільки всі завдання обробляються уніфікованим механізмом черги та пулу потоків.

Таким чином, багатопотокова модель виконання у поєднанні з Java Concurrency API забезпечує ефективне використання обчислювальних ресурсів, підвищує продуктивність системи та гарантує її стійкість до змін середовища e-commerce. Це рішення відповідає сучасним вимогам до високонавантажених систем збору та обробки даних (рисунок 2.9).

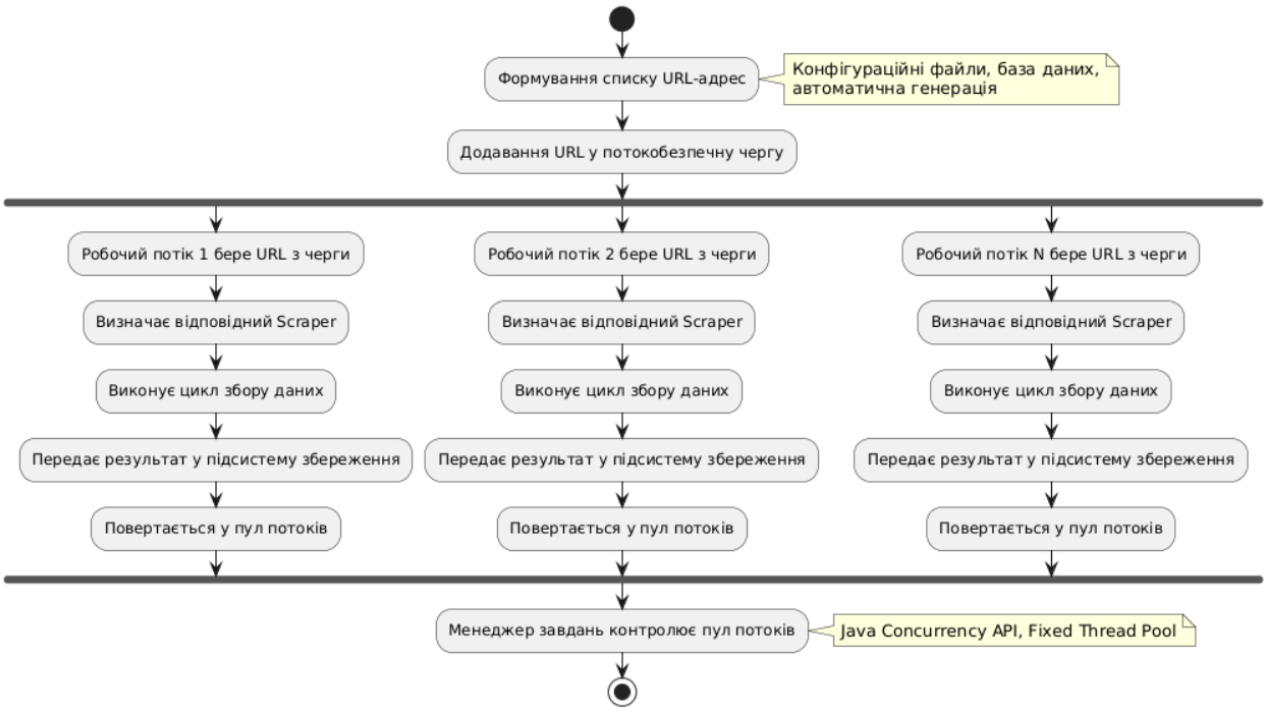


Рисунок 2.9 – Схема багатопотокової обробки завдань збору даних.

Діаграма демонструє багатопотокову модель виконання, де URL-адреси розміщуються у потокобезпечній черзі та обробляються кількома робочими

потоками паралельно. Менеджер завдань керує пулом потоків, забезпечуючи ефективне використання ресурсів і скорочення часу збору даних.

2.2.6. Проєктування механізмів обходу захисту e-commerce платформ

Сучасні e-commerce платформи активно впроваджують різноманітні технічні засоби захисту від автоматизованого доступу, що суттєво ускладнює процес збору даних. До найбільш поширених механізмів належать обмеження частоти запитів, аналіз поведінкових патернів користувачів, використання CAPTCHA, а також блокування підозрілих або надмірно активних IP-адрес. Такі заходи спрямовані на мінімізацію ризиків несанкціонованого вилучення інформації та захист комерційних інтересів платформ.

Для забезпечення стабільної роботи системи моніторингу в архітектурі передбачено окремий інфраструктурний модуль, відповідальний за мережеву взаємодію та адаптацію до умов доступу. Даний модуль реалізує комплекс механізмів, серед яких:

- ротація User-Agent для імітації різних типів клієнтів (браузери, мобільні пристрої).
- випадкові затримки між запитами, що дозволяють уникати підозрілої регулярності звернень.
- обробка помилок та повторні спроби виконання операцій із застосуванням експоненціального бекофу.
- використання проксі-серверів для розподілу навантаження та обходу блокувань IP-адрес.

Завдяки цим заходам система здатна імітувати поведінку реального користувача, знижуючи ймовірність виявлення та блокування з боку вебресурсів. Крім того, модуль підтримує гнучку конфігурацію параметрів доступу, що дозволяє адаптуватися до змін політики конкретних платформ.

Особливу роль у роботі підсистеми відіграє використання headless-браузерів (наприклад, Chrome Headless або Puppeteer), які виконують JavaScript-код сторінки та формують контент аналогічно звичайному браузеру. Це забезпечує можливість роботи з динамічними інтерфейсами, що активно застосовуються сучасними e-commerce платформами, та дозволяє отримувати дані, недоступні при використанні класичних HTTP-клієнтів. Headless-браузери також підтримують емуляцію дій користувача (клік, скролінг, введення тексту), що робить їх незамінними для обходу складних механізмів захисту та доступу до прихованих елементів сторінки (рисунок 2.10).

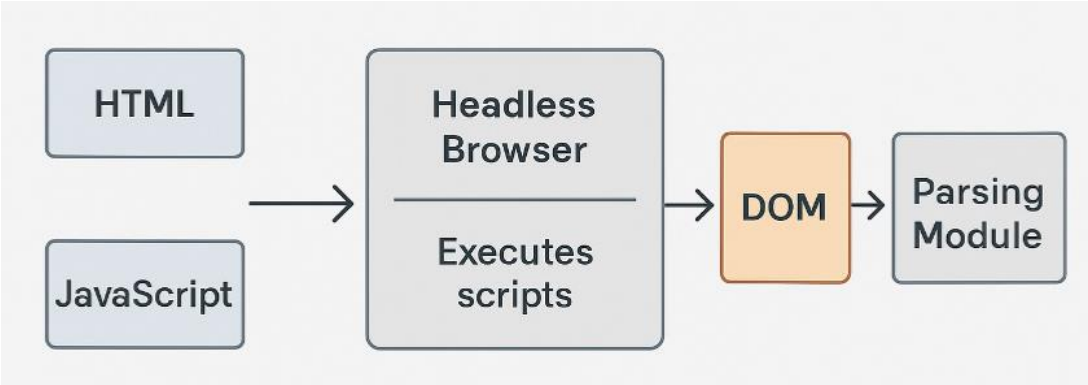


Рисунок 2.10 – Схема роботи headless-браузера у процесі web-скрепінгу

Рисунок ілюструє процес обробки вебконтенту за допомогою headless-браузера, який отримує HTML та JavaScript, виконує сценарії та формує повністю згенеровану DOM-структуру. Згенерований DOM передається до модуля парсингу для вилучення структурованих даних.

2.2.7. Проєктування моделі даних інтелектуальної системи моніторингу

Ефективність функціонування інтелектуальної системи моніторингу значною мірою залежить від якості проєктування моделі даних, оскільки саме структура збереження інформації визначає швидкість доступу до даних, точність аналітичних обчислень та здатність системи до масштабування. У контексті предметної області e-commerce модель даних повинна бути здатною обробляти як статичні характеристики товарів (назва, бренд, категорія), так і динамічні

параметри (ціна, наявність, рейтинг), а також зберігати історичні дані для аналізу ринкових тенденцій, сезонності та поведінки споживачів.

У проєктованій системі обрано реляційну модель даних, реалізовану на базі СУБД MySQL, що забезпечує баланс між продуктивністю, гнучкістю та стабільністю. Вибір MySQL обґрунтовується її зрілістю як технології, широкою підтримкою у Java-екосистемі, наявністю ефективних механізмів індексації, транзакційного контролю та можливістю виконання складних аналітичних запитів за допомогою SQL. Структура бази даних спроєктована відповідно до принципів нормалізації, зокрема третьої нормальної форми (3NF), що дозволяє мінімізувати надлишковість, уникнути аномалій оновлення та забезпечити логічну цілісність даних.

Центральною сутністю моделі є товар, який описується набором атрибутів, що змінюються рідко або залишаються сталими протягом тривалого часу (наприклад, артикул, назва, категорія, виробник). Динамічні параметри — такі як ціна, наявність, рейтинг — винесені в окремі таблиці, пов’язані з основною сутністю відношенням «один-до-багатьох». Це дозволяє накопичувати історію змін, виконувати часовий аналіз, будувати графіки динаміки та виявляти тренди. Для кожного запису зберігається мітка часу, що забезпечує можливість ретроспективного аналізу.

Окрему увагу приділено збереженню текстових даних, зокрема відгуків користувачів, які є важливим джерелом інформації для оцінки якості товару та поведінки покупців. З урахуванням подальшого застосування методів обробки природної мови (NLP), у структурі таблиці передбачено зберігання як сирого тексту відгуку, так і результатів інтелектуального аналізу, зокрема оцінки тональності, класифікації за темами, виявлення ключових слів. Це дозволяє інтегрувати модулі семантичного аналізу та формувати узагальнені метрики споживчої оцінки.

Загалом, модель даних орієнтована на гнучкість, розширюваність та аналітичну придатність, що забезпечує її відповідність вимогам високонавантаженої системи моніторингу та дозволяє ефективно працювати з

великими обсягами гетерогенних даних у динамічному середовищі e-commerce (рисунок 2.11).

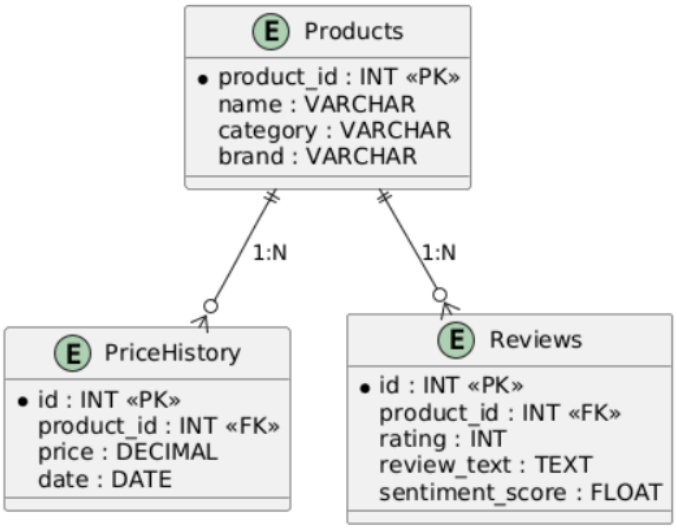


Рисунок 2.11 – ER-діаграма моделі даних системи моніторингу

Діаграма показує реляційну модель даних, де таблиця Products є центральною сутністю з первинним ключем product_id. Вона має зв’язки «один-до-багатьох» із таблицями PriceHistory та Reviews, які містять зовнішній ключ product_id для зберігання історії цін та відгуків користувачів.

2.2.8. Проєктування шару доступу до даних (DAO та Repository)

Для забезпечення чіткого розмежування відповідальності між бізнес-логікою та механізмами збереження інформації в проєктованій системі застосовано шаблон проєктування Data Access Object (DAO). Використання цього патерна дозволяє інкапсулювати всі операції доступу до бази даних в окремому шарі, що забезпечує гнучкість архітектури, зменшує залежність ядра системи від конкретної реалізації СУБД та спрощує процес тестування й модифікації. DAO виступає своєрідним «посередником» між бізнес-логікою та рівнем збереження даних, надаючи уніфікований інтерфейс для виконання CRUD-операцій (Create, Read, Update, Delete).

У межах проєктованої архітектури доступ до даних реалізується через набір репозиторіїв, кожен з яких відповідає за роботу з окремою сутністю предметної області. Наприклад:

- ProductRepository — забезпечує операції створення, пошуку та оновлення записів у таблиці *Products*;
- PriceHistoryRepository — відповідає за збереження та вибірку часових рядів, що відображають динаміку зміни цін;
- ReviewRepository — реалізує доступ до відгуків користувачів, включно з можливістю вибірки даних для подальшого NLP-аналізу.

Такий підхід дозволяє ізолювати специфіку роботи з кожною сутністю та забезпечує масштабованість системи при розширенні моделі даних.

Для реалізації взаємодії з базою даних у Java застосовується стандартний механізм JDBC або легковаговий ORM-підхід (наприклад, MyBatis). Це рішення поєднує контроль над SQL-запитами з перевагами об’єктного представлення даних, що спрощує розробку та підтримку коду. Всі запити до бази даних виконуються у межах транзакцій, що гарантує цілісність інформації та узгодженість стану системи навіть у випадку помилок або збоїв під час виконання операцій. Крім того, транзакційний контроль дозволяє реалізувати механізми відкату (rollback), що є критично важливим для високонавантажених систем моніторингу.

Виходячи з вище описаного, застосування шаблону DAO у поєднанні з репозиторіями та транзакційним контролем формує надійний та гнучкий шар доступу до даних, який забезпечує стабільність роботи системи, її розширюваність та відповідність принципам об’єктно-орієнтованого проєктування.

Для забезпечення ефективної взаємодії між різними шарами системи у проєктованій архітектурі застосовується шаблон Data Transfer Object (DTO). DTO-об’єкти являють собою прості Java-класи, що містять лише поля даних та гетери/сетери для їх доступу, не включаючи бізнес-логіки чи методів обробки.

Такий підхід дозволяє чітко розмежувати відповідальність між рівнем збереження даних та бізнес-логікою, уникнути прямої залежності від внутрішніх моделей бази даних та забезпечити гнучкість при зміні структури сховища.

У процесі збору інформації скрапер формує DTO-об’єкт, який містить ключові атрибути товару: назву, ціну, наявність, рейтинг, відгуки тощо. Цей об’єкт передається до шару бізнес-логіки, де виконується подальша обробка — нормалізація форматів, очищення від шумів, валідація та підготовка до збереження у базі даних. Завдяки цьому забезпечується уніфікація формату даних незалежно від джерела інформації, що значно спрощує інтеграцію аналітичних модулів, які працюють з єдиною структурою даних.

Використання DTO має низку переваг:

- ізоляція шарів — бізнес-логіка не залежить від конкретної реалізації моделі даних у СУБД;
- гнучкість інтеграції — зміни у структурі бази даних не потребують модифікації аналітичних модулів, достатньо оновити DTO;
- зручність тестування — DTO-об’єкти легко використовувати у модульних тестах як фікстури для перевірки бізнес-логіки;
- масштабованість — уніфікований формат даних дозволяє без проблем додавати нові джерела інформації та розширювати функціональність системи.

2.2.9. Нефункціональні вимоги до архітектури системи

Окрім реалізації функціональних можливостей, архітектура інтелектуальної системи моніторингу повинна відповідати низці нефункціональних вимог, які визначають її придатність до практичного використання та довгострокової експлуатації. До ключових вимог належать продуктивність, масштабованість, надійність, підтримуваність та портативність.

Забезпечення продуктивності забезпечується використання багатопотокової моделі виконання забезпечує високу швидкість обробки даних навіть при

значному обсязі інформації. Завдяки оптимізації потоків та контролю навантаження система здатна працювати у режимі реального часу, що критично важливо для моніторингу динамічних e-commerce платформ.

Масштабованість забезпечується за рахунок використання модульної структури. Модульна структура архітектури та чітке розмежування відповідальності між компонентами дозволяють безболісно розширювати систему. У разі необхідності додавання нових джерел даних або аналітичних модулів достатньо інтегрувати нові компоненти без істотних змін існуючого коду, що відповідає принципам відкритості/закритості.

Надійність. Система повинна гарантувати цілісність даних та стійкість до збоїв. Для цього застосовуються транзакційні механізми, обробка винятків, повторні спроби виконання операцій та журналювання. Такий підхід мінімізує ризик втрати інформації та забезпечує стабільність роботи навіть у випадку помилок на рівні окремих модулів.

Підтримуваність. Завдяки модульності та використанню шаблонів проєктування (DAO, DTO, Factory Method, Strategy) система легко адаптується до змін бізнес-логіки та технологічних вимог. Чітка ізоляція шарів спрощує тестування, налагодження та модернізацію програмного забезпечення.

Портативність. Кросплатформеність Java та використання віртуальної машини JVM дозволяють розгортати систему на різних операційних системах без необхідності адаптації програмного коду. Це є важливою перевагою для практичного впровадження у різних інфраструктурних середовищах, включно з хмарними платформами.

Таким чином, дотримання нефункціональних вимог забезпечує не лише коректність роботи системи, але й її довготривалу життєздатність, можливість масштабування та адаптації до змін ринку, що робить архітектуру придатною для використання у реальних умовах високонавантажених e-commerce середовищ.

В даному підрозділі було здійснено детальне проєктування архітектури інтелектуальної системи моніторингу e-commerce платформ. Запропонована багаторівнева модульна структура забезпечує гнучкість, масштабованість та

стійкість системи до змін зовнішнього середовища, що є критично важливим для роботи у динамічних умовах електронної комерції.

Архітектурні рішення для підсистеми збору даних базуються на використанні патернів Strategy та Factory Method, що дозволяє легко інтегрувати нові джерела інформації та підтримувати принцип відкритості/закритості. Реалізація багатопотокової обробки за допомогою Java Concurrency API забезпечує високу продуктивність системи при роботі з великими обсягами даних та оптимальне використання ресурсів.

Спроєктована модель даних та механізми доступу до неї дозволяють зберігати як статичні характеристики товарів, так і динамічні параметри з історією змін, включно з відгуками користувачів. Це створює основу для подальшого інтелектуального аналізу та формування аналітичних висновків.

2.3. Розробка модулів збору, обробки, зберігання та підготовки даних для подальшої аналітики і візуалізації

Етап детального проєктування модулів інтелектуальної системи моніторингу виступає логічним продовженням архітектурних рішень, сформульованих у попередньому підрозділі, та водночас є ключовою перехідною ланкою між концептуальним рівнем проєктування і безпосередньою програмною реалізацією. На цьому етапі здійснюється не лише формалізація алгоритмів, але й визначення внутрішніх механізмів взаємодії компонентів, опис інтерфейсів та протоколів обміну даними, а також деталізація логіки обробки інформації на кожній стадії її життєвого циклу — від моменту надходження з зовнішніх джерел до збереження у базі та підготовки для аналітичної обробки. Такий рівень деталізації дозволяє забезпечити узгодженість між окремими модулями, підвищити надійність системи та створити основу для її масштабованої реалізації у реальних умовах високонавантаженого середовища e-commerce.

Особливістю задачі моніторингу e-commerce платформ є необхідність роботи з великими обсягами динамічної, слабо структурованої та часто

нестабільної інформації, що надходить із різнорідних веб-джерел у режимі реального часу. Така специфіка даних зумовлює підвищені вимоги до надійності системи, її здатності до масштабування при зростанні кількості джерел та обсягів інформації, а також до узгодженості роботи окремих програмних модулів. Важливим завданням є забезпечення безперервності процесу збору та обробки даних навіть у випадку збоїв або зміни структури веб-ресурсів. Саме тому у даному підрозділі розглядаються не лише функціональні можливості компонентів, але й алгоритмічні підходи до організації їх взаємодії, синхронізації потоків, управління конкурентним доступом та гарантування цілісності даних на всіх етапах життєвого циклу.

У межах цього підрозділу здійснюється детальне проєктування ключових модулів інтелектуальної системи моніторингу, кожен з яких виконує специфічну функцію та формує цілісний технологічний ланцюг обробки даних. До складу системи входять: модуль збору даних, що відповідає за інтеграцію з веб-джерелами та отримання первинної інформації; модуль обробки та парсингу, який забезпечує структуризацію та нормалізацію даних; підсистема зберігання історичних даних, орієнтована на накопичення часових рядів та текстових відгуків; а також модуль підготовки інформації для аналітики та візуалізації, що формує узгоджені набори даних для подальшого застосування методів інтелектуального аналізу та побудови графічних інтерфейсів. Кожен компонент реалізується у вигляді ізольованого модуля, що відповідає принципам модульності, єдиного обов’язку (Single Responsibility Principle) та низької зв’язаності (Low Coupling), що у сукупності забезпечує гнучкість архітектури, простоту супроводу та можливість масштабування системи без істотних змін у її структурі.

2.3.1. Алгоритмічна реалізація модуля збору даних (Collector Module)

Модуль збору даних є вхідною точкою всієї інтелектуальної системи моніторингу та виконує критично важливу функцію — отримання первинної

інформації з e-commerce платформ. На цьому етапі формується потік так званих «сирих» даних у вигляді HTML-документів та динамічно згенерованого контенту, що надходить від веб-ресурсів. Подальша обробка цих даних здійснюється семантичними та аналітичними модулями, тому саме ефективність роботи Collector Module визначає актуальність, повноту та якість усіх наступних результатів аналізу.

Алгоритмічна реалізація модуля передбачає використання web-скрепінгу з інтеграцією headless-браузерів для коректної обробки JavaScript-залежних сторінок, а також застосування багатопотокових механізмів для паралельного збору інформації з великої кількості джерел. Collector Module включає механізми керування чергами запитів, обробки помилок та повторних спроб у випадку недоступності ресурсів, що забезпечує стійкість системи до збоїв та змін у структурі веб-сторінок.

Таким чином, модуль збору даних виступає фундаментом усієї системи, оскільки саме він визначає якість вхідного інформаційного потоку, від якого залежить точність аналітичних висновків та ефективність подальшої візуалізації результатів.

З огляду на те, що цільові веб-ресурси характеризуються високою динамікою контенту, частими змінами структури сторінок та застосуванням різноманітних механізмів захисту від автоматизованого доступу, модуль збору даних повинен забезпечувати багатопотокове виконання, гнучку адаптацію до змін мережевого середовища та коректну обробку помилок. Для досягнення цих вимог у проєктованій системі застосовано алгоритмічний підхід на основі черг завдань та керованих пулів потоків, що дозволяє ефективно розподіляти навантаження між паралельними процесами, уникати блокувань та забезпечувати стабільність роботи навіть при значному обсязі запитів. Така організація дає змогу реалізувати механізми повторних спроб у випадку збоїв, балансувати пріоритети завдань та гарантувати узгодженість даних, що надходять із різних джерел. У результаті Collector Module набуває властивостей масштабованості,

відмовостійкості та продуктивності, необхідних для роботи у високонавантажених умовах e-commerce середовища.

Основою алгоритму збору даних є використання потокобезпечної структури BlockingQueue, що надається стандартною бібліотекою Java Concurrency. Ця структура забезпечує синхронізований доступ до черги завдань у багатопотоковому середовищі, запобігаючи конфліктам при одночасному додаванні та вилученні елементів. У черзі зберігається список URL-адрес, які підлягають обробці, причому кожен елемент інкапсулює не лише посилання на веб-сторінку, але й додаткову службову інформацію: час останнього сканування, кількість попередніх невдалих спроб доступу, а також пріоритет виконання. Така організація дозволяє реалізувати механізми адаптивного повторного опитування, балансування навантаження між потоками та динамічне коригування частоти звернень до ресурсів. У результаті BlockingQueue виступає центральним елементом алгоритму, що гарантує узгодженість роботи Collector Module та стійкість системи до збоїв у процесі збору даних.

Процес збору даних реалізується у вигляді множини робочих потоків (Worker Threads), які функціонують за класичною моделлю «виробник–споживач» (Producer–Consumer). У цій схемі планувальник системи виконує роль виробника, формуючи та наповнюючи чергу новими завданнями, тоді як керований пул потоків виступає споживачем, що асинхронно обробляє ці завдання у паралельному режимі. Такий підхід забезпечує оптимальне використання обчислювальних ресурсів, дозволяє уникати простоїв та перевантаження окремих потоків, а також гарантує рівномірний розподіл навантаження між робочими процесами. Завдяки цьому система зберігає масштабованість при збільшенні кількості джерел даних, підтримує високу продуктивність та демонструє стійкість до збоїв, що є критично важливим для роботи у високонавантажених умовах e-commerce середовища.

Алгоритм роботи потоку-воркера. Кожен робочий потік функціонує за уніфікованим алгоритмом, що складається з послідовності логічно пов’язаних етапів. Після отримання чергового URL-адресу з черги завдань потік виконує

перевірку часу попереднього звернення до цього ресурсу з метою дотримання політики ввічливості (Politeness Policy). Такий механізм запобігає надмірному навантаженню на сервер цільового сайту та знижує ризик блокування доступу. У випадку, якщо інтервал між запитами є занадто коротким, потік здійснює адаптивну затримку перед виконанням HTTP-запиту, що дозволяє узгодити частоту звернень із вимогами конкретного ресурсу. Дотримання цього принципу забезпечує стабільність процесу збору даних, підвищує надійність роботи системи та сприяє коректному функціонуванню модуля збору в умовах багатопотокового середовища.

На наступному етапі робочий потік звертається до сервісу управління проксі-серверами, який повертає валідну проксі-адресу для виконання запиту. Використання проксі дозволяє рівномірно розподіляти мережеве навантаження між різними точками виходу в Інтернет, а також значно знижує ризик блокування з боку e-commerce платформи у випадку надмірної кількості звернень з однієї IP-адреси. Після отримання проксі та встановлення з’єднання потік формує та виконує HTTP-запит, після чого здійснює аналіз коду відповіді сервера. На цьому етапі визначається коректність виконання запиту: у разі отримання успішного статусу (наприклад, 200 OK) дані передаються на подальший етап парсингу, тоді як при виникненні помилок (наприклад, 403 Forbidden, 404 Not Found, 500 Internal Server Error) система не припиняє роботу, а коректно обробляє виняткову ситуацію. Завдання повертається в кінець черги із збільшеним лічильником спроб, що дозволяє повторити обробку через певний проміжок часу.

Такий алгоритм (рисунок 2.12) забезпечує високу стійкість системи до мережевих збоїв та непередбачуваних змін у структурі веб-ресурсів, що особливо важливо для роботи з динамічними e-commerce платформами. Завдяки використанню потокобезпечних структур даних, механізмів повторних спроб та адаптивного управління проксі-серверами система зберігає безперервність процесу збору інформації навіть у нестабільному веб-середовищі. Це дозволяє мінімізувати втрати даних, підтримувати актуальність інформаційного потоку та гарантувати узгодженість результатів на всіх етапах життєвого циклу даних. У

результаті Collector Module виступає надійним фундаментом, що забезпечує продуктивність та відмовостійкість всієї інтелектуальної системи моніторингу.

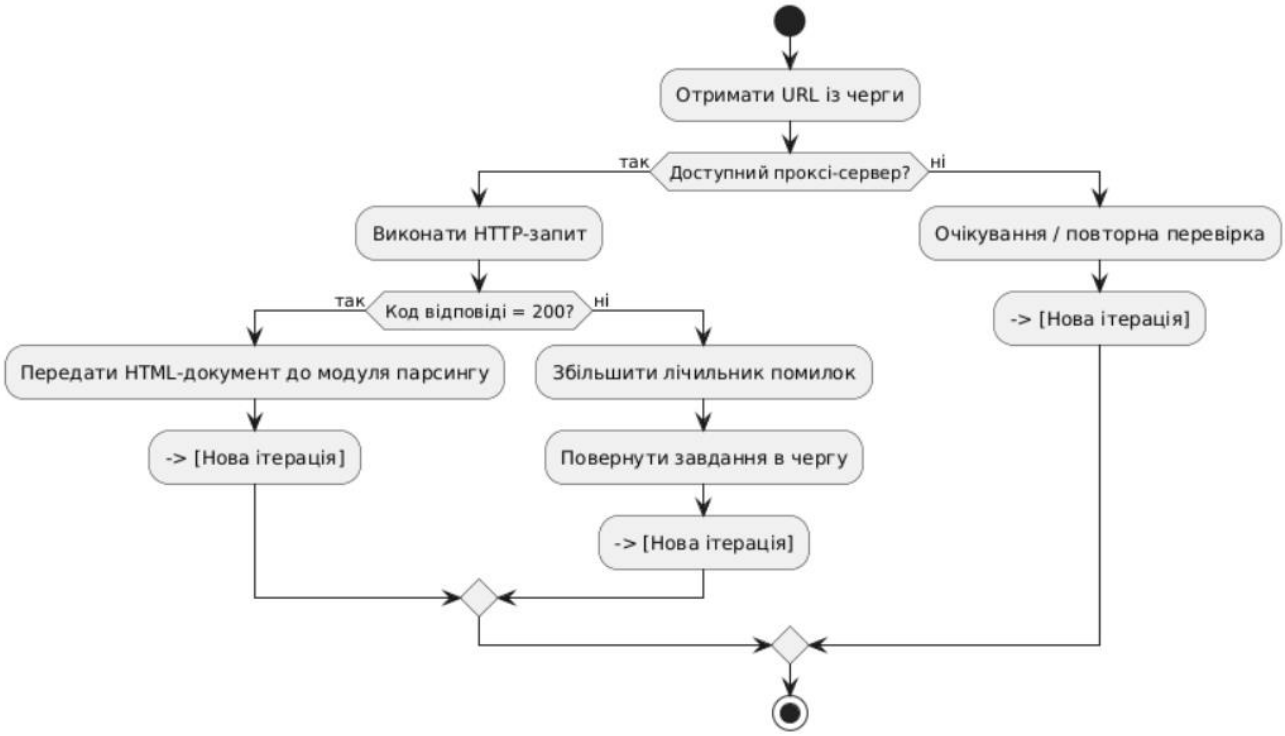


Рисунок 2.12 – Блок-схема алгоритму роботи модуля збору даних

Блок-схема відображає послідовність роботи модуля збору даних: від отримання URL-адреси з черги до передачі HTML-документа в модуль парсингу. Вона демонструє механізми перевірки доступності проксі, обробки кодів відповіді сервера та повторного додавання завдань у чергу у випадку помилок.

2.3.2. Розробка модуля обробки та парсингу даних (Parsing & Processing Engine)

Модуль обробки та парсингу виконує ключову функцію трансформації неструктурованої інформації, отриманої з веб-ресурсів, у формалізований вигляд, придатний для подальшого зберігання, аналітичної обробки та візуалізації. На відміну від етапу збору, де основним завданням є стабільне отримання HTML-контенту, етап парсингу орієнтований на семантичний аналіз структури веб-сторінок та коректне вилучення бізнес-значущих атрибутів, таких як назви товарів, ціни, характеристики, рейтинги та відгуки користувачів. Для цього

застосовуються алгоритми розпізнавання DOM-структури, регулярні вирази, а також спеціалізовані бібліотеки для роботи з HTML та XML. Важливим аспектом є забезпечення гнучкості алгоритмів, які повинні адаптуватися до змін у верстці та динамічному контенті e-commerce платформ. Таким чином, модуль парсингу виступає центральною ланкою, що перетворює «сирі» дані у структуровані інформаційні об’єкти, які можуть бути інтегровані у систему зберігання та використані для подальшої аналітики.

Особливість e-commerce платформ полягає у відсутності єдиного стандарту представлення даних: різні онлайн-магазини застосовують власні HTML-шаблони, унікальні класи CSS, багаторівневі вкладені структури та динамічно сформовані елементи, що ускладнює процес автоматизованого вилучення інформації. Тому модуль парсингу має бути гнучким, розширюваним та водночас ізольованим від інших компонентів системи, аби зміни у структурі окремих сайтів не впливали на стабільність роботи всієї системи. У проєктованій архітектурі це досягається шляхом використання об’єктно-орієнтованого підходу, застосування шаблонів проєктування (зокрема Strategy та Factory Method) та чіткого поділу відповідальності між класами. Такий підхід дозволяє легко додавати нові парсери для різних форматів сторінок, повторно використовувати існуючі компоненти та забезпечувати високу підтримуваність системи. У результаті модуль парсингу виступає універсальним інструментом, здатним адаптуватися до різноманітних форматів даних і гарантувати коректність їх трансформації у структурований вигляд.

Модуль парсингу реалізується як окрема підсистема, що взаємодіє з модулем збору даних через чітко визначений інтерфейс, забезпечуючи незалежність та узгодженість роботи компонентів. Вхідними даними для цього модуля є HTML-документ разом із супровідними метаданими запиту (URL-адреса, джерело отримання, час сканування, кількість попередніх спроб доступу). На основі цих даних здійснюється семантичний аналіз структури веб-сторінки та вилучення бізнес-значущих атрибутів. Вихідними результатами роботи є типізовані об’єкти передачі даних (Data Transfer Objects, DTO), які інкапсулюють

ключову інформацію про товар: назву, ціну, характеристики, рейтинг, кількість відгуків та інші параметри. Такий підхід забезпечує стандартизоване представлення даних, спрощує їх інтеграцію у підсистему зберігання та створює основу для подальшої аналітики й візуалізації.

Для досягнення слабкої зв’язності між логікою парсингу та структурою конкретного сайту застосовується принцип конфігураційної ізоляції. Це означає, що всі знання про HTML-верстку цільового ресурсу концентруються або в окремому класі-парсері, або у спеціальному конфігураційному профілі, який містить набір CSS-селекторів та правил вилучення даних. Такий підхід дозволяє чітко відокремити ядро системи від специфічних деталей реалізації, забезпечуючи гнучкість та розширюваність модуля. У разі зміни верстки сайту модифікації торкаються лише відповідного парсера чи профілю, не впливаючи на роботу інших компонентів. Це значно спрощує підтримку системи, знижує ризик помилок та забезпечує можливість швидкої адаптації до нових форматів даних без втручання у базову архітектуру.

Процес парсингу складається з послідовності етапів, кожен з яких виконує чітко визначену функцію у трансформації «сирих» даних у структуровану форму. На першому етапі за допомогою бібліотеки Jsoup HTML-документ перетворюється у DOM-дерево, що забезпечує зручну навігацію по елементах сторінки. Це дозволяє застосовувати CSS-селектори або XPath-подібні запити для точного доступу до потрібних вузлів та атрибутів. Такий підхід робить можливим ефективне вилучення бізнес-значущих даних навіть із складних та багаторівневих HTML-структур, а також створює основу для наступних етапів — нормалізації, валідації та формування типізованих об’єктів передачі даних (DTO).

На наступному етапі здійснюється вилучення ключових атрибутів товару, зокрема його назви, ціни, статусу наявності, унікального ідентифікатора, а також додаткових характеристик (наприклад, колір, розмір, бренд, рейтинг чи кількість відгуків). Отримані текстові значення у більшості випадків містять службові символи, пробіли, одиниці вимірювання або валютні позначення, що унеможливлює їх безпосереднє використання в обчисленнях чи аналітичних

операціях. Тому на цьому етапі застосовуються процедури очищення та нормалізації даних: видалення зайвих символів, приведення числових значень до єдиного формату, уніфікація валютних позначень та стандартизація одиниць вимірювання.

Для усунення проблеми некоректного представлення атрибутів застосовується етап нормалізації даних. Зокрема, при обробці цін використовується механізм регулярних виразів, який дозволяє видалити всі нечислові символи, окрім десяткового роздільника. Це забезпечує отримання чистого числового значення, придатного для подальших обчислень. Отримане значення перетворюється у числовий тип BigDecimal, що гарантує високу точність фінансових розрахунків та запобігає накопиченню похибок, характерних для типів із плаваючою комою (наприклад, float чи double). Такий підхід дозволяє стандартизувати цінові дані, уніфікувати їх формат та забезпечити коректність при інтеграції у сховище даних, використанні у бізнес-аналітиці та побудові фінансових звітів.

Окрему увагу приділено обробці статусу наявності товару, оскільки різні e-commerce платформи використовують власні текстові маркери для позначення стану («Купити», «Немає в наявності», «Очікується», «Передзамовлення» тощо). Для усунення цієї різнорідності у системі реалізовано механізм зіставлення текстових значень із внутрішнім переліком станів, що дозволяє стандартизувати інформацію незалежно від джерела. Технічно це реалізується за допомогою перелічуваного типу (enum), який визначає обмежений набір допустимих станів, наприклад: AVAILABLE, OUT_OF_STOCK, EXPECTED, PREORDER. Такий підхід забезпечує уніфікацію даних, спрощує їх подальшу обробку та інтеграцію у бізнес-логіку системи, а також дозволяє уникнути неоднозначностей при аналітичних розрахунках і візуалізації.

Результатом роботи модуля парсингу є створення об’єкта типу ProductDTO, який виступає контейнером для структурованих даних, отриманих із веб-ресурсів. Цей об’єкт не містить бізнес-логіки та використовується виключно для передачі інформації між модулями системи, забезпечуючи чітке розділення

відповідальності. Такий підхід спрощує процес тестування, підвищує читабельність коду та зменшує ризик виникнення побічних ефектів при модифікації логіки обробки. Крім того, використання DTO сприяє стандартизації даних, полегшує інтеграцію з іншими підсистемами та створює основу для подальшої аналітики й візуалізації.

Формування DTO супроводжується обов'язковим етапом валідації, під час якого перевіряється повнота та коректність отриманих атрибутів. Якщо обов'язкові поля (наприклад, назва товару, ціна, ідентифікатор) відсутні або містять некоректні значення, система автоматично створює запис у журналі помилок та передає інформацію про збій у модуль моніторингу. Це дозволяє оперативно виявляти проблеми, пов'язані зі змінами структури сайтів, ще на ранніх етапах роботи, а також забезпечує прозорість і контрольованість процесу збору даних. Завдяки такому підходу система зберігає стійкість до зовнішніх змін, а розробники отримують можливість швидко реагувати на нові формати верстки та мінімізувати ризик втрати критично важливої інформації. Схема трансформації даних від HTML-документа до DTO-об'єкта приведена на рисунку 2.13.

Рисунок 2.13 – Схема трансформації даних від HTML-документа до DTO-об'єкта

Схема показує послідовний процес трансформації даних: від сирого HTML-документа через парсинг, вилучення, очищення та валідацію. Завершальним результатом є формування об'єкта ProductDTO, який містить структуровані атрибути товару для подальшої обробки.

В результаті реалізації модуля парсингу система отримує уніфікований потік структурованих даних, незалежний від джерела їх походження. Це створює надійне підґрунтя для подальшого етапу — збереження інформації та формування історичних зрізів, що розглядається у наступному підрозділі.

2.3.3. Організація модуля зберігання даних та управління історією змін (Storage Layer)

Модуль зберігання даних виступає центральною ланкою між етапами збору інформації та її аналітичного опрацювання. Його основне призначення полягає не лише у фіксації поточного стану товарів, а й у забезпеченні довготривалого накопичення історичних даних, що є критично важливим для аналізу тенденцій, прогнозування змін та побудови звітів. У контексті моніторингу e-commerce платформ особливу роль відіграє коректне управління змінами, адже ціни, статус наявності та інші характеристики товарів можуть змінюватися кілька разів протягом доби. Завдяки цьому модуль зберігання забезпечує повноцінну картину динаміки ринку, дозволяючи системі не лише відображати актуальні дані, але й аналізувати їх у часовому розрізі для виявлення закономірностей та формування стратегічних висновків.

При проєктуванні модуля зберігання було обрано реляційну модель даних, яка забезпечує високий рівень структурованості, підтримку цілісності та можливість виконання складних аналітичних запитів. Взаємодія з базою даних реалізується через шар доступу до даних (Data Access Layer), побудований відповідно до патерну DAO (Data Access Object). Такий підхід дозволяє ізолювати SQL-логіку від бізнес-логіки системи, що значно підвищує гнучкість архітектури. У результаті стає можливим безболісно змінювати СУБД, оптимізувати запити або впроваджувати нові механізми кешування без впливу на інші модулі системи. Це забезпечує масштабованість, підтримуваність та адаптивність рішення, що є критично важливим для довготривалого моніторингу e-commerce платформ.

Однією з ключових задач модуля зберігання є коректна ідентифікація товарів у процесі повторних сканувань. У проєктованій системі унікальність кожного товару визначається на основі його URL-адреси або внутрішнього ідентифікатора платформи (SKU), якщо він доступний. Такий механізм дозволяє уникнути дублювання записів у таблиці товарів та забезпечує коректний зв’язок

між статичними характеристиками (наприклад, назва, категорія, виробник) та динамічними атрибутами (ціна, статус наявності, рейтинг). Завдяки цьому система зберігає цілісність даних, підтримує узгодженість історичних записів і створює основу для подальшої аналітики, прогнозування та побудови звітів.

Після надходження об’єкта ProductDTO модуль зберігання виконує перевірку на наявність відповідного запису у таблиці products. Якщо товар відсутній, створюється новий запис із фіксацією базових атрибутів (ідентифікатор, URL, назва, початкова категорія). У випадку, якщо товар уже існує в системі, відбувається часткове оновлення лише тих полів, що можуть змінюватися з часом, наприклад, категорії або назви, якщо вони були скориговані на стороні продавця. Такий підхід дозволяє підтримувати актуальність даних, уникати дублювання записів та зберігати історичну послідовність змін, що є критично важливим для аналітики та прогнозування.

Для відображення динаміки змін у системі використовується окрема таблиця історії цін, у якій кожен запис відповідає конкретному моменту часу. Такий підхід дозволяє зберігати повну хронологію змін без втрати попередніх значень. Водночас безконтрольне додавання записів при кожному скануванні призвело б до надмірного зростання обсягу бази даних.

З метою оптимізації роботи сховища даних у системі реалізовано алгоритм умовного збереження (upsert-логіку). Перед додаванням нового запису система порівнює отримане значення ціни та статусу наявності з останнім зафіксованим станом у таблиці історії. Якщо значення залишаються незмінними, новий запис не створюється — оновлюється лише поле часу останньої перевірки, що дозволяє зберігати актуальність без надмірного дублювання. Якщо ж виявлено зміну, система додає новий запис, тим самим фіксуючи момент переходу до нового стану. Такий підхід забезпечує баланс між повнотою історичних даних та ефективністю використання ресурсів, зменшує обсяг бази та водночас гарантує точність відображення динаміки змін.

Такий підхід дозволяє суттєво зменшити кількість надлишкових записів у базі даних і водночас зберегти всі значущі події, що мають вплив на аналітику.

Завдяки цьому система фіксує лише ті зміни, які дійсно важливі для бізнес-аналізу, наприклад, різку зміну ціни або тривалу відсутність товару на складі. У подальшому ці події можуть бути легко ідентифіковані при аналітичних запитах, що забезпечує точне відображення динаміки ринку та створює основу для прогнозування тенденцій і формування стратегічних рішень.

Приклад реалізації upsert-логіки для таблиці історії цін у різних СУБД:

```
INSERT INTO price_history (product_id, price, availability, timestamp, last_checked)
VALUES (:product_id, :price, :availability, NOW(), NOW())
ON CONFLICT (product_id) DO UPDATE
SET last_checked = EXCLUDED.last_checked
WHERE price_history.price = EXCLUDED.price
AND price_history.availability = EXCLUDED.availability;
```

Якщо для товару вже є запис із тією ж ціною та статусом наявності, оновлюється лише поле last_checked.

Якщо значення відрізняються — створюється новий запис.

Оскільки процес збереження даних включає декілька взаємопов’язаних операцій — оновлення товару, додавання запису в історію цін та збереження відгуків, — критично важливо гарантувати їх атомарність. Для цього у модулі зберігання застосовується механізм транзакцій, який забезпечує виконання всіх дій у межах єдиного логічного блоку. Якщо хоча б одна з операцій завершується помилкою, транзакція відкочується, і база даних повертається до попереднього узгодженого стану. Таким чином, усі операції, що стосуються одного результату сканування, виконуються в межах єдиної транзакції, що гарантує цілісність даних, запобігає появі частково збережених результатів та забезпечує надійність системи при високих навантаженнях.

У разі виникнення помилки на будь-якому етапі виконання транзакція скасовується (rollback), що запобігає появі частково збережених або неконсистентних даних у базі. Такий підхід особливо важливий у багатопотоковому середовищі, де одночасно можуть виконуватися десятки

операцій запису. Використання транзакцій гарантує, що всі взаємопов’язані дії — оновлення товару, додавання запису в історію цін, збереження відгуків — виконуються як єдиний атомарний блок. Це забезпечує узгодженість даних, підвищує надійність системи та дозволяє уникнути ситуацій, коли частина інформації була збережена, а інша втрачена через помилку (рисунок 2.14).

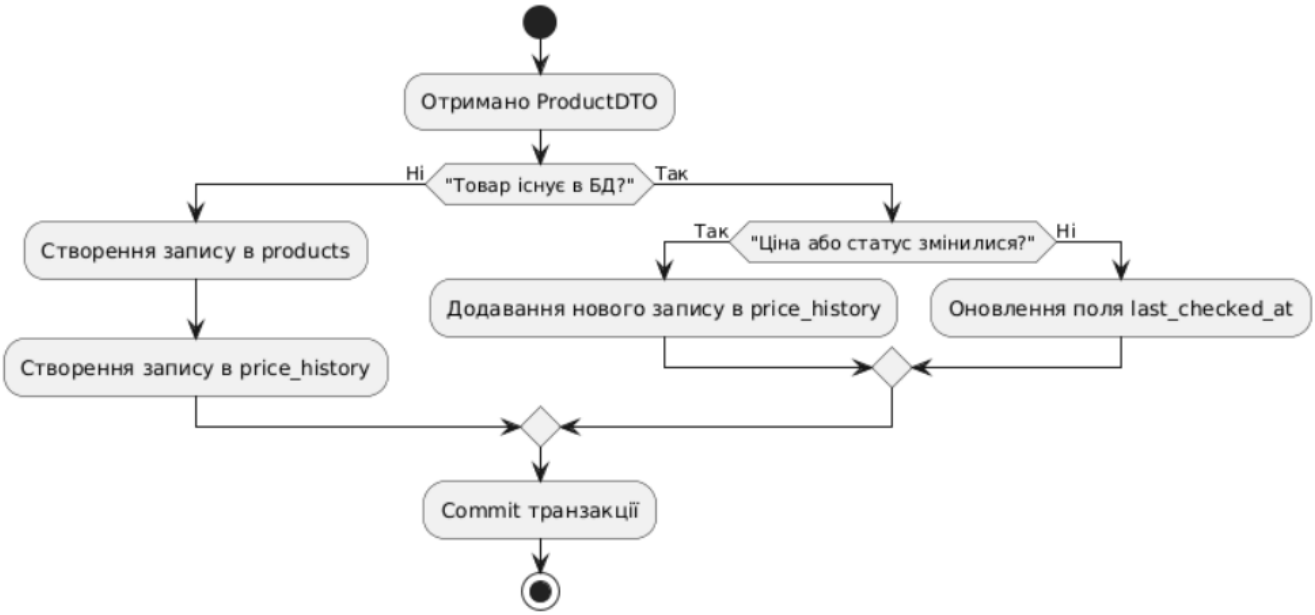


Рисунок 2.14 – Алгоритм збереження даних з перевіркою змін (Upsert Logic)

У підсумку, модуль зберігання забезпечує надійне та ефективне накопичення даних, формуючи історичну базу для подальшої аналітики. Саме на цьому рівні відбувається перехід від окремих фактів збору інформації до структурованого масиву знань про ринок.

Висновки до розділу:

1. Модуль зберігання даних є ключовою ланкою між етапами збору та аналітичного опрацювання інформації, забезпечуючи структуроване та довготривале накопичення даних.

2. Використання реляційної моделі та патерну DAO гарантує цілісність, масштабованість і гнучкість системи, а також спрощує оптимізацію запитів та зміну СУБД.

3. Реалізовані механізми ідентифікації товарів (на основі URL або SKU) запобігають дублюванню записів та забезпечують коректний зв’язок між статичними й динамічними характеристиками.

4. Ведення історії цін у спеціалізованій таблиці дозволяє відстежувати динаміку змін, а застосування умовного збереження (upsert-логіки) мінімізує надлишкові записи, зберігаючи лише значущі події.

5. Використання транзакційного механізму гарантує атомарність і узгодженість усіх операцій, що особливо важливо у багатопотоковому середовищі з високим навантаженням.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ВІЗУАЛІЗАЦІЯ РЕЗУЛЬТАТІВ МОНІТОРИНГУ В E-COMMERCE

З огляду на складність предметної області, виникає необхідність обробки великих обсягів динамічних даних, легкої масштабованості й гнучкості системи, тому реалізація системи повинна базуватись на мікросервісній архітектурі з використанням Java та фреймворку Spring Boot. Даний підхід дозволяє ізолювати окремі функціональні підсистеми, забезпечує незалежний розвиток компонентів, а також підвищує надійність та підтримуваність системи в цілому.

Тому загальна архітектура системи повинна мати сукупність спеціалізованих сервісів, кожен з яких буде відповідати за окремий аспект функціонування системи моніторингу. Основна архітектура системи становить сервіс збору даних (Scraping Service), який реалізовує web-скрепінг та автоматизоване отримання інформації про товари, ціни та відгуки з e-commerce платформи. Даний сервіс повинен функціонувати в багатопотоковому режимі, використовувати механізм headless-браузер та передавати результати отриманої інформації до іншого компоненту.

Для інтеграції з магазинами що використовують дані у структурованому вигляді, необхідно використовувати наступний сервіс який обробляє XML-фіди. Даний сервіс повинен приймати, валідувати та трансформувати XML-прайс-лист у внутрішній формат даних. Даний підхід дозволяє поєднати класичний web-скрепінг з більш стабільним каналом отримання даних від платформ, що підвищить повноту та актуальність зібраних даних.

Взаємодія між сервісами здійснюється за допомогою API-сервісу, який реалізує REST-інтерфейси для обміну даними між внутрішніми сервісами та зовнішніми клієнтами. Також даний сервіс виступає точкою інтеграції з клієнтськими застосунками, веб-інтерфейсом користувача та сервісом повідомлення.

Для швидкого інформування користувачів про зміни цін, наявність товарів або аналітичного висновку в системі використовується Telegram-сервіс, який автоматично надсилає повідомлень. Це дозволяє реалізувати механізм push-повідомлень без необхідності постійно взаємодіяти користувача з веб-інтерфейсом.

В окремий сервіс винесено аналітичну обробку даних, що реалізує методи часових рядів, агрегації даних та інтелектуальний аналіз відгуків користувачів з використанням NLP та елементів штучного інтелекту. Даний підхід дозволяє масштабувати аналітичні алгоритми незалежно від підсистеми збору даних та забезпечує гнучкість у випадку розширення функціональності.

Окремо використовується сервіс аутентифікації та авторизації, який відповідає за керування користувачами, контролю доступу до функціональних можливостей системи та захист API.

Отже реалізація системи базується на мікросервісній архітектурі, яка поєднує сервіси збору, обробки, аналітики, зберігання та візуалізації даних в єдиній узгодженій системі. Такий підхід забезпечує масштабованість, гнучкість та надійність системи моніторингу, та забезпечує можливість подальшого розвитку системи.

3.1. Реалізація програмного забезпечення для автоматизованого збору та оновлення інформації про товари й ціни

Реалізація програмного забезпечення інтелектуальної системи моніторингу e-commerce платформ здійснюється відповідно до архітектурних принципів, сформульованих у розділі 2, та базується на мікросервісному підході. Такий підхід дозволяє розподілити складну функціональність між окремими незалежними компонентами, кожен з яких виконує чітко визначену роль у загальному процесі збору, обробки та надання аналітичної інформації.

Була розроблена архітектура інтелектуальної системи моніторингу товарів на e-commerce платформах з використанням мікросервісної архітектури яка приведена на рисунку 3.1.

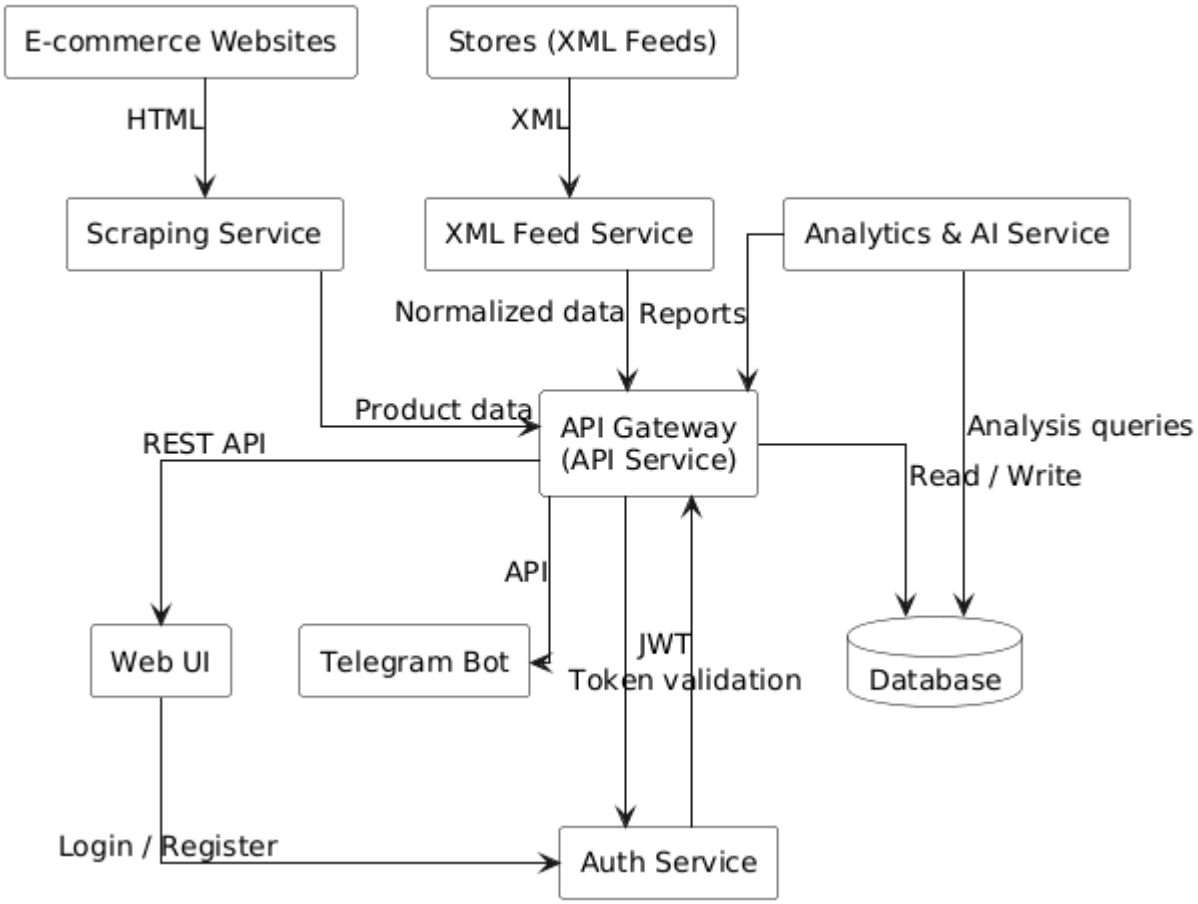


Рисунок 3.1 – Архітектура інтелектуальної системи моніторингу товарів на e-commerce платформах

Розроблена архітектура складається с наступних сервісів:

- сервіс збору даних (Scraping Service);
- сервіс обробки XML-фідів;
- API сервіс для взаємодії між нашими сервісами та зовнішніми;
- сервіс сповіщання (Telegram Bot);
- веб-інтерфейс користувача (Web UI);
- сервіс аутентифікації та авторизації.

Даний підхід дозволяє ізолювати критичні ділянки логіки, мінімізувати зв’язність між компонентами та забезпечити можливість масштабування окремих сервісів залежно від навантаження

Реалізація сервісу збору даних Scraping Service, є одним з ключових компонентів системи яка відповідає за автоматизоване отримання інформації з e-commerce платформ.

- Java Concurrency API для багатопотокової обробки;
- бібліотеки Jsoup та Selenium (headless browser) для отримання та аналізу HTML-контенту;
- конфігураційні профілі для підтримки різних e-commerce платформ.

Сервіс працює в режимі фонової обробки, можна налаштовувати періодичність процесу збору даних за допомогою планувальника (реалізація в Java планувальника за допомогою анотації @Scheduled) або виконання за запитом з боку API-сервісу.

На рисунку 3.2 наведена логічна схема роботи Scraping Service. На рисунку 3.3 наведена послідовна схема виконання роботи Scraping Service. З рисунку 3.2 та 3.3 послідовність роботи сервісу Scraping Service виглядає наступним чином:

1. Ініціалізація та керування чергою: планувальник Scheduler формує завдання та наповнює чергу BlockingQueue, для забезпечення потокобезпечного розподілу URL-адрес.
2. Паралельна обробка: пул робочих потоків (Worker Pool) асинхронно обробляються з черги.
3. Мережева взаємодія та маскування: для кожного запиту Proxy Manager динамічно підбирає проксі-сервіс, що дозволяє ефективно обходити anti-scraping захист. Завантаження контенту здійснюється за допомогою HTTP-клієнтів або Headless Browser для SPA-сайтів.
4. Аналітика та трансформування: отриманий HTML Response передається до модулю Parsing Module в якому відбувається обробка та перетворення в структуровані об’єкти DTO.

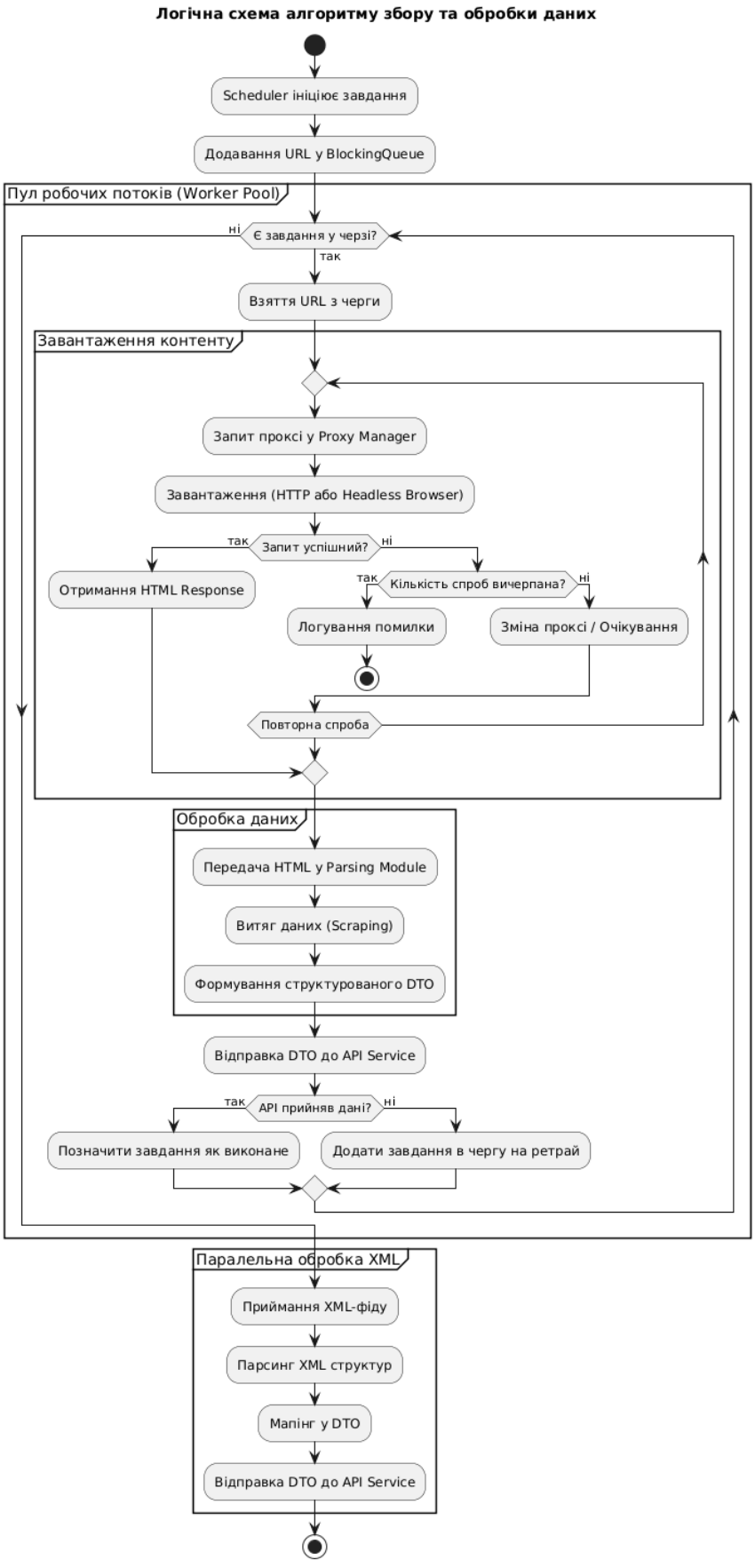


Рисунок 3.2 – Логічна схема роботи Scraping Service

5. Надійність та стійкість: в системі передбачений механізм обробки помилок в разі невдалих запитів або блокувань.

6. Додатковий канал: Паралельно з основним циклом працює сервіс обробки XML-фідів, для забезпечення інтеграції від платформ напряму без етапу веб-скрапінгу.

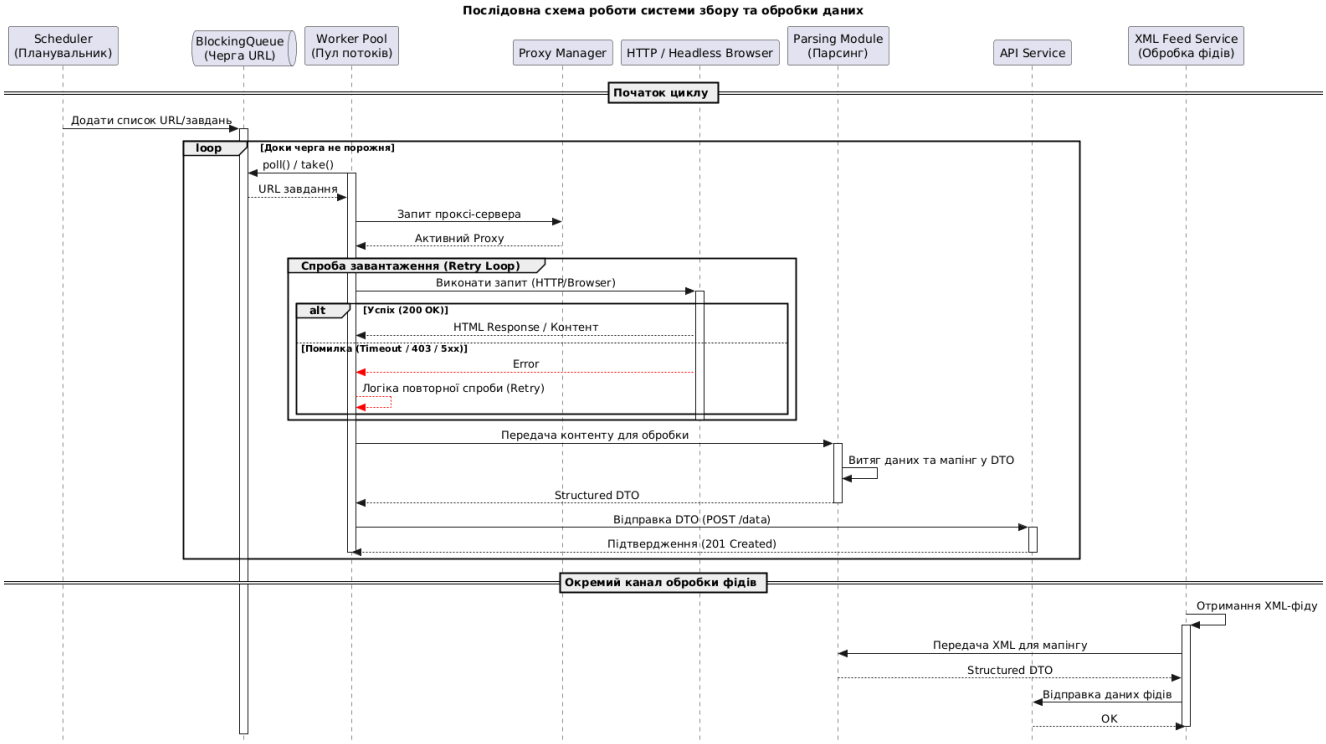


Рисунок 3.3 – Послідовна схема роботи Scraping Service

Центральною точкою для інтеграції в нашій системі є API-сервіс. Він виконує роль центрального вузлу системи та забезпечує взаємодії між всіма внутрішніми сервісами та зовнішніми клієнтами.

API-сервіс виконує роль центрального вузла системи та забезпечує взаємодію між всіма внутрішніми сервісами й зовнішніми клієнтами. Даний сервіс реалізує REST API для наступного функціоналу:

- доступ до даних як актуальних так і до попередніх;
- ініціалізація поцесу збору даних;
- отримання аналітичних показників;
- інтеграція з Telegram-ботом та веб-інтерфейсом.

Сервіс реалізовано до принципів RESTful-архітектури, з використанням JSON як основний формат обміну даних. На рисунку 3.4 наведено схему взаємодії між сервісами через API-сервіс.

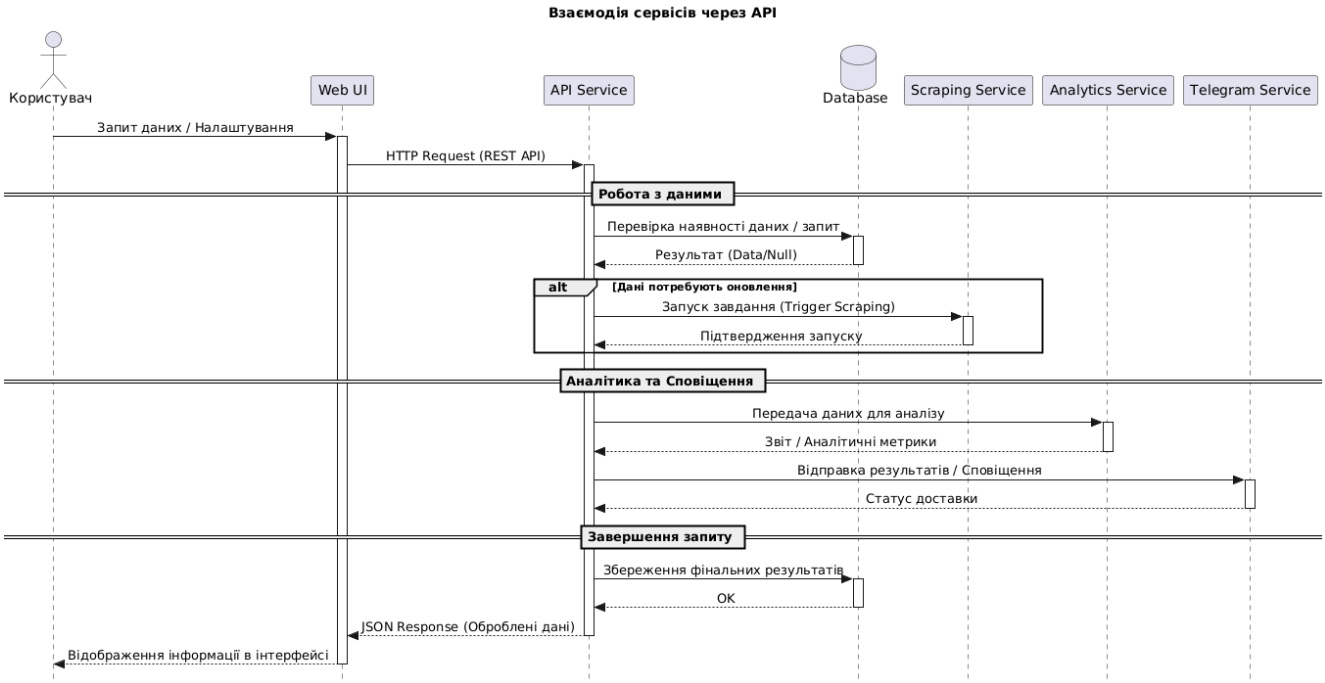


Рисунок 3.4 – Схема взаємодії сервісів через API-сервіс

Відповідно до рисунку 3.4 алгоритм роботи системи в межах одного запиту виконується наступним чином:

Користувач за допомогою веб-інтрефейсу (Web UI) виконує запит до системи. Клієнтська частина передає даний запит до API Service.

API Service виконує роботу з базою даних (Database), для отримання даних результатів збору даних, отримання або встановлення налаштувань користувача.

У разі необхідності оновлення або відсутності даних, сервіс надсилає команду до сервісу Scraping Service для запуску циклю скрапінгу для отримання актуальних даних після виконання веб-скрапінгу.

Отримані дані передаються до модулю аналітики, для подальшої обробки ат формування статистики.

В даному підрозділі було розглянуто практичну реалізацію та архітектуру інтелектуальної системи моніторингу e-commerce платформ. Використання мікросервісної архітектури та Java Spring Boot дозволяє реалізовувати гнучку,

масштабовану та відмово стійку систему та працювати з великим обсягом динамічних даних.

3.2. Методи аналітичної обробки даних та побудови інтелектуальних звітів і дашбордів

Однією з ключових функцій інтелектуальної системи моніторингу e-commerce платформ є аналітика та обробка накопичених даних та представлення результатів в зручному форматі та вигляді користувачу. В нашій системі за це відповідає модуль Analytics & AI Service.

Модуль аналітики реалізовано в окремому сервісі Analytics & AI Service, який взаємодіє з базою даних та API-сервісом. Основна задача даного сервісу є агрегація, статична обробка, інтелектуальний аналіз та підготовка даних для візуалізації в веб-інтерфейсі та сервісі сповіщання (Telegram-бот).

Аналітична обробка даних у проектованій системі здійснюється у вигляді послідовного багатоступеневого конвеєра, що забезпечує поетапну трансформацію первинної інформації у формат, придатний для подальшої інтерпретації та візуалізації. На першому етапі виконується вибірка структурованих даних із реляційної бази даних, яка містить інформацію про товари, історію зміни цін та відгуки користувачів. Доступ до даних реалізується через шар Data Selection Layer, що використовує SQL-запити та репозиторії, ізолюючи бізнес-логіку від механізмів зберігання.

На другому етапі отримані дані передаються до модуля агрегації та нормалізації, де здійснюється узагальнення показників за часовими інтервалами, вирівнювання форматів даних та усунення пропусків. Це дозволяє підготувати дані до коректного порівняльного аналізу та обчислення аналітичних показників.

Наступним кроком є розрахунок аналітичних метрик, зокрема середніх і мінімальних цін, темпів зміни вартостя а також агрегованих показників попиту та доступності товарів. Ці метрики формують основу для побудови звітів і графіків динаміки.

На четвертому етапі здійснюється інтеграція результатів інтелектуального аналізу, зокрема обробка текстових відгуків користувачів із застосуванням методів обробки природної мови (NLP). У межах цього етапу визначається тональність відгуків, виділяються ключові теми та формуються узагальнені показники споживчого сприйняття товарів.

Завершальним етапом аналітичного конвеєра є формування структурованих наборів даних у форматі JSON, придатних для подальшої візуалізації у користувацьких інтерфейсах Web UI та Telegram Bot. Такий підхід забезпечує чітке розмежування між рівнем аналітики та рівнем представлення, підвищує гнучкість системи та спрощує інтеграцію нових каналів відображення результатів.

Узагальнена схема послідовної трансформації даних у процесі аналітичної обробки та візуалізації приведена на рисунку 3.5

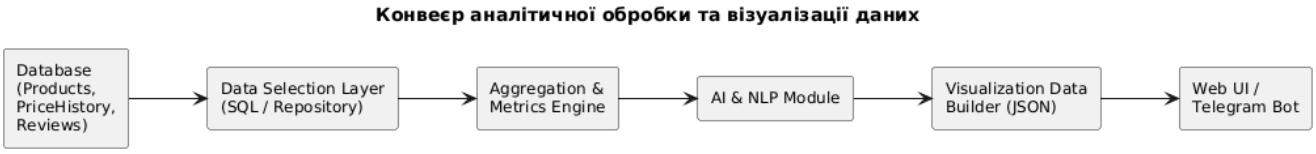


Рисунок 3.5 – Узагальнена схема аналітичного конвеєра

До базового елементу аналітичної обробки даних відноситься агрегація даних, що дозволяє узагальнити показники. До основних агрегованих показників відноситься:

- мінімальна, максимальна та середня ціна товару за вибраний період;
- кількість змін ціни за певний інтервал часу;
- середня тривалість наявності товару;
- частота зміни статусу доступності товару.

На основі агрегованих даних система розраховує аналітичні показники KPI (Key Performance Indicators), які використовуються для оцінки конкурентоспроможності та позиції товару на ринку. Аналітичний звіт наша система має наступні показники: Market Average Price, Price Deviation Index, Price Volatility Index, Availability Rate, Competitor Density.

Market Average Price це середня ринкова ціна товару, що обчислюється як середнє арифметичне значення цін усіх зафіксованих продавців за вибраний часовий період. Даний показник використовується як еталонне значення для порівняння цін окремих магазинів та дозволяє оцінити загальний рівень цін на ринку. Market Average Price є базовою метрикою для формування інших похідних показників, зокрема індексів відхилення та волатильності.

Price Deviation Index це індекс відхилення ціни конкретного продавця від середньоринкового значення, виражений у відсотках. Показник розраховується як відносна різниця між ціною магазину та значенням Market Average Price. Даний індекс дозволяє швидко визначити, чи є ціна продавця конкурентною, завищеною або заниженою, та використовується для візуального маркування у дашбордах за допомогою кольорових індикаторів.

Price Volatility Index це індикатор стабільності або нестабільності ціни товару протягом заданого періоду часу. Показник базується на аналізі частоти та амплітуди змін ціни і дозволяє виявляти товари з високою ціновою динамікою, що може свідчити про акційні пропозиції, агресивну цінову політику або нестабільність ринку. Високе значення індексу волатильності сигналізує про підвищені ризики для прогнозування та потребу у більш частому моніторингу.

Availability Rate це показник доступності товару, який відображає відсоток часу, протягом якого товар перебував у статусі «в наявності» за обраний період спостереження. Даний KPI формується на основі історії змін статусу доступності та дозволяє оцінити надійність продавця, стабільність постачання та реальний потенціал продажу товару. Низьке значення Availability Rate може свідчити про логістичні проблеми або нерегулярність поставок.

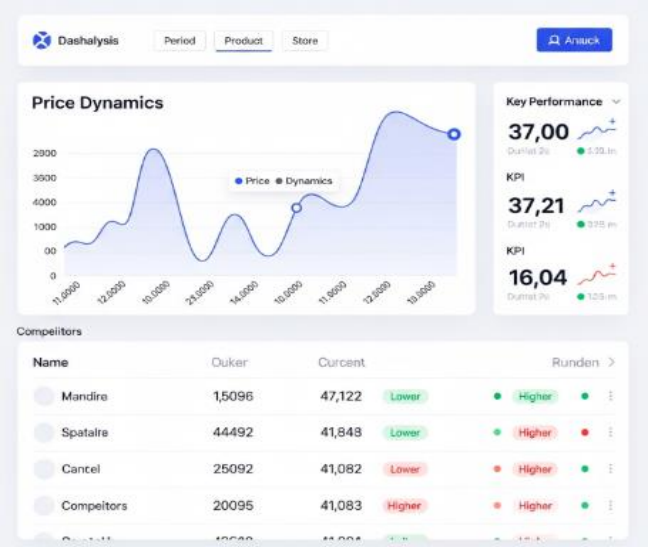
Competitor Density це показник щільності конкурентів, що визначається як кількість продавців, які одночасно пропонують аналогічний товар на ринку. Цей KPI використовується для оцінки рівня конкуренції у конкретному сегменті та дозволяє інтерпретувати інші показники у контексті ринкового середовища. Наприклад, висока щільність конкурентів у поєднанні з низьким Price Deviation Index може свідчити про цінову війну.

Зазначені показники формують аналітичне ядро системи та використовуються як для оперативного моніторингу ринку, так і для побудови узагальнених звітів та візуалізацій. Їх поєднання дозволяє отримати комплексне уявлення про стан ринку, динаміку цін та конкурентне середовище, що створює основу для прийняття обґрунтованих управлінських рішень.

Окремий напрям аналітики у системі пов’язаний з обробкою текстових відгуків користувачів. Враховуючи значний вплив думки споживачів на рішення про купівлю, результати NLP-аналізу інтегруються у загальну модель оцінки товару. Сервіс реалізовує аналіз відгуків тобто позитивний, нейтральний або негативний, проведення агрегацію оцінок у зведений індекс репутації. Результати NLP-аналізу зберігаються у базі даних та використовуються як додатковий аналітичний вимір поряд із ціновими показниками.

Для кінцевого користувача результати аналітичної обробки представлені у вигляді інтерактивних дашбордів та автоматизованих звітів. Веб-інтерфейс реалізує візуалізацію даних з використанням графіків, таблиць та індикаторів стану, що дозволяє швидко оцінити ринкову ситуацію.

Паралельно з веб-інтерфейсом реалізовано Telegram-інтеграцію, яка дозволяє надсилати користувачеві стислий аналітичний зведений звіт або сповіщення про критичні події (зміна ціни, відсутність товару). На рисунку 3.6 приведено приклад макету дашборду системи моніторингу.



Рисунко 3.6 – Дашборд системи моніторингу

На макеті інтерфейсу дашборду з рисунку 3.6 бачмо присутність, панелі фільтрів (період, товар, магазин), графік динаміки цін, панель KPI, таблицю конкурентів з індикаторами.

В даному підрозділі було розглянуто методи аналітичної обробки даних та побудови інтелектуальних звітів у межах розроблюваної системи моніторингу e-commerce. Запропонований аналітичний підхід поєднує класичні методи статистичної обробки з елементами інтелектуального аналізу текстових даних, що дозволяє отримувати комплексну оцінку ринку та поведінки споживачів. Реалізація дашбордів та сервісів сповіщень забезпечує зручне представлення результатів та підвищує практичну цінність системи.

3.3 Методика та інструменти тестування програмної реалізації системи моніторингу e-commerce

Розробка інтелектуальної системи моніторингу e-commerce платформ передбачає не лише реалізацію функціональних можливостей, але й забезпечення надійності та стабільності її роботи в умовах реального використання. З огляду на багатокомпонентну архітектуру, багатопотоковий характер виконання та інтеграцію із зовнішніми веб-ресурсами, процес тестування набуває особливої важливості. Саме систематичне тестування дозволяє виявити потенційні помилки на ранніх етапах, оцінити коректність взаємодії між модулями та підтвердити відповідність програмної реалізації архітектурним рішенням, сформульованим у попередніх розділах.

У межах даного підрозділу розглядається методика тестування програмної реалізації системи, а також інструменти, які застосовуються для перевірки її функціональних та не функціональних характеристик. Тестування розглядається як невід’ємна частина життєвого циклу програмного забезпечення, що супроводжує всі етапи розробки, від реалізації окремих класів до перевірки цілісної роботи системи в умовах наближених до реальних.

Оскільки наша архітектура системи складається з різних сервісів то таку систему необхідно тестувати багаторівневим підходом, для забезпечення перевірки як окремих компонентів, та взаємодію компонентів.

Загальна стратегія тестування включає наступне:

- модульне тестування окремих компонентів;
- інтеграційне тестування взаємодії сервісів;
- тестування API та зовнішніх інтерфейсів;
- тестування роботи з базою даних;
- навантажувальне та стрес-тестування.

Кожен рівень тестування має власну мету та набір інструментів, що дозволяє забезпечити комплексну перевірку системи. На рисунку 3.7 наведено рівні тестування, нижній рівень Unit Tests, середній рівень Integration Tests, верхній рівень System & Load Tests, та бачимо що чим вище рівень тим збільшується складність тестів, але зменшується їх кількість.

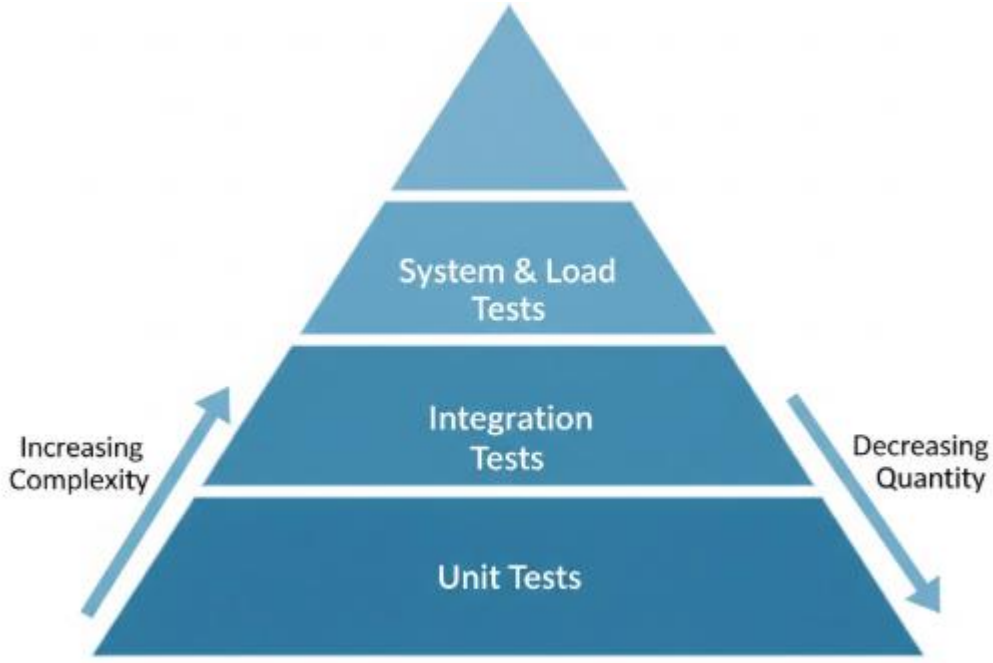


Рисунок 3.7 – Рівні тестування

Модульне тестування (Unit Testing) спрямоване на перевірку коректності роботи окремих класів та методів без урахування зовнішніх залежностей. У

проектованій системі особливу увагу приділено тестуванню сервісного шару, де зосереджена бізнес логіка обробки даних.

Для реалізації Unit Testing використовується наступні фреймворки:

- JUnit 5 фреймворк для написання тестів;
- Mockito фреймворк для створення mock-об’єктів, що імітують зовнішні залежності (DAO, API клієнтів).

Основні сценарії модульного тестування включають тестування алгоритмів нормалізації, валідації даних, перевірку розрахунків показників KPI, обробки винятків.

Даний підхід тестування дозволяє ізолювати логіку модулю, що тестується та гарантувати стабільність незалежно від стану інших компонентів системи.

Як приклад в нас є клас ProductKpiService, в якому є метод, що визначає процент доступності товару calculateAvailabilityRate, тоді модульний тест для даного методу будет такий:

```
class ProductKpiServiceTest {
    @Test
    void testCalculateAvailabilityRate_NormalCase() {
        ProductKpiService service = new ProductKpiService();
        // Лог доступності: товар був доступний 3 рази з 5
        List<Boolean> log = Arrays.asList(true, true, false, true, false);
        double result = service.calculateAvailabilityRate(log);
        // Очікуваний результат: 3/5 = 60%
        assertEquals(60.0, result, 0.001, "Доступність повинна бути 60%");
    }

    @Test
    void testCalculateAvailabilityRate_AllAvailable() {
        ProductKpiService service = new ProductKpiService();
        // Лог доступності: товар завжди був у наявності
        List<Boolean> log = Arrays.asList(true, true, true, true);
```

```
        double result = service.calculateAvailabilityRate(log);
        // Очікуваний результат: 100%
        assertEquals(100.0, result, 0.001, "Доступність повинна бути 100%");
    }

    @Test
    void testCalculateAvailabilityRate_AllUnavailable() {
        ProductKpiService service = new ProductKpiService();
        // Лог доступності: товар жодного разу не був у наявності
        List<Boolean> log = Arrays.asList(false, false, false);
        double result = service.calculateAvailabilityRate(log);
        // Очікуваний результат: 0%
        assertEquals(0.0, result, 0.001, "Доступність повинна бути 0%");
    }

    @Test
    void testCalculateAvailabilityRate_EmptyLog() {
        ProductKpiService service = new ProductKpiService();
        // Перевірка винятку: якщо лог порожній, метод має кинути IllegalArgumentException
        assertThrows(IllegalArgumentException.class, () -> {
            service.calculateAvailabilityRate(Arrays.asList());
        });
    }
}
```

Даний модульний тест ProductKpiServiceTest виконує тести коли система моніторингу фіксує наявність товару на Rozetka один раз на годину. Метод calculateAvailabilityRate розраховує відсоток доступності товару, в тестах ми перевіряємо сценарії: нормальні випадок, всі доступні, всі недоступні, відсутність даних оскільки лог пустий.

Інтеграційне тестування спрямоване на перевірку коректності взаємодії між окремими модулями та сервісами системи. З урахуванням мікросервісної архітектури особлива увага приділяється перевірці обміну даними між сервісом збору, API-сервісом, аналітичним сервісом та сервісом збереження даних.

На цьому етапі тестування перевіряється:

- коректність передачі DTO між сервісами;
- узгодженість форматів даних;
- обробка помилок при недоступності окремих компонентів;
- стабільність роботи у багатопотоковому режимі.

Для інтеграційного тестування використовуються вбудовані можливості Spring Boot Test, що дозволяють піднімати тестове середовище з мінімальною конфігурацією.

На рисунку 3.8 приведена схема інтеграційного тестування сервісів

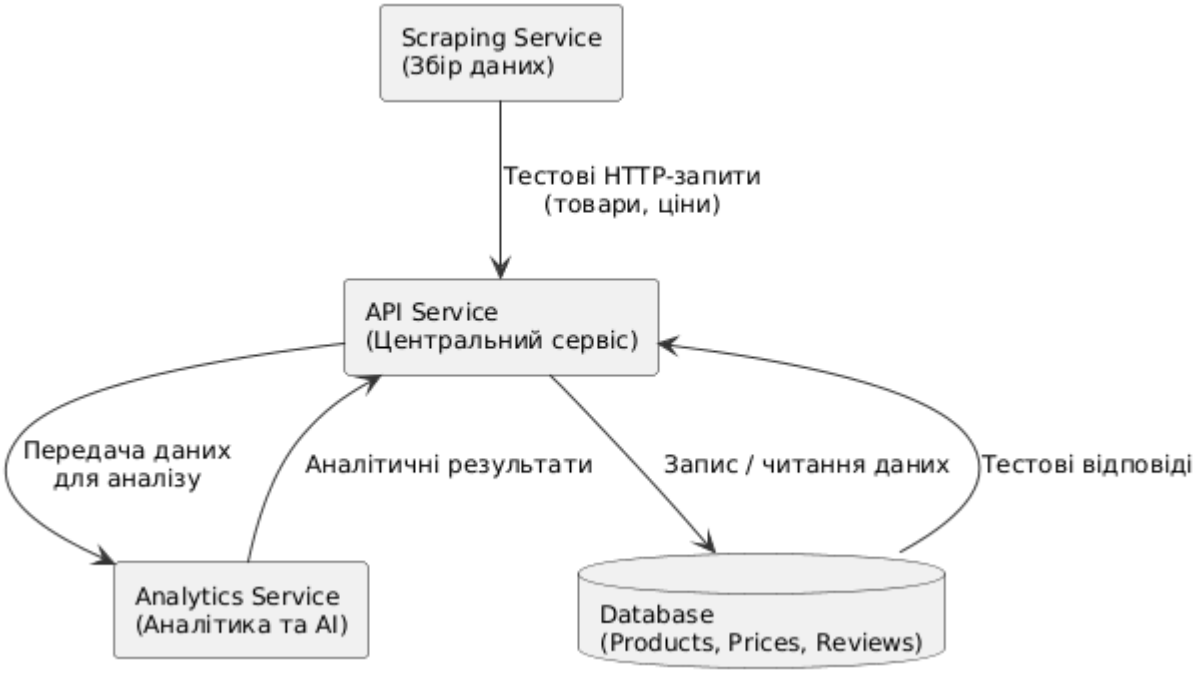


Рисунок 3.8 – Схема інтеграційного тестування сервісів

Приклад інтеграційного тесту:

```
@SpringBootTest // Завантажує повний контекст Spring Boot
@Testcontainers // Активує підтримку Testcontainers
public class AnalyticsIntegrationTest {
   // 1. Налаштування контейнера бази даних (запускається перед тестом)
```

```
    @Container
static PostgreSQLContainer<?> postgres = new PostgreSQLContainer<>("postgres:15-alpine")
            .withDatabaseName("monitoring_test_db")
            .withUsername("testuser")
            .withPassword("testpass");

    @Autowired
    private PriceRepository priceRepository; // Репозиторій для роботи з БД

    @Autowired
    private AnalyticsService analyticsService; // Сервіс, який ми тестуємо

    @Test
    void shouldCalculateAveragePriceCorrectly() {
        // 1: Підготовка даних (Arrange) ---
        // Імітуємо роботу Scraping Service: додаємо "сирі" дані в базу
        ProductPrice priceRozetka = new ProductPrice();
        priceRozetka.setProductName("iPhone 15 128GB");
        priceRozetka.setStoreName("Rozetka");
        priceRozetka.setPrice(new BigDecimal("40000"));
        priceRozetka.setScrapedAt(LocalDateTime.now());

        ProductPrice priceAllo = new ProductPrice();
        priceAllo.setProductName("iPhone 15 128GB");
        priceAllo.setStoreName("Allo");
        priceAllo.setPrice(new BigDecimal("42000")); // Трохи дорожче
        priceAllo.setScrapedAt(LocalDateTime.now());

        ProductPrice priceProm = new ProductPrice();
```

```
        pricеProm.setProductName("iPhone 15 128GB");
        pricеProm.setStoreName("Prom");
        pricеProm.setPrice(new BigDecimal("38000")); // Дешевше
        pricеProm.setScrapedAt(LocalDateTime.now());

        // Зберігаємо ціни в реальну тестову БД
        priceRepository.saveAll(List.of(priceRozetka, priceAllo, pricеProm));

        //  2: Виконання дії (Act) ---
        // Викликаємо метод аналітичного сервісу, який робить SQL-запит AVG()
        BigDecimal averagePrice = analyticsService.calculateAveragePrice("iPhone 15 128GB");

        // : Перевірка результату (Assert) ---
        // (40000 + 42000 + 38000) / 3 = 40000
        BigDecimal expectedAverage = new BigDecimal("40000.00"); // Очікуване значення

        // Перевіряємо, що результат не null
        Assertions.assertNotNull(averagePrice, "Середня ціна не повинна бути null");

      // Перевіряємо точність обчислень (використовуємо compareTo для BigDecimal)
        Assertions.assertEquals(0, expectedAverage.compareTo(averagePrice),
            "Розрахована середня ціна має відповідати очікуваній");

        System.out.println("Тест пройдено успішно! Середня ціна: " + averagePrice);
    }
}
```

Оскільки система надає доступ до даних через REST API та інтегрується з Telegram-ботом та веб-інтерфейсом, важливим етапом є тестування зовнішніх інтерфейсів взаємодії.

API-тестування охоплює наступні компоненти:

- перевірку коректності HTTP-методів (GET, POST);
- тестування валідації вхідних параметрів;
- перевірку форматів JSON-відповідей;
- обробку помилкових запитів та статус-кодів.

Для API-тестування використовуються наступні інструменти та фреймворки:

- Postman — для ручного тестування API;
- Spring MockMvc — для автоматизованих тестів контролерів.

Такий підхід гарантує стабільність публічних інтерфейсів та коректність інтеграції з клієнтськими застосунками.

На рисунку 3.9 наведена схема прикладу тестування REST API

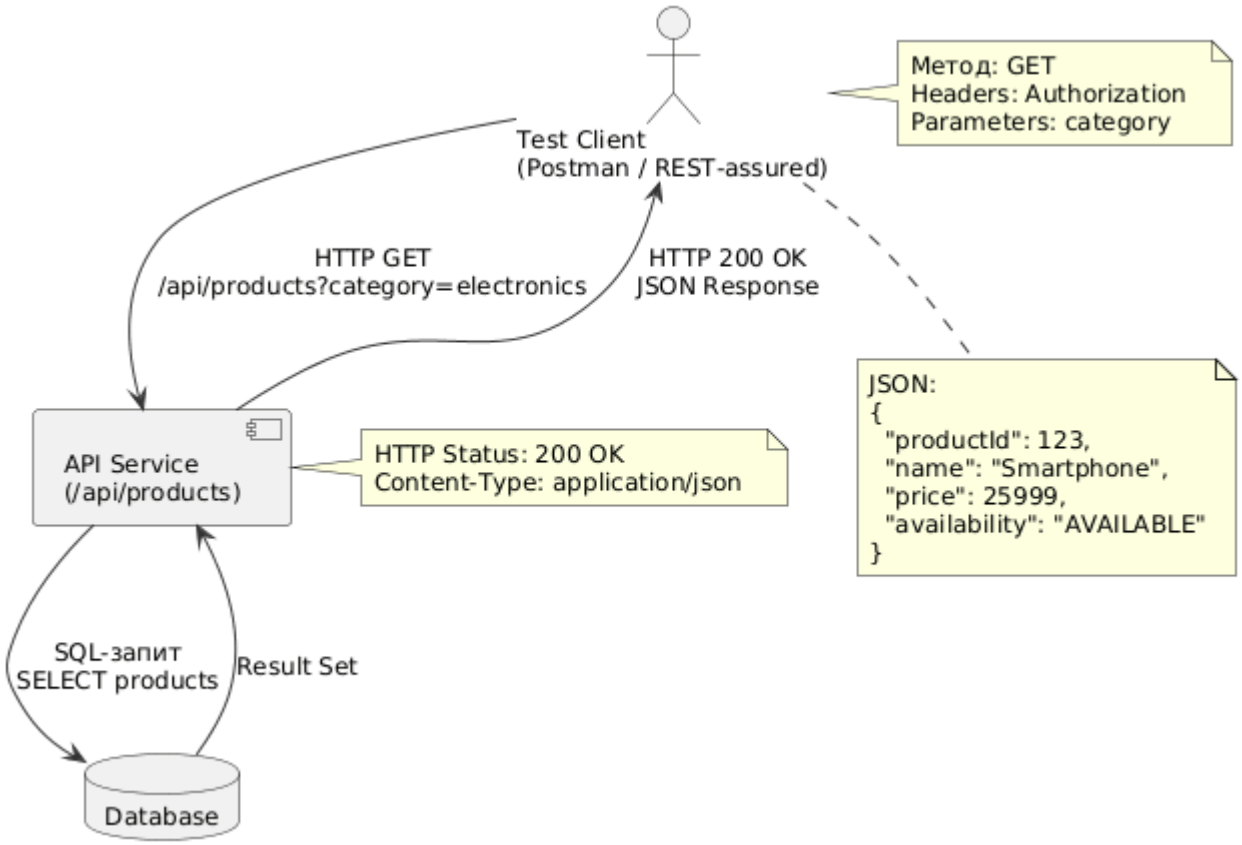


Рисунок 3.9 – Приклад тестування REST API

Тестування шару збереження даних, робота з базою даних, спрямоване на перевірку коректності CRUD-операцій, транзакційної логіки та upsert-механізмів. Особливу увагу приділено перевірці цілісності даних у багатопотоковому середовищі.

Основні сценарії тестування роботи з базою даних включають:

- коректність збереження нових товарів;
- оновлення історії цін без дублювання записів;
- перевірку транзакційного відкату у випадку помилок.

Для тестування роботи з базою даних використовуються тестові бази даних або in-memory рішення (наприклад H2), що дозволяє ізольовано перевіряти логіку доступу до даних.

На рисунку 3.10 наведена схема тестування транзакцій.

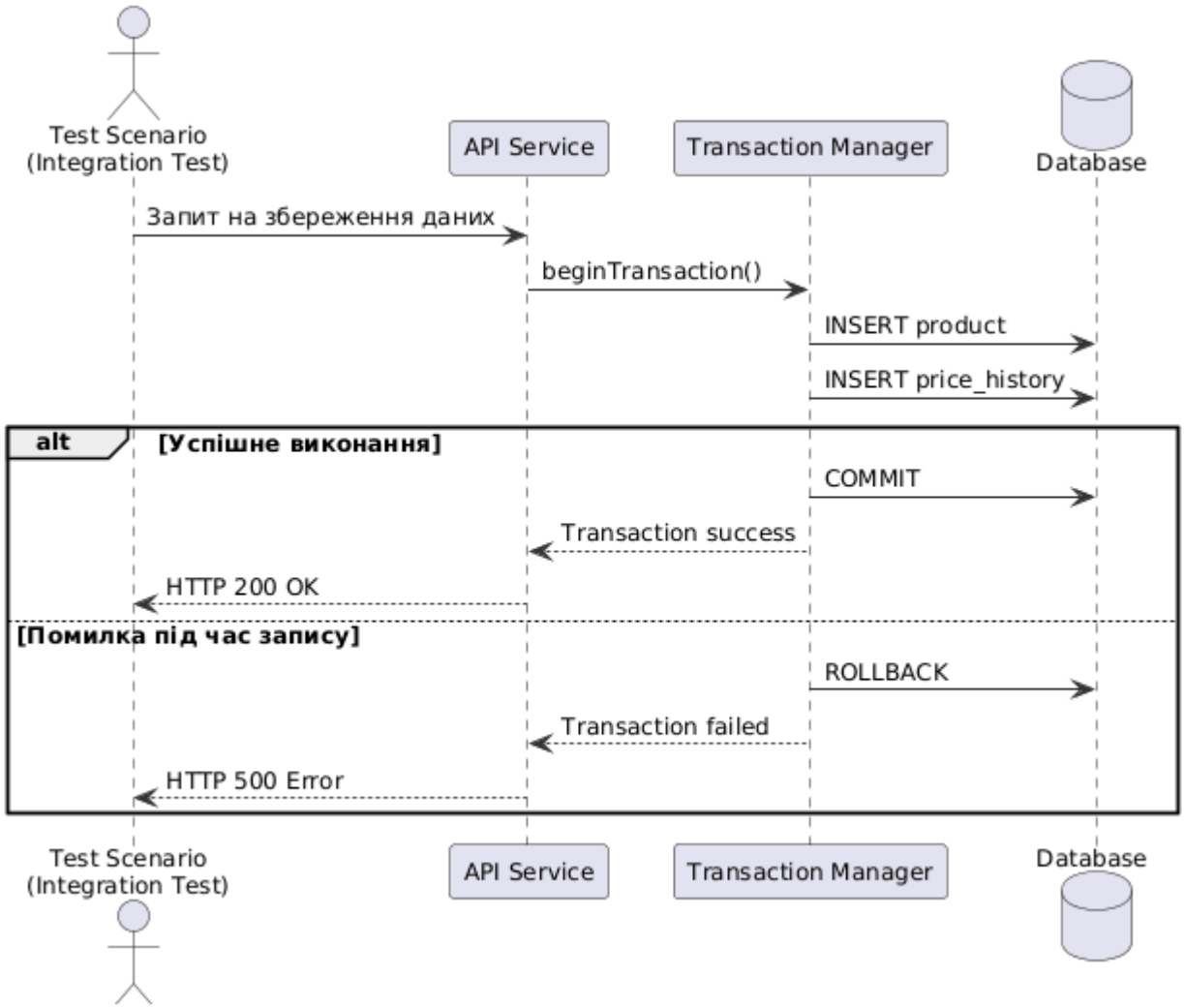


Рисунок 3.10 – Схема тестування транзакцій

В даному підрозділі було розглянуто методику та інструменти тестування програмної реалізації інтелектуальної системи моніторингу e-commerce. Застосування багаторівневого підходу до тестування дозволяє забезпечити коректність роботи окремих компонентів, стабільність взаємодії між сервісами та надійність системи в умовах високого навантаження. Використання сучасних інструментів автоматизованого тестування у поєднанні з ручною перевіркою зовнішніх інтерфейсів створює основу для якісної та стійкої програмної реалізації, готової до подальшого розвитку та практичного використання.

ВИСНОВКИ

В першому розділі було розглянуто теоретичні основи функціонування електронної комерції та сучасні підходи до моніторингу даних в сфері e-commerce. Розглянуто структуру та функціональні особливості e-commerce платформ, визначено основні моделі взаємодії учасників ринку (B2B, B2C, C2C, B2G, G2C тощо) та охарактеризовано типи даних, що формуються в процесі їх функціонування. Обґрунтовано, що e-commerce платформи є складним об’єктом моніторингу через високу динамічність інформації, наявність механізмів антибот-захисту, часті зміни структури вебсторінок та значні обсяги структурованих, напівструктурованих і неструктурованих даних. Також проаналізовано основні методи збору, попередньої обробки та аналітичного опрацювання даних у середовищі електронної комерції. Визначено, що найбільш поширеними підходами до отримання даних є використання офіційних API та технологій web-скрепінгу, які часто застосовуються у комбінованому вигляді. Розглянуто етапи фільтрації, нормалізації та трансформації даних, а також методи їх подальшого аналізу, зокрема статистичний аналіз, аналіз часових рядів, зіставлення товарних позицій та обробку текстових даних з використанням методів машинного навчання та обробки природної мови. Проаналізовано сучасний стан, тенденції та перспективи розвитку технологій web-скрепінгу та інтелектуального аналізу інформації. Проведено порівняльний аналіз інструментів та мов програмування, що використовуються для збору даних, та обґрунтовано доцільність використання мови програмування Java для побудови масштабованих і багатопотокових систем моніторингу e-commerce платформ. Окрему увагу приділено питанням етичних і правових обмежень збору даних, а також перспективам інтеграції web-скрепінгу з методами глибокого навчання та інтелектуального аналізу.

В другому розділі було проведено аналіз існуючих рішень моніторингу електронної комерції, та було зроблено висновок, що більшість існуючих рішень або орієнтовані на кінцевого користувача та не надають достатньої аналітичної

глибини, або є комерційними продуктами з високою вартістю та обмеженою можливістю адаптації. Це обґрунтовує доцільність розробки власної інтелектуальної системи моніторингу e-commerce, побудованої на основі технологій web-скрепінгу, багатопотокової архітектури Java та методів інтелектуального аналізу даних, що детально розглядається у наступних підрозділах. Також здійснено детальне проєктування архітектури інтелектуальної системи моніторингу e-commerce платформ. Запропонована багаторівнева модульна структура забезпечує гнучкість, масштабованість та стійкість системи до змін зовнішнього середовища, що є критично важливим для роботи у динамічних умовах електронної комерції. Проаналізовано та розроблено модуль зберігання даних є ключовою ланкою між етапами збору та аналітичного опрацювання інформації, забезпечуючи структуроване та довготривале накопичення даних. Реалізовані механізми ідентифікації товарів на основі URL, що запобігають дублюванню записів та забезпечують коректний зв’язок між статичними й динамічними характеристиками.

В третьому розділі реалізовано програмну частину системи на базі платформи Java та фреймворку Spring Boot. Розроблено набір мікросервісів, що відповідають за скрепінг веб-ресурсів, обробку XML-фідів, надання API-доступу до даних, аналітичну обробку інформації та візуалізацію результатів у вигляді дашбордів і звітів. Аналітична підсистема системи реалізує розрахунок ключових показників ефективності (KPI), таких як середня ринкова ціна, індекси відхилення та волатильності цін, рівень наявності товарів і щільність конкурентного середовища. Також розроблено методику тестування програмної реалізації системи, що включає модульне, інтеграційне, системне та навантажувальне тестування. Застосування сучасних інструментів тестування для Java та Spring Boot дозволило оцінити стабільність, коректність та продуктивність розробленого програмного забезпечення в умовах реального навантаження.

Практична значущість отриманих результатів полягає у створенні працездатного прототипу інтелектуальної системи моніторингу e-commerce, який може бути використаний як основа для подальшого розвитку комерційних або

дослідницьких рішень. Запропонована архітектура та реалізовані алгоритмічні підходи дозволяють легко розширювати функціональність системи, підключати нові джерела даних і впроваджувати додаткові методи аналітики.

Подальші напрями розвитку роботи можуть включати інтеграцію більш складних моделей машинного навчання для прогнозування цін, розширення інтелектуального аналізу поведінки споживачів, а також оптимізацію процесів збору даних із використанням розподілених обчислювальних середовищ та хмарних технологій

ПЕРЕЛІК ПОСИЛАНЬ

1. Козак Л. В. Електронна комерція: теорія, практика, перспективи розвитку : навч. посіб. — К. : КНЕУ, 2020. — 312 с.
2. Мельник О. М., Шевчук І. Б. Інформаційні системи та технології в електронній комерції. — Львів : ЛНУ ім. І. Франка, 2019. — 284 с.
3. Ситник В. Ф., Орленко Н. С. Інтелектуальні інформаційні системи : підручник. — К. : КНТ, 2018. — 408 с.
4. Плескач В. Л., Затонацька Т. Г. Електронна комерція. — К. : Знання, 2017. — 535 с.
5. Державна служба статистики України. Електронна торгівля в Україні: аналітичний огляд. — К., 2022. — Режим доступу: https://ukrstat.gov.ua
6. Ковальчук В. В. Методи збору та аналізу великих даних у веб-середовищі // Вісник НТУУ «КПІ». — 2021. — №3. — С. 45–52.
7. Hotline.ua. Аналітика ринку електронної комерції в Україні. — 2023. — Режим доступу: https://hotline.ua
8. Laudon K. C., Traver C. G. E-Commerce: Business, Technology, Society. — 16th ed. — Pearson, 2022. — 720 p.
9. Mitchell R. Web Scraping with Python: Collecting More Data from the Modern Web. — 2nd ed. — O’Reilly Media, 2018. — 320 p.
10. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. — 4th ed. — Pearson, 2021. — 1136 p.
11. Provost F., Fawcett T. Data Science for Business. — O’Reilly Media, 2013. — 414 p.
12. Han J., Kamber M., Pei J. Data Mining: Concepts and Techniques. — 3rd ed. — Morgan Kaufmann, 2011. — 744 p.
13. Zhao B., Wu J. Web Data Extraction and Applications in E-Commerce // IEEE Access. — 2020. — Vol. 8. — P. 123456–123468.

14. García-Molina H., Ullman J. D., Widom J. Database Systems: The Complete Book. — 2nd ed. — Pearson, 2008. — 1112 p.

15. Few S. Information Dashboard Design: Displaying Data for At-a-Glance Monitoring. — 2nd ed. — Analytics Press, 2013. — 211 p.

16. Sahin K. — *The Java Web Scraping Handbook* (2020) — практичний гайд з web scraping на Java.

17. Apache Software Foundation — *StormCrawler (Java Web Crawler)* (документація/ресурси проекту).

18. Tudose C. — *Java Persistence with Spring Data and Hibernate* (Manning, 2023).

19. Liu W., Mondal S., Chen T.-H. — *An Empirical Study on Database Access Bugs in Java Apps* (2024).

20. *Jakarta Persistence Specification (JPA 3.x)* — офіційна специфікація persistence стандартів.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

Презентація

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

ІНТЕЛЕКТУАЛЬНА СИСТЕМА МОНІТОРИНГУ ТОВАРІВ НА E-COMMERCE ПЛАТФОРМАХ З ВИКОРИСТАННЯМ WEB-СКРЕПІНГУ ТА ВІЗУАЛІЗАЦІЇ

Ст.гр.САДМ-61 Толмачев Р.В.

Науковий керівник, доцент Патракеєв І.М.

Мета

- ПЗ для автоматизованого аналізу, та структуризації даних веб-додатків

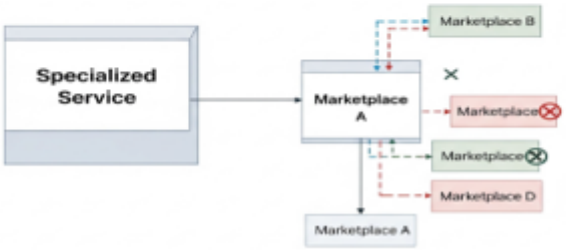
- усунення рутинних операцій;
- зниження трудовитрат при виконанні традиційних процесів і операцій;
- надання більших можливостей користувача за рахунок використання інформаційних технологій;
- збільшення швидкості обробки інформації і процесів перетворення;

Проблеми та актуальність

- Інтернет переповнений інформацією. Людина не в силах оброблювати потоки даних.
- Динамічний JavaScript-контент.
- Антибот-захист (CAPTCHA, rate limiting, IP-блокування).
- Часті зміни HTML-структури сторінок.
- Великі обсяги слабко структурованих та неструктурованих даних.

Завдання автоматизації

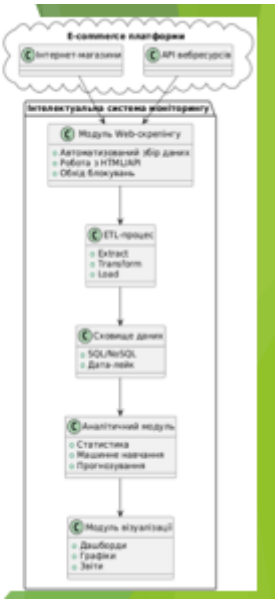
- забезпечення більшої ефективності та якості обслуговування клієнтів;
- надання широких можливостей для статистичного аналізу та підвищення точності обліку та звітності інформації;
- зниження трудовитрат при виконанні традиційних процесів і операцій;
- збільшення швидкості обробки інформації і процесів перетворення;



Логіка роботи парсеру

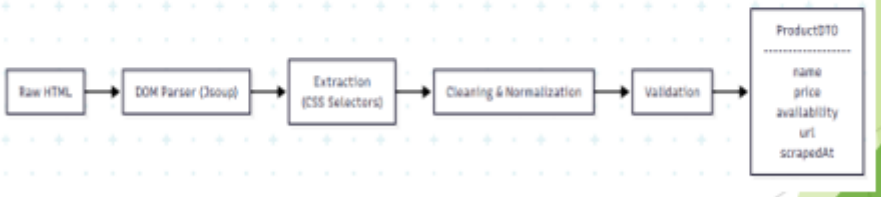
Аналіз технічних рішень

- Парсер полегшує виокремлення і анліз даних (робить всю роботу за вас)
- Популярність чат-ботів зростає.
- Компанії все частіше замовляють розробникам нових чат-ботів.



Переваги

- додавання нових критеріїв парсингу без зміни ядра;
- зміну або розширення наборів видобутих даних;
- підключення нових e-commerce платформ;
- адаптацію до змін HTML-структури сторінок.



Використовувані програмні продукти

- Jsoup — для парсингу статичних HTML-сторінок;
- Selenium WebDriver — для роботи з динамічним JavaScript-контентом;
- HtmlUnit — легкий headless-браузер для зменшення навантаження;
- WebMagic — Java-фреймворк для масового web-scraping (аналог Scrapy).

Менеджер потоків:

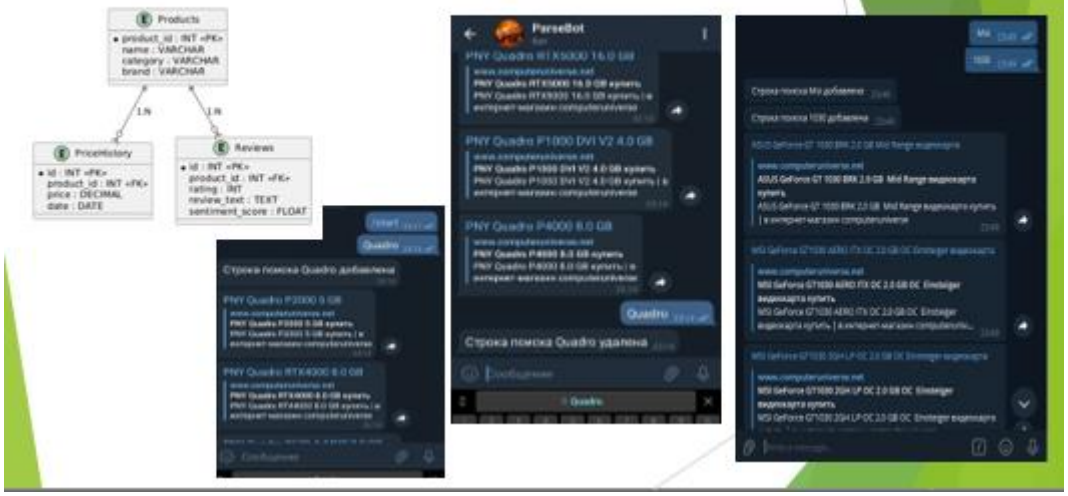
Система побудована на основі Thread Pool- кілька робочих потоків паралельно обробляють задачі

- використовується ExecutorService;
- URL-адреси зберігаються у потокобезпечній черзі BlockingQueue;

Менеджер потоків:

- керує навантаженням;
- забезпечує ефективне використання ресурсів;
- скорочує час збору даних.

Приклад роботи парсеру



Висновок

- В процесі роботи над модулем були алгоритмізовані, та автоматизовані процеси пошуку, та структуризації даних. Було спроектовано, та алгоритмізовано роботу парсеру. В результаті отримано діючу модель автоматизованого аналізу, та структуризації даних веб-додатків.

Апробація

- В процесі роботи над модулем були алгоритмізовані, та автоматизовані процеси пошуку, та структуризації даних. Було спроектовано, та алгоритмізовано роботу парсеру. В результаті отримано діючу модель автоматизованого аналізу, та структуризації даних веб-додатків.