МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА СИСТЕМИ ВЕБСКРАПІНГУ ДЛЯ ВІДСТЕЖЕННЯ ДИНАМІКИ ЦІН НА ТОВАРИ ТА АНАЛІЗУ ДАНИХ З ВИКОРИСТАННЯМ SPRING»**

Виконав: студент 2 курсу, групи 8.1214-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
В.О. Кузьмінов
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Горбенко В.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Борю С.Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Кузьмінову Владиславу Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка системи вебскрапінгу для відстеження динаміки цін на товари та аналізу даних з використанням Spring

керівник роботи Горбенко Віталій Іванович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 28 » лютого 2025 року № 326-с

2. Строк подання студентом роботи 04.12.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Теоретичні основи вебскрапінгу.
2. Проєктування системи.
3. Програмна реалізація системи вебскрапінгу.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

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

7. Дата видачі завдання 07.03.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	27.03.2025	
2.	Збір вихідних даних.	24.04.2025	
3.	Обробка методичних та теоретичних	03.06.2025	
	джерел.		
4.	Розробка першого та другого розділу.	19.08.2025	
5.	Розробка третього розділу.	24.11.2025	
6.	Оформлення та нормоконтроль	01.12.2025	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	18.12.2025	

Студент _____ В.О. Кузьмінов
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.І. Горбенко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка системи вебскрапінгу для відстеження динаміки цін на товари та аналізу даних з використанням Spring»: 80 с., 13 рис., 3 табл., 26 джерел, 4 додатки.

АНАЛІЗ ДАНИХ, БАЗА ДАНИХ, ВЕБСКРАПІНГ, ВІЗУАЛІЗАЦІЯ ДАНИХ, ДИНАМІКА ЦІН, КОНКУРЕНТНИЙ АНАЛІЗ, ЦІНОВА СТРАТЕГІЯ, JAVA, SPRING FRAMEWORK.

Об’єкт дослідження – процеси збору та аналітичного оброблення цінових даних на товари з інтернет-майданчиків для підтримки прийняття маркетингових та управлінських рішень.

Мета роботи: розробити архітектуру та програмну реалізацію автоматизованої системи вебскрапінгу на основі Spring Framework для ефективного моніторингу, аналізу та візуалізації динаміки цін на товари.

Методи дослідження – методи збору та аналізу вимог до програмного забезпечення, методи моделювання, проєктування, конструювання програмного забезпечення.

У роботі представлено автоматизовану систему для збору та аналізу цінових даних з інтернет-майданчиків. Проведено огляд сучасних підходів і технологій вебскрапінгу, визначено переваги та недоліки різних методів отримання даних.

У результаті експериментальної перевірки підтверджено ефективність системи: забезпечено стабільний збір даних із заданих джерел із точністю 99,8%, а автоматизація процесу дозволила скоротити час на моніторинг цін на 95% порівняно з ручним збором. Отримані результати можуть бути використані для розробки комерційних сервісів моніторингу цін, впровадження в бізнес-процеси електронної торгівлі та побудови систем підтримки прийняття рішень в області маркетингу.

SUMMARY

Master's qualifying paper "Development of a Web Scraping System for Tracking Price Dynamics and Data Analysis Using Spring": 80 pages, 13 figures, 3 tables, 26 references, 4 supplements.

DATA ANALYSIS, DATABASE, WEB SCRAPING, DATA VISUALIZATION, PRICE DYNAMICS, COMPETITIVE ANALYSIS, PRICING STRATEGY, JAVA, SPRING FRAMEWORK.

The object of the study is processes of collecting and analytically processing price data for goods from online platforms to support marketing and management decision-making.

The aim of the study is develop the architecture and software implementation of an automated web scraping system based on the Spring Framework for effective monitoring, analysis, and visualization of commodity price dynamics.

The methods of research are collecting and analyzing software requirements, methods of modeling, designing, and constructing software.

The paper presents an automated system for collecting and analyzing price data from online platforms. A review of modern web scraping approaches and technologies is conducted, and the advantages and disadvantages of various data extraction methods are identified.

As a result of experimental testing, the effectiveness of the system was confirmed: stable data collection from specified sources with an accuracy of 99.8% was ensured, and the automation of the process allowed to reduce the time for price monitoring by 95% compared to manual collection. The results obtained can be used to develop commercial price monitoring services, implement them in e-commerce business processes, and build decision support systems in the field of marketing.

ЗМІСТ

Завдання на кваліфікаційну роботу……..…..2
Реферат 4
Summary 5
Вступ..... 8
1 Теоретичні основи вебскрапінгу 10
1.1 Базовий огляд предметної області..... 10
1.2 Дослідження вимог кінцевих споживачів 13
1.3 Аналіз та обробка даних цін 16
1.4 Аналіз існуючих рішень та конкурентів на ринку 19
1.5 Висновки до розділу 1 21
2 Проєктування системи..... 23
2.1 Технічне завдання 23
2.1.1 Область застосування 23
2.1.2 Формування вимог до системи..... 26
2.2 Моделі бізнес-процесів..... 28
2.3 Структура бази даних 30
2.4 Аналіз інструментів для розробки..... 33
2.5 Архітектура програмного забезпечення 36
2.5.1 Аналіз різних шаблонів архітектури програмного забезпечення 36
2.5.2 Архітектура та взаємодія компонентів системи 38
2.6 Проєктування модуля вебскрапінгу..... 43
2.7 Висновки до розділу 2 47
3 Програмна реалізація системи вебскрапінгу..... 48
3.1 Реалізація backend-частини..... 48
3.1.1 Базові налаштування..... 48
3.1.2 Реалізація REST API..... 51
3.1.3 Робота з базою даних..... 53

3.2 Реалізація frontend-частини 54
3.3 Реалізація модуля вебскрапінгу 58
3.4 Висновки до розділу 3 61
Висновки 62
Перелік посилань 64
Додаток А Конфігураційний клас SecurityConfig 67
Додаток Б Файл DashboardPage 69
Додаток В Сервіс ScrapingService 73
Додаток Г Файл SeleniumScraper 78

ВСТУП

Сучасний розвиток технологій вебскрапінгу та аналітики даних відкриває нові можливості для ефективного моніторингу ринкових тенденцій та цінової конкуренції. Однією з ключових задач сучасного користувача є автоматизоване збирання та аналіз цінової інформації з інтернет-майданчиків для прийняття обґрунтованих рішень. Прикладом таких рішень є системи моніторингу цін, такі як Price2Spy, Competera або DataCrops, які дозволяють відстежувати динаміку цін, аналізувати поведінку конкурентів та оптимізувати цінові стратегії.

Метою кваліфікаційної роботи є проєктування та реалізація системи вебскрапінгу для відстеження динаміки цін на товари та аналізу даних з використанням Java та Spring Boot для реалізації серверної частини, React та TypeScript для розробки клієнтського інтерфейсу, Selenium та Jsoup для скрапінгу інформації, а також PostgreSQL як систем зберігання даних.

Для досягнення поставленої мети сформульовано такі задачі:

– проаналізувати сучасні підходи та технології вебскрапінгу та аналітики цінових даних;
– сформулювати функціональні та нефункціональні вимоги до системи моніторингу цін;
– спроєктувати трирівневу архітектуру системи з чітким розмежуванням функціоналу;
– реалізувати основний API та бізнес-логіку на Spring Boot;
– розробити інтерфейс користувача з використанням React та TypeScript;
– налаштувати скрапінг через Selenium та Jsoup;
– організувати зберігання та управління даними в PostgreSQL;
– виконати тестування окремих компонентів.

Структурно робота складається з трьох розділів. У першому розділі розглянуто теоретичні аспекти вебскрапінгу та аналізу цінових даних, а також

проведено огляд існуючих систем моніторингу цін і застосовуваних у них технологічних рішень. Другий розділ присвячено аналізу вимог до системи, проєктуванню її архітектури, моделей взаємодії компонентів і структури бази даних, а також обґрунтуванню вибору технологічного стеку. У третьому розділі наведено опис програмної реалізації основних компонентів системи, інтеграцію модулів.

1 ТЕОРЕТИЧНІ ОСНОВИ ВЕБСКРАПІНГУ

1.1 Базовий огляд предметної області

Сучасна електронна комерція є однією з галузей економіки, що найбільш швидко розвиваються, яка надає значний вплив на повсякденне життя та бізнес-процеси. Сьогодні споживачі мають доступ до величезної кількості товарів та послуг, представлених на онлайн-платформах, що створює високу конкуренцію між продавцями та вимагає від них постійної оптимізації цінових стратегій. У той же час, кінцеві користувачі стикаються з труднощами при пошуку вигідних пропозицій, особливо в умовах високої динамічності цін та величезного асортименту товарів.

Однією з ключових проблем, з якими стикаються учасники ринку, є необхідність оперативного збору та аналізу даних про ціни, знижки та доступність товарів. Для вирішення цього завдання використовуються технології вебскрапінгу.

Теоретично, вебскрапінг – це практика збору даних будь-якими засобами, окрім взаємодії програми з API (або, очевидно, через людину, яка використовує веббраузер). Найчастіше це досягається шляхом написання автоматизованої програми, яка запитує вебсервер, запитує дані (зазвичай у формі HTML та інших файлів, що складають вебсторінки), а потім аналізує ці дані для вилучення необхідної інформації.

На практиці вебскрапінг охоплює широкий спектр методів і технологій програмування, таких як аналіз даних, розбір природної мови та інформаційна безпека [1].

Вебскрапінг (збирання вебданих) – це процес вилучення даних з інтернету, який відповідає певним вимогам. Збір і аналіз даних, їх участь в інформації та прийнятті рішень, а також діяльність, пов’язана з дослідженнями, роблять процес аналізу чутливим для всіх типів промисловості.

Популярність інтернету та його ресурсів призводить до того, що інформаційні домени розвиваються щодня, що також спричиняє зростання попиту на необроблені дані. Дані є основною вимогою в галузі науки, техніки та менеджменту. Зібрані або впорядковані дані обробляються з різним ступенем логіки, щоб отримати інформацію та отримати подальше розуміння.

Коли справа доходить до вебскрейпінгу, універсального рішення не існує.

Вибір інструменту або методу багато в чому залежить від типу сайту, з яким потрібно працювати, особливо якщо це загальнодоступні дані, від способу завантаження даних і від того, який тип вебсканування або отримання даних необхідний [2].

Вебскрапінг надає інструменти та методи, які використовуються для збору даних із вебсайтів відповідно до особистих чи бізнес-потреб, але з урахуванням ряду юридичних міркувань.

Крім того, вебскрепери можуть знайти місця, куди не можуть дістатися традиційні пошукові системи. Пошук у Google за запитом «найдешевші рейси до Бостона» призведе до появи безлічі рекламних оголошень та популярних сайтів пошуку авіаквитків. Google знає лише те, що ці вебсайти пишуть на своїх сторінках із вмістом, а не точні результати різних запитів, введених у програму пошуку авіаквитків. Однак добре розроблений вебскрепер може відобразити вартість авіаквитка до Бостона з плином часу на різних вебсайтах і підказати найкращий час для покупки квитка [1].

Майже все можна витягти з HTML, єдина інформація, яку «важко» витягти, знаходиться всередині зображень або інших медіафайлів.

Ось деякі галузі, де використовується вебскрапінг:

– новинні портали: для агрегації статей з різних джерел даних: reddit, форуми, twitter та окремі новинні вебсайти;
– агентства нерухомості;
– пошукові системи;
– туристична галузь (порівняльники цін на авіаквитки/готелі);

- електронна комерція, для моніторингу цін конкурентів;
- банки: агрегація банківських рахунків (наприклад, Mint та інші подібні програми);
- журналістика: також відома як «журналістика даних»;
- SEO;
- аналіз даних;
- «онлайн-маркетинг на основі даних»;
- маркетингові дослідження [3].

Проте, незважаючи на переваги автоматизації, процес вилучення даних із вебсторінок пов’язаний із низкою складнощів. Однією з ключових проблем є мінливість HTML-структури сайтів, яка потребує постійної адаптації алгоритмів, що використовуються. Крім того, багато сучасних інтернет-ресурсів використовують динамічне завантаження контенту, що значно ускладнює збір даних. Ці виклики формують потребу у створенні адаптивних рішень, які забезпечують стабільність і точність роботи навіть в умовах зовнішніх факторів, що змінюються.

Ще одним важливим аспектом є створення універсальних рішень для автоматизації збору даних. Це передбачає розробку як інструментів для отримання та аналізу даних, а й коштів, дозволяють бізнесу адаптувати систему під свої потреби. Наприклад, використання механізму визначених правил конфігурування скриптів парсингу дає можливість компаніям самостійно керувати процесом вилучення даних, що збільшує гнучкість і знижує залежність від сторонніх рішень.

Автоматизований збір даних з Інтернету майже такий же старий, як і сам Інтернет. Хоча вебскрапінг не є новим терміном, у минулі роки ця практика була більш відома як скрапінг екрану, інтелектуальний аналіз даних, вебхарвестінг або подібні [2].

Таким чином, розробка системи вебскрапінгу, здатної не тільки ефективно збирати дані є актуальним і практично значущим завданням, що поєднує наукові та прикладні аспекти. Такі системи здатні надавати як

кінцевим користувачам, так і бізнесу потужні інструменти для аналізу та візуалізації даних, значно полегшуючи прийняття рішень. Ефективна робота з великими обсягами інформації та застосування сучасних методів обробки відкривають широкі можливості для подальших досліджень та інновацій. В умовах зростаючої значущості даних у цифровій економіці подібні рішення стають важливою складовою трансформації бізнес-процесів.

1.2 Дослідження вимог кінцевих споживачів

У процесі розробки інформаційних систем важливе значення має попереднє дослідження вимог кінцевих споживачів, оскільки саме вони визначають функціональне наповнення, архітектурні рішення та подальшу ефективність використання програмного продукту. Система вебскрапінгу для відстеження динаміки цін на товари та аналізу даних орієнтована на широку категорію користувачів, зацікавлених у систематичному отриманні актуальної цінової інформації з вебресурсів електронної комерції. Умови сучасного цифрового ринку характеризуються високою динамічністю цін, частими акційними пропозиціями та значною конкуренцією між продавцями, що зумовлює необхідність автоматизованих інструментів моніторингу та аналізу.

Кінцевими споживачами такої системи можуть виступати як фізичні особи, які прагнуть оптимізувати власні витрати шляхом відстеження змін цін на конкретні товари, так і представники малого та середнього бізнесу, аналітики ринку, маркетологи, а також дослідники, які використовують зібрані дані для прийняття управлінських або економічних рішень. Незважаючи на різноманітність цільової аудиторії, усі користувачі мають спільну потребу в отриманні достовірної, актуальної та структурованої інформації у зручній для сприйняття формі.

Компанії використовують вебскрейпінг для збору важливої ринкової інформації. Загальнодоступні вебсайти можуть надавати інформацію про

рекламні стратегії конкурентів, ціни на продукцію та відгуки клієнтів, що сприяє поточному уявленню про ринкову ситуацію [4].

Однією з ключових вимог кінцевих споживачів є можливість автоматизованого збору цінових даних після первинного налаштування системи. На відміну від повністю автономних рішень, у межах даного проєкту передбачається напівавтоматичний підхід до вебскрапінгу, який полягає в одноразовому ручному налаштуванні CSS-селекторів для кожного окремого вебсайту. Таке налаштування виконується користувачем системи з урахуванням структури HTML-документа конкретного ресурсу. Після коректного підключення селекторів подальший процес збору інформації про товари здійснюється автоматично без необхідності повторного втручання, що відповідає очікуванням користувачів щодо зменшення рутинних операцій.

Важливою вимогою з боку кінцевих споживачів є чітке розуміння обмежень системи. Зокрема, система вебскрапінгу не передбачає автоматичної адаптації до змін у структурі HTML або CSS вебсторінок. У разі модифікації розмітки сайту користувачеві необхідно повторно налаштувати відповідні селектори для відновлення коректної роботи парсингу. Застосування такого підходу визнано прийнятним. Це зумовлено можливістю зосередження на стабільності роботи системи та спрощенням її архітектури за умови збереження контролю над процесом збирання даних.

З точки зору кінцевих споживачів, важливим є забезпечення коректності та достовірності отриманих даних. Очікується, що система після налаштування селекторів буде точно ідентифікувати необхідні елементи сторінки, зокрема ціну товару, його назву та інші базові характеристики. Помилки у визначенні значень можуть призвести до викривлення історії цін та хибних результатів аналізу, що негативно впливає на практичну цінність системи. Тому кінцеві споживачі зацікавлені у прозорому механізмі налаштування та тестування селекторів перед запуском регулярного збору даних.

Суттєвою вимогою є збереження історії змін цін для кожного відстежуваного товару. Кінцеві споживачі очікують, що система буде

накопичувати дані упродовж тривалого часу та надавати можливість їх подальшого аналізу. Історичні дані дозволяють виявляти закономірності, аналізувати динаміку зростання або зниження вартості, а також оцінювати ефективність цінової політики продавців. Таким чином, збереження та структуроване представлення історії цін є однією з базових функціональних вимог.

Окрему увагу кінцеві споживачі приділяють зручності використання системи. Інтерфейс користувача має бути логічним та інтуїтивно зрозумілим, навіть з урахуванням необхідності початкового технічного налаштування. Процес підключення нового джерела даних повинен бути чітко структурованим, з можливістю перевірки правильності роботи селекторів. Після завершення налаштування користувачі очікують простий доступ до результатів скрапінгу, історії цін та аналітичних візуалізацій у вигляді графіків.

Важливою вимогою є наявність інструментів пошуку, фільтрації та сортування даних. У разі відстеження значної кількості товарів користувачі мають потребу швидко знаходити потрібні позиції за заданими параметрами, зокрема за назвою, діапазоном цін або датою оновлення інформації. Такі можливості значно підвищують ефективність роботи з системою та зменшують час аналізу.

Кінцеві споживачі також зацікавлені у функціональності повідомлень про зміну цін. Можливість отримання сповіщень у разі досягнення заданих умов, наприклад зниження ціни до певного рівня, дозволяє користувачам оперативно реагувати на вигідні пропозиції без постійного ручного контролю.

Важливою є можливість налаштування критеріїв таких повідомлень відповідно до індивідуальних потреб.

З урахуванням багатокористувацького режиму експлуатації система повинна забезпечувати базові механізми безпеки, зокрема реєстрацію та авторизацію користувачів. Кінцеві споживачі очікують, що їхні персональні дані та налаштування будуть захищені від несанкціонованого доступу, а доступ до функціональності системи буде чітко розмежований.

Таким чином, дослідження вимог кінцевих споживачів дозволяє сформувати чітке уявлення про функціональні та нефункціональні характеристики системи вебскрапінгу з напівавтоматичним налаштуванням. Отримані результати є підґрунтям для подальшого проєктування архітектури системи, вибору технологічних засобів та реалізації програмного продукту, який відповідає реальним очікуванням користувачів і має практичну цінність у сфері аналізу цінових даних.

1.3 Аналіз та обробка даних цін

Аналіз та обробка даних цін є одним з ключових етапів функціонування систем вебскрапінгу, оскільки саме на цьому етапі відбувається перетворення зібраної інформації на аналітично значущі дані. У межах системи відстеження динаміки цін на товари з вебресурсів електронної комерції аналіз цінових показників дозволяє не лише фіксувати поточні значення вартості, але й досліджувати їх зміну у часі, виявляти закономірності та оцінювати тенденції ринку.

Візьмемо, наприклад, гіганта електронної комерції Amazon: Amazon використовує великі дані, зібрані з Інтернету, та поведінку своїх клієнтів, щоб оновлювати ціни на свої товари приблизно кожні десять хвилин. Їхнє ціноутворення встановлюється відповідно до загальних тенденцій на ринку, моделей покупок користувачів та бізнес-цілей, серед іншого. Завдяки великим даним Amazon може розумно пропонувати знижки на товари найпопулярнішого ринку та водночас отримувати великий прибуток від менш популярних товарів. Ця стратегія, заснована на даних, виявилася плідною, оскільки вони значно подвоїли свої річні продажі з 2018 по 2021 рік [5].

Збір інформації про ціни в реальному часі з кількох платформ електронної комерції за допомогою технології вебкраулінгу і порівняння. Отримання інформації про ціни реальному часі на платформах електронної

комерції є основним завданням побудови системи порівняння цін. Використовуючи технологію вебкраулінгу, можна збирати ціни на зазначені товари з різних платформ електронної комерції, таких як Taobao, JD.com та Amazon, та проводити порівняльний аналіз. Цей процес зазвичай включає такі кроки: запит вебсторінки, аналіз контенту, отримання інформації про ціни, обробка даних і порівняння [5].

Під час збору та обробки даних необхідно враховувати особливості формування цін на вебсайтах електронної комерції. Зміни вартості можуть відбуватися нерегулярно та залежати від різноманітних факторів, таких як акційні пропозиції, сезонні розпродажі або зміна попиту. У зв’язку з цим система повинна забезпечувати фіксацію даних з визначеною періодичністю, яка дозволяє виявляти як довгострокові тенденції, так і короткострокові коливання. Надто рідкісний збір даних може призвести до втрати важливої інформації, тоді як надто частий до надмірного навантаження на систему.

Одним з основних завдань аналізу є виявлення трендів зміни цін. Під трендом розуміється загальний напрямок руху вартості товару у часі, який може характеризуватися зростанням, зниженням або стабільністю. Аналіз трендів ґрунтується на порівнянні послідовних значень цін та їх узагальненні за певні проміжки часу. Виявлення таких тенденцій дозволяє отримати узагальнене уявлення про поведінку цін та оцінити перспективи їх подальшої зміни.

Окрім аналізу загальних трендів, важливим є дослідження окремих змін вартості товарів. Аналіз змін цін передбачає визначення різниці між поточними та попередніми значеннями, а також оцінку інтенсивності коливань. Значні або різкі зміни можуть свідчити про наявність акцій, знижок або інших факторів, що впливають на цінову політику продавця. Фіксація таких змін є важливою для подальшого інформування користувачів та формування механізмів сповіщення.

Для підвищення точності аналізу здійснюється попередня обробка зібраних даних. Отримана під час вебскрапінгу інформація може містити різні

формати представлення цін, додаткові символи або некоректні значення. Тому необхідним етапом є нормалізація даних, яка передбачає приведення цін до єдиного числового формату, перевірку коректності значень та усунення помилкових записів. Така обробка дозволяє забезпечити коректність подальших аналітичних розрахунків.

Основні операції, що виконуються під час аналізу та обробки цінових даних у системі вебскрапінгу, наведено в таблиці 1.1.

Таблиця 1.1 – Основні етапи аналізу та обробки даних цін

Етап	**Опис**
Збір даних	Отримання цінових значень з вебресурсів із фіксацією часу.
Збереження	Запис отриманих даних у базу даних для формування історії.
Нормалізація	Приведення цін до єдиного формату та перевірка коректності.
Аналіз змін	Визначення різниці між послідовними значеннями цін.
Виявлення трендів	Оцінка загального напряму зміни вартості товарів.
Візуалізація	Подання результатів у вигляді графіків та таблиць.

Результати аналізу цінових даних повинні бути представлені у формі, зручній для сприйняття та подальшого використання. Візуалізація динаміки цін у вигляді графіків дозволяє наочно відобразити зміну вартості товарів у часі та спрощує інтерпретацію результатів. Графічне подання інформації є ефективним інструментом для виявлення тенденцій, а також для порівняння різних періодів та товарів між собою.

Аналіз та обробка даних цін є невід’ємною складовою системи вебскрапінгу, яка забезпечує перехід від простого збору інформації до її аналітичного використання. Накопичення історичних даних, їх нормалізація, виявлення змін створюють основу для подальшого аналізу та прийняття обґрунтованих рішень. Реалізація зазначених процесів у межах програмної системи дозволяє підвищити практичну цінність отриманих даних та забезпечити ефективне використання результатів вебскрапінгу.

1.4 Аналіз існуючих рішень та конкурентів на ринку

Ринок систем моніторингу цін є динамічно розвиваючоюся сферою, де існує широкий спектр рішень – від глобальних SaaS-платформ до локальних спеціалізованих інструментів. Аналіз конкурентних пропозицій дозволяє виявити ключові тенденції, оптимальні архітектурні підходи та потенційні ніші для розроблюваної системи.

Глобальні SaaS-платформи представлені такими лідерами ринку як Price2Spy, Competera та DataCrops. Ці рішення пропонують комплексні набори функцій, включаючи автоматизований скрапінг з тисяч джерел, розширену аналітику цінових трендів, та інтеграцію з електронними торговими платформами. Технологічно вони базуються на мікросервісних архітектурах, використовують хмарні інфраструктури (AWS, Google Cloud) та підтримують REST API для інтеграції. Їхніми основними перевагами є масштабованість, багатомовність та глибина аналітики, а недоліками – висока вартість ліцензій та складність адаптації до специфіки локальних ринків.

Регіональні та нішеві рішення включають платформи, орієнтовані на конкретні географічні ринки або галузі. На українському ринку представлені такі сервіси як Price.ua, Hotline та Prom.ua, які поєднують функції порівняння цін з можливостями електронної торгівлі. Ці системи відрізняються глибоким розумінням локальних особливостей, підтримкою україномовного інтерфейсу та інтеграцією з місцевими платіжними системами. Технологічно вони часто використовують гібридні архітектури з елементами монолітів та мікросервісів, що дозволяє швидко адаптуватися до змін на ринку.

Відкриті та саморобні рішення представлені Open Source проєктам а саме Scrapy, Beautiful Soup та Selenium, які дозволяють організаціям будувати власні системи скрапінгу. Перевагою цих рішень є гнучкість, відсутність ліцензійних витрат та можливість тонкої настройки під конкретні потреби.

Недоліками є високі вимоги до технічної експертизи, необхідність самостійної підтримки інфраструктури та обмежені можливості аналітики в порівнянні з комерційними платформами.

Технологічні підходи конкурентів можна класифікувати за кількома критеріями:

– архітектура скрапінгу: одні системи використовують централізовані сервери парсингу, інші – розподілені мережі проксі-серверів для обходу геоблокувань;
– обробка даних: аналіз та обробка отриманих даних про ціни;
– інтерфейси користувача: вебпанелі, мобільні додатки, Telegram-боти, API для інтеграції;
– моделі монетизації: платні підписки, pay-per-use, фріміум з обмеженим функціоналом.

Ключові конкурентні переваги, що спостерігаються на ринку:

– точність та актуальність даних: системи-лідери забезпечують оновлення цін з частотою до кількох хвилин;
– широта покриття: підтримка сотень тисяч джерел даних з різних країн та мов;
– глибина аналітики: розширені звіти, прогнозні моделі, інтеграція з бізнес-інструментами;
– зручність інтерфейсів: інтуїтивні dashboards, персоналізовані сповіщення, мобільний доступ;
– масштабованість: можливість обробки мільйонів товарних позицій без втрати продуктивності.

Технічні виклики, з якими стикаються існуючі рішення:

– обхід анти-скрапінг захистів: CAPTCHA, JavaScript-енкодінг, динамічні селектори;
– обробка великих даних: ефективне зберігання та аналіз терабайтів історичних цін;
– забезпечення стабільності: мінімізація downtime при змінах структури цільових сайтів;
– міжнародна підтримка: робота з різними мовами, валютами, часовими поясами.

Висновки аналізу дозволяють сформулювати конкурентні переваги розроблюваної системи:

– фокус на українському ринку з глибоким розумінням локальних особливостей;
– гібридна архітектура що поєднує переваги мікросервісів з простотою моноліту;
– інноваційні інтерфейси з акцентом на Telegram-бот та мобільний досвід;
– гнучка модель ціноутворення доступна для малого та середнього бізнесу;
– відкриті API для легкої інтеграції з існуючими бізнес-процесами.

Цей аналіз забезпечує розуміння конкурентного ландшафту та дозволяє сформулювати унікальні пропозиції цінності для розроблюваної системи моніторингу цін, що забезпечить її конкурентоспроможність на ринку.

1.5 Висновки до розділу 1

У першому розділі було здійснено комплексний аналіз предметної області, пов’язаної з використанням вебскрапінгу для моніторингу та аналізу цін у сфері електронної комерції. Розглянуто сучасний стан розвитку e-commerce, який характеризується високою динамічністю цін, значною конкуренцією між продавцями та зростаючою роллю даних у процесі прийняття управлінських рішень. Встановлено, що в умовах великого обсягу товарних пропозицій та частих змін вартості ручний аналіз цін є малоефективним, що зумовлює необхідність застосування автоматизованих засобів збору та обробки інформації.

У ході базового огляду предметної області визначено сутність вебскрапінгу як технології автоматизованого вилучення структурованих і напівструктурованих даних із вебресурсів. Проаналізовано теоретичні засади

та практичні аспекти застосування вебскрапінгу, а також основні сфери його використання, зокрема електронну комерцію, маркетингові дослідження, журналістику даних та аналітику ринку. Показано, що вебскрапінг є універсальним інструментом для отримання актуальної інформації з відкритих джерел, проте його ефективність значною мірою залежить від стабільності структури вебсторінок і правильно обраних методів збору даних.

У межах дослідження вимог кінцевих споживачів встановлено, що потенційними користувачами системи моніторингу цін можуть бути як приватні особи, так і представники бізнесу, аналітики та дослідники. Незважаючи на різноманітність цільової аудиторії, основними вимогами є автоматизований збір даних після первинного налаштування, достовірність і точність отриманої інформації, збереження історії змін цін, а також зручний та інтуїтивно зрозумілий інтерфейс. Обґрунтовано доцільність напівавтоматичного підходу до вебскрапінгу, що передбачає ручне налаштування CSS-селекторів з подальшою автоматизацією процесу збору даних, що дозволяє зберегти гнучкість системи та спростити її архітектуру.

Окрему увагу приділено аналізу та обробці цінових даних, які є ключовим етапом функціонування системи. Визначено основні операції обробки даних, зокрема нормалізацію, збереження історичних значень, аналіз змін та виявлення трендів. Показано, що накопичення та аналітична обробка історичних даних про ціни дозволяють виявляти закономірності ринку, оцінювати ефективність цінової політики та формувати підґрунтя для прийняття обґрунтованих рішень.

У результаті аналізу існуючих рішень та конкурентів на ринку встановлено, що сучасні комерційні платформи забезпечують високий рівень автоматизації та аналітики, проте характеризуються високою вартістю та складністю адаптації до локальних умов. Водночас відкриті та саморобні рішення надають значну гнучкість, але вимагають високої технічної кваліфікації. Це підтверджує актуальність розробки власної системи моніторингу цін, орієнтованої на локальний ринок, з оптимальним поєднанням функціональності, простоти використання та доступності.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Технічне завдання

2.1.1 Область застосування

Розробка систем для автоматизованого збору, обробки та аналізу даних про ціни на товари з інтернет-магазинів має широкий спектр застосування у сучасних умовах цифрової економіки. Основними напрямками використання таких систем є галузі електронної комерції, маркетингової аналітики, конкурентного моніторингу, стратегічного планування та персоналізованого сервісу для споживачів.

Це спростило б спосіб життя маркетологів та продавців, роздрібних торговців та споживачів, оскільки це надає прості платформи для продажу замовлення товарів через Інтернет. Це також означає, що користувачі не витрачатимуть багато часу та зусиль на пошук найкращих пропозицій товарів, оновлень товарів та акцій на вебсайтах електронної комерції. Вони повинні самостійно фільтрувати та порівнювати результати пошуку, що займає багато часу та існує ймовірність неоднозначних результатів [6].

Моніторинг конкурентів, також відомий як конкурентна розвідка, передбачає систематичне відстеження та аналіз дій, стратегій і позиціонування конкурентів на ринку. Цей процес допомагає підприємствам зрозуміти сильні та слабкі сторони, можливості та загрози, які створюють їхні конкуренти.

Це відкриває нові захопливі можливості для емпіричних економічних та бізнес-досліджень, але також породжує нові статистичні питання та виклики. Оцінка оптимальності цінової дискримінації в індустрії програмного забезпечення з використанням великого набору панельних даних електронної комерції, зібраних з сайтів. Це включає метод «зворотного проєктування» фактичних рівнів попиту на основі рейтингів продажів, про які повідомляє

Amazon, а також підходи до оцінки еластичності попиту, змінних витрат та оптимальності цінових рішень безпосередньо з загальнодоступних даних електронної комерції. Аналіз ставить багато нових викликів для статистичного аналізу даних електронної комерції [7].

Відстежуючи конкурентів, компанії можуть отримати уявлення про цінові стратегії, запуски продуктів, маркетингові кампанії, відгуки клієнтів і загальні тенденції ринку. Ця інформація є безцінною для прийняття обґрунтованих рішень, випередження на ринку та адаптації до змін у конкурентному середовищі. По суті, моніторинг конкурентів дозволяє підприємствам виявляти прогалини на ринку, передбачати зміни в галузі та розробляти ефективніші стратегії для підтримки конкурентної переваги.

Моніторинг конкурентів – це в основному SWOT-аналіз конкурента електронної комерції.

SWOT-аналіз – це інструмент стратегічного планування, який керівники проєктів можуть використовувати для оцінки сильних і слабких сторін, можливостей і загроз проєкту. Він передбачає визначення мети проєкту та визначення внутрішніх і зовнішніх факторів, які сприяють і несприятливі для досягнення цієї мети. Сильні та слабкі сторони зазвичай виникають всередині організації, а можливості та загрози – із зовнішніх джерел [8].

Це дає знати про кожен крок конкурента, особливо щодо його маркетингових стратегій, цінової політики та запасів:

– основна мета бізнесу – максимізація прибутку та отримання найбільшої частки ринку;
– для бізнесу дуже важливо стежити за стратегіями своїх конкурентів;
– комплексний аналіз допоможе залишатися на вершині конкурентів.

На конкурентному ринку бізнес повинен приймати зміни, щоб вижити. Бізнес прагне задовольнити потреби споживачів і хоче робити це ефективніше, ніж будь-який інший конкурент. Недооцінка або ігнорування конкуренції може втрачати бізнес. Моніторинг конкуренції є важливою частиною загальної маркетингової стратегії. Моніторинг конкуренції надає оперативну

та доречну інформацію про конкурентів за короткий час, покращує реакцію бізнесу та дає краще знання про потенційні загрози як від вже сформованої конкуренції, так і від нових підприємств.

Моніторинг конкурентів передбачає здійснення систематичного контролю за низкою ключових параметрів.

Стратегії ціноутворення. Це допоможе підприємствам використовувати ці дані на основі конкуренції. Саме ціни, встановлені конкурентами, мають великий вплив на ціни бізнесу. Підприємство може або використовувати ціноутворення за поточним курсом, тобто вибирати ціну в цілому відповідно до конкуренції. Або бізнес може використовувати стратегію руйнівного ціноутворення, яка передбачає зниження цін протягом тривалого часу, щоб вигнати завершення.

Маркетингові стратегії. Допомагають відповідати на конкурентні маркетингові кампанії своїми власними ініціативами. Можна використовувати ці знання для створення маркетингових стратегій, які враховують слабкі сторони конкурентів і підвищують ефективність бізнесу.

Стратегічне планування у ритейлі та інших галузях також виграє від використання таких систем. Зібрані дані є основою для прогнозування майбутніх змін у попиті та цінах, а також для аналізу впливу зовнішніх факторів, таких як економічна ситуація чи зміни у законодавстві. Це дозволяє бізнесу мінімізувати ризики, пов’язані з коливаннями ринку та приймати обґрунтовані рішення.

Крім того, системи автоматизації збору даних знаходять застосування у створенні персоналізованих рішень для кінцевих споживачів. Наприклад, повідомлення про знижки та вигідні пропозиції, засновані на індивідуальних уподобаннях користувачів, підвищують їхню задоволеність і лояльність. Такі рішення також використовуються в сервісах рекомендацій, де користувачам пропонують товари, що відповідають їх інтересам.

Таким чином, сфера застосування систем автоматизованого збору та аналізу даних охоплює як кінцевих споживачів, так і бізнес-організації,

надаючи їм можливості для покращення процесів прийняття рішень, оптимізації стратегій та підвищення конкурентоспроможності. Інтеграція таких рішень у повсякденну практику сприяє прискоренню цифрової трансформації та адаптації компаній до умов ринку, що швидко змінюються.

2.1.2 Формування вимог до системи

Визначення вимог до системи є одним із ключових етапів процесу проєктування програмного забезпечення, оскільки саме на цьому етапі формалізуються очікування щодо її функціональності, поведінки та якісних характеристик. Система вебскрапінгу для відстеження динаміки цін на товари та аналізу даних повинна відповідати визначеним вимогам з урахуванням особливостей предметної області, обраного напівавтоматичного підходу до збору даних, а також умов подальшої експлуатації. Сукупність вимог дозволяє забезпечити відповідність розроблюваної системи реальним потребам користувачів та створює основу для прийняття архітектурних і технологічних рішень. Вимоги до системи доцільно поділяти на функціональні та нефункціональні.

Функціональні вимоги визначають конкретні функції та операції, які система повинна виконувати для задоволення потреб бізнесу та користувачів. Вони описують, що система повинна робити та як вона повинна взаємодіяти з користувачами або іншими системами [9].

Нефункціональні вимоги, або NFR, – це набір специфікацій, що описують операційні можливості та обмеження системи. Це, по суті, вимоги, які визначають, наскільки добре вона працює, включаючи такі речі, як швидкість, безпека, надійність, цілісність даних тощо [10].

З урахуванням поставлених завдань до основних функціональних вимог системи належать:

– забезпечення можливості реєстрації та авторизації користувачів для доступу до функціональності системи;

- реалізація керування джерелами даних, зокрема додавання, редагування та видалення вебсайтів для подальшого скрапінгу;
- надання інструментів для одноразового ручного налаштування CSS-селекторів для кожного окремого вебсайту з метою визначення елементів, що містять необхідну інформацію;
- автоматизований збір даних після завершення початкового налаштування селекторів без необхідності повторного втручання користувача;
- збереження отриманих даних у базі даних із фіксацією часових міток для формування історії змін цін;
- забезпечення можливості перегляду історії цін для обраних товарів;
- надання засобів пошуку, фільтрації та сортування товарів за визначеними параметрами;
- візуалізація результатів аналізу у вигляді графіків;
- реалізація механізму повідомлень про зміну цін відповідно до заданих користувачем критеріїв.

Нефункціональні вимоги визначають якісні характеристики системи та обмеження, що впливають на спосіб її реалізації та експлуатації. Вони є важливими для забезпечення стабільної, безпечної та зручної роботи системи у процесі її використання.

До основних нефункціональних вимог системи вебскрапінгу належать:

- забезпечення достатнього рівня продуктивності для обробки запитів користувачів та аналізу накопичених даних без значних затримок;
- стабільність роботи системи під час тривалої експлуатації та при збільшенні обсягу зібраних даних;
- захист персональних даних користувачів і забезпечення безпечного зберігання облікових даних;
- зручність використання інтерфейсу, що забезпечує інтуїтивно зрозумілу взаємодію навіть за наявності етапу початкового технічного налаштування;

– масштабованість архітектури з урахуванням можливого розширення функціональності у майбутньому;
– підтримуваність та зрозумілість програмного коду для спрощення супроводу та модифікації системи.

Сформульовані функціональні та нефункціональні вимоги визначають загальні рамки проєктування та реалізації системи. Вони забезпечують системний підхід до розробки програмного продукту та слугують основою для подальшого проєктування архітектури, моделі даних і програмної реалізації системи.

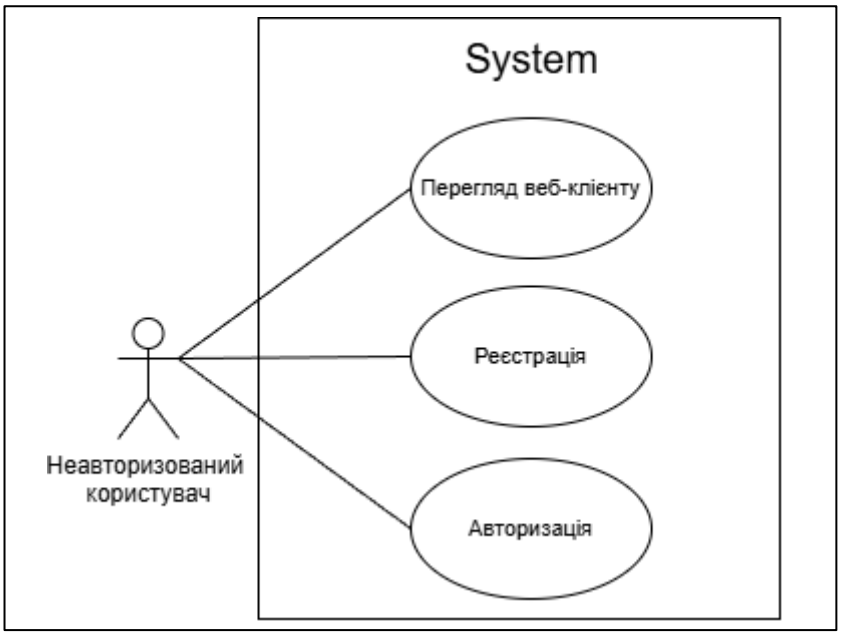
2.2 Моделі бізнес-процесів

Уніфікована мова моделювання (UML) дозволяє описувати системи словами та зображеннями. Примітними є різні графічні діаграми, особливо така як діаграма варіантів використання [11]. Застосування стандартних нотацій UML забезпечує однозначність інтерпретації проєктних рішень усіма учасниками розробки.

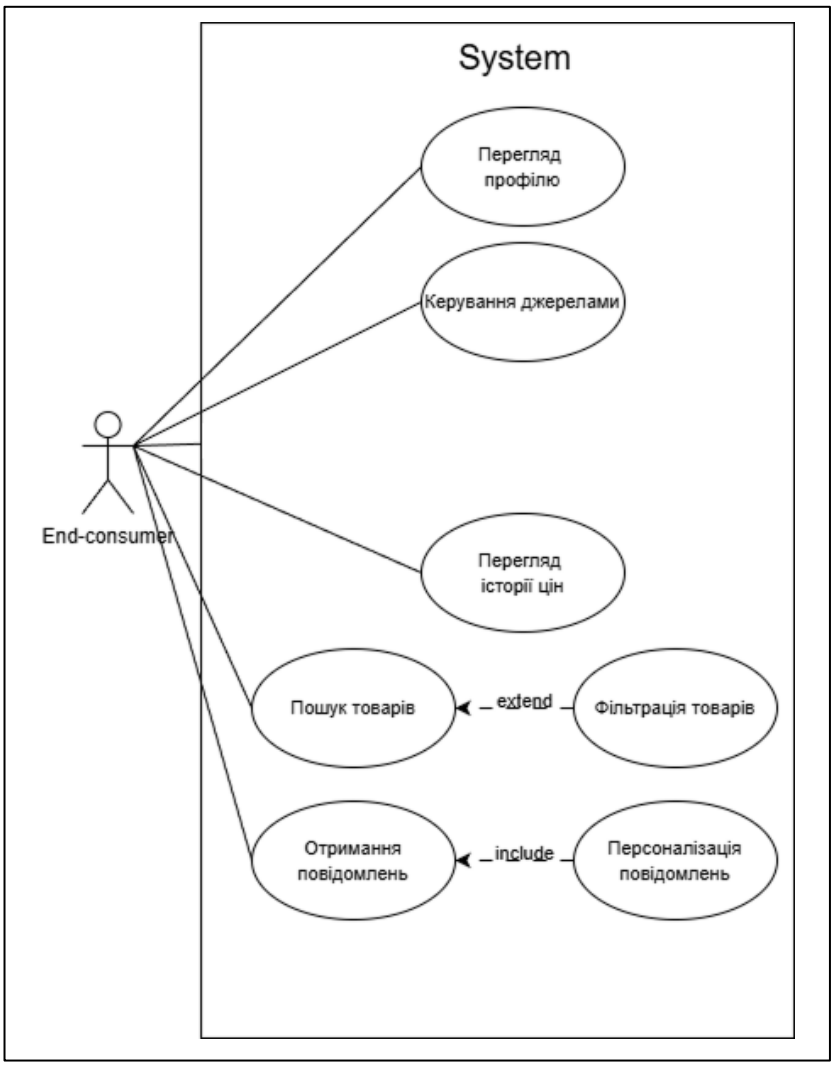
Використання діаграм прецедентів дозволяє формалізувати вимоги до системи через опис взаємодії зовнішніх акторів із програмними функціями. Проєктування системи є ключовим етапом розробки, на якому визначаються функціональні можливості системи, її архітектура та основні компоненти. Таким чином, створюється теоретичне підґрунтя для подальшої технічної реалізації та верифікації програмного продукту.

На рисунку 2.1 зображено діаграма прецедентів, яка ілюструє основні дії, які виконуються різними типами користувачів у системі.

Діаграма прецедентів (use case diagram) відображає взаємодію двох типів користувачів із системою: неавторизований користувач та кінцевий користувач. Розглядаються основні функції та взаємодії для кожного з цих учасників.



а) Неавторизований користувач



б) Кінцевий споживач

Рисунок 2.1 – Діаграма прецедентів

У таблиці 2.1 представлені актори та прецеденти діаграми прецедентів.

Таблиця 2.1 – Актори та прецеденти

Актор	**Прецедент**
Неавторизований користувач	– перегляд вебклієнту: можливість ознайомлення із головною сторінкою системи; – реєстрація: процес створення облікового запису для отримання доступу до системи; – авторизація: процес входу в систему з використанням облікових даних.
Кінцевий споживач	– перегляд профілю: можливість переглянути дані, введені користувачем при реєстрації; – керування джерелами: можливість додавати, редагувати та видаляти список вебсайтів для парсингу; – перегляд історії цін: отримання історії зміни цін на вибрані товари; – пошук товарів: можливість пошуку та вибору товарів необхідних для парсингу; – фільтрація товарів: можливість фільтрувати товари за параметрами, а також сортувати за перевагами; – отримання повідомлень: можливість отримувати повідомлення про знижки на відповідній сторінці; – персоналізація повідомлень: можливість налаштувати критерії повідомлень.

Діаграма прецедентів є важливим інструментом проєктування, який допомагає визначити основні функції та взаємодії в системі. Аналізуючи цю діаграму, можна чітко уявити ролі та дії різних типів користувачів, що є основою для подальшої розробки системи та її компонентів.

2.3 Структура бази даних

Структура бази даних є фундаментальною частиною будь-якої інформаційної системи, що забезпечує зберігання, керування та доступ до даних.

У цьому проєкті використовується реляційна база даних, спроєктована за допомогою ER-діаграми (діаграми «сутність-зв’язок»), що ілюструє логічні зв’язки між різними компонентами системи.

На етапі концептуального моделювання здійснюється ідентифікація ключових сутностей, таких як користувачі, проєкти та завдання, а також визначення їхніх атрибутів. У контексті реляційних баз даних зв’язок між двома таблицями має дві потужності: потужність – кількість елементів – пов’язаних з першою таблицею та потужність – кількість елементів – пов’язаних з другою таблицею [9].

Для забезпечення логічної цілісності на рівні схеми впроваджуються обмеження цілісності, що включають перевірку унікальності та каскадні операції при зміні пов’язаних записів. Логічна модель бази даних побудована з урахуванням вимог до системи та особливостей предметної області.

Важливим аспектом проєктування є нормалізація таблиць, що дозволяє усунути аномалії оновлення та забезпечити оптимальне використання дискового простору. В основі моделі лежить реляційний підхід, що передбачає подання даних у вигляді взаємопов’язаних таблиць із чітко визначеними первинними та зовнішніми ключами. Такий підхід забезпечує цілісність даних, зменшує надлишковість інформації та спрощує виконання аналітичних запитів.

ER-діаграма відображає основні сутності системи та зв’язки між ними. Вона дозволяє наочно представити структуру бази даних, визначити характер взаємодії між таблицями та обґрунтувати вибір атрибутів кожної сутності. На діаграмі показано зв’язки типу «один до одного», «один до багатьох» та «багато до багатьох», що відповідають реальним сценаріям використання системи, зокрема взаємодії користувачів із товарами та джерелами скрапінгу.

На рисунку 2.2 зображена ER-діаграма. Основними сутностями бази даних є користувачі, ролі, джерела даних, товари, історія цін та повідомлення. Сутність користувача призначена для зберігання облікових даних і персональної інформації, необхідної для авторизації та персоналізації роботи

системи. Для реалізації механізму розмежування доступу використовується окрема сутність ролей, пов’язана з користувачами через проміжну таблицю, що дозволяє гнучко керувати правами доступу.

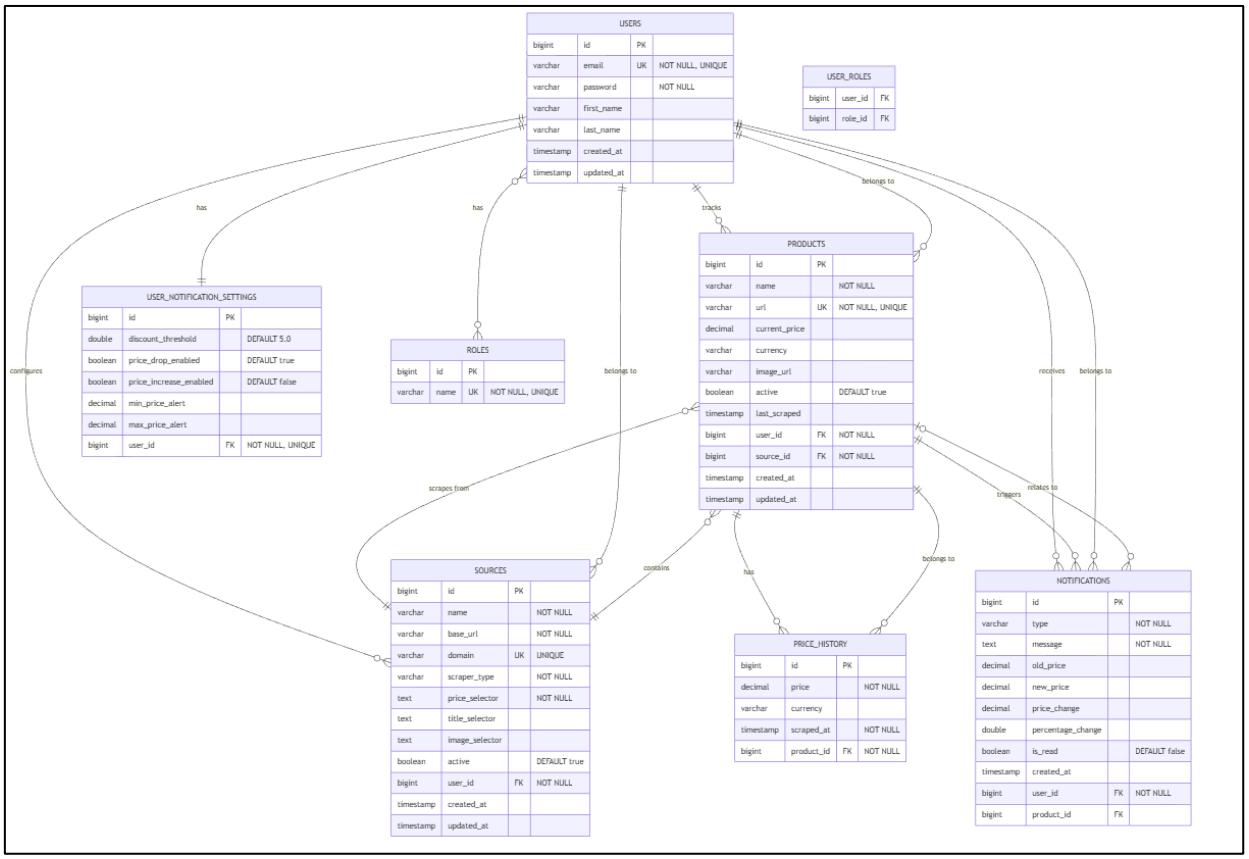


Рисунок 2.2 – ER-діаграма

Сутність джерел даних відображає вебсайти, з яких здійснюється скрапінг інформації. Для кожного джерела зберігаються параметри, необхідні для напівавтоматичного збору даних, зокрема базова адреса сайту, тип скрапера та CSS-селектори. Такий підхід відповідає обраному принципу одноразового ручного налаштування структури сторінки для кожного вебресурсу. Сутність товарів пов’язана з конкретним джерелом і містить інформацію про поточну ціну, валюту, посилання на сторінку товару та часові мітки останнього оновлення.

Для забезпечення аналізу динаміки цін у системі передбачено сутність історії цін, яка зберігає всі зафіксовані значення вартості товару з прив’язкою до часу збору даних. Це дозволяє будувати графіки, визначати тенденції та

виконувати порівняльний аналіз. Окрема сутність повідомлень використовується для реалізації механізму сповіщень користувачів про зміни цін відповідно до заданих правил. Повідомлення пов’язуються як із користувачем, так і з конкретним товаром, що забезпечує персоналізований характер сповіщень.

У таблиці 2.2 представлено узагальнену характеристику основних сутностей бази даних.

Таблиця 2.2 – Основні сутності бази даних

Назва сутності	**Призначення**
Users	Зберігання облікових та персональних даних користувачів.
Roles	Визначення ролей та рівнів доступу в системі.
Sources	Опис вебсайтів і параметрів скрапінгу.
Products	Зберігання інформації про товари та їх поточні ціни.
Price_History	Фіксація історії змін цін у часі.
Notifications	Зберігання повідомлень про зміну цін.

Спроєктована база даних забезпечує повну підтримку функціональних можливостей системи вебскрапінгу. Запропонована структура є логічно узгодженою, розширюваною та придатною для подальшого розвитку системи в межах обраної архітектури та технологічного стеку.

2.4 Аналіз інструментів для розробки

Аналіз інструментів для розробки є важливим етапом у створенні системи вебскрапінгу для відстеження динаміки цін на товари. Цей процес включає вивчення різних технологій, фреймворків та інструментів, які будуть використані для розробки та реалізації функціональності системи. Мета даного аналізу полягає у виборі оптимального набору інструментів, який дозволить ефективно реалізувати завдання проєкту з огляду на його особливості та вимоги. У ході аналізу проводиться оцінка різних інструментів,

їх відповідності поставленим цілям та вимогам, а також їх придатності для вирішення конкретних завдань проєкту.

Java – це мова програмування та платформа розробки. Вона продовжує залишатися платформою розробки, яку обирають підприємства та розробники [13]. Java вибрано як основну мову розробки серверної частини завдяки її продуктивності та стабільності. У проєкті Java використовуватиметься для реалізації ключових модулів системи, включаючи обробку запитів користувачів, керування даними та виконання бізнес-логіки.

Spring Boot допомагає створювати автономні, готові до використання у виробничому середовищі програми на основі Spring. Більшість програм Spring Boot потрібне мінімальне налаштування Spring [14]. Він буде застосовуватися для побудови серверної архітектури та розробки RESTful API. У цьому проєкті він забезпечить зв’язок між фронтендом та бекендом, а також реалізує безпеку та управління даними.

Apache Maven – це інструмент збирання для Java-проєктів. Використовуючи об’єктну модель проєкту (POM), Maven управляє компіляцією, тестуванням та документуванням проєкту [15]. Maven буде використовуватися для управління залежностями та автоматизації складання проєкту. Це спростить інтеграцію різних компонентів, які у системі.

JUnit 5 та Mockito будуть застосовуватися для написання та виконання модульних тестів. Вони забезпечать перевірку коректності роботи серверної частини проєкту, що особливо важливо задля забезпечення надійності системи.

Selenium – це комплексний проєкт, що поєднує низку інструментів та бібліотек, які дозволяють автоматизувати роботу веббраузерів та підтримувати цю автоматизацію. В основі Selenium лежить WebDriver – інтерфейс для написання наборів інструкцій, які можуть виконуватися в багатьох браузерах [16]. Для виконання завдань із синтаксичного аналізу даних передбачено використання бібліотеки Selenium як основного інструмента взаємодії з вебконтентом.

Jsoup – це Java-бібліотека, що спрощує роботу з реальним HTML та XML. Вона пропонує простий у використанні API для отримання URL-адрес, аналізу, вилучення та обробки даних за допомогою методів DOM API, CSS та селекторів XPath. Jsoup реалізує специфікацію WHATWG HTML5 і перетворює HTML-код на той же DOM, що і сучасні браузери [17]. Для реалізації функціональних можливостей зі збору та структурування даних застосовується бібліотека Jsoup.

Hibernate робить реляційні дані видимими для програми, написаної Java, в природній і типізованою формі, що спрощує написання складних запитів та роботу з їх результатами [18]. Hibernate буде використовуватися як ORM-інструмент для взаємодії з базою даних PostgreSQL. Це забезпечить зберігання структурованих даних із високим ступенем надійності.

PostgreSQL – це потужна об’єктно-реляційна система баз даних з відкритим вихідним кодом, яка використовує та розширює мову SQL у поєднанні з безліччю функцій, що забезпечують безпечне зберігання та масштабування найскладніших робочих навантажень з даними [19]. PostgreSQL стане основною базою даних проєкту. Вона використовуватиметься для зберігання інформації про товари, ціни, знижки, а також дані, необхідні для аналізу.

TypeScript – це строго типізована мова програмування, що базується на JavaScript. TypeScript додає додатковий синтаксис до JavaScript для більш тісної інтеграції з редактором [20]. Використовуватиметься для програмування фронтенд частини програми.

React.js – це бібліотека для мови програмування JavaScript з відкритим вихідним кодом для розробки інтерфейсів користувача [21]. Вона допоможе швидко і легко реалізувати реактивність – явище, коли у відповідь на зміну одного елемента змінюється інше.

Docker – це відкрита платформа для розробки, розповсюдження та запуску програм. Docker дозволяє відокремити програми від інфраструктури, що забезпечує швидку доставку програмного забезпечення. Використання Docker дозволяє застосовувати до управління інфраструктурою ті самі підходи

та методи, що й до розробки програмного забезпечення [22]. Docker буде використовуватися для контейнеризації програм, що спростить їх розгортання та керування. Усі компоненти системи будуть ізольовані у контейнерах для забезпечення стабільності та спрощення оновлень.

Проведений аналіз інструментів для розробки вебдодатку підтверджує, що обраний стек технологій є обґрунтованим та ефективним для досягнення поставленої мети. Перш за все, враховується можливість створення гнучкої архітектури, що здатна забезпечити високу продуктивність при обробці великих масивів даних. Застосування модульного принципу побудови системи сприяє легкості супроводження програмного продукту та спрощує процес його модернізації. Також забезпечується відповідність сучасним стандартам безпеки передачі та зберігання конфіденційної інформації користувачів. Загалом, обраний стек технологій забезпечує необхідну функціональність та інструменти для створення сучасного та професійного вебдодатку, здатного ефективно вирішувати поставлені завдання.

2.5 Архітектура програмного забезпечення

2.5.1 Аналіз різних шаблонів архітектури програмного забезпечення

Одним із найважливіших етапів є вибір відповідної архітектури для програмного забезпечення [23].

Архітектура програмного забезпечення – це структура структур інформаційної системи, що складається з сутностей і їхніх зовнішніх видимих властивостей, а також зв’язків між ними. Також це опис підсистем і компонентів програмної системи та зв’язків між ними.

Підсистеми та компоненти зазвичай вказуються в різних представленнях, щоб показати відповідні функціональні та нефункціональні властивості програмної системи. Програмна система є артефактом. Це результат діяльності з розробки програмного забезпечення.

Архітектура програмного забезпечення є основою будь-якої успішної програми. Вона визначає структуру, організацію та різні взаємодії компонентів системи, які зазвичай мають кінцевий вплив на продуктивність, масштабованість, зручність обслуговування та безпеку. Вибір правильної архітектури полягає не лише в поставленні галочок, а й у виборі плану, який відповідає унікальним потребам і цілям проєкту [23].

Вибір правильного шаблону архітектури програмного забезпечення є найважливішим етапом розробки складних і масштабованих систем. Архітектурні рішення визначають структуру системи, взаємодію її компонентів, і навіть її гнучкість і продуктивність під час вирішення поставлених завдань. На етапі проєктування необхідно враховувати потенційні вимоги до системи, такі як обробка великих обсягів даних, підтримка багатопоточності, можливість адаптації до умов, що змінюються, і зручність масштабування. Різні архітектурні підходи пропонують унікальні можливості та обмеження, що потребує ретельного аналізу для обґрунтованого вибору найбільш відповідного шаблону. Розгляд та порівняння популярних архітектурних стилів дозволяє закласти міцний фундамент для розробки програмного рішення, що відповідає цілям та очікуванням проєкту.

Розглянемо основні патерни.

Монолітна архітектура: все в одному блоці. Простота та легкість розробки невеликих, чітко визначених проєктів. Найбільша проблема може виникнути при масштабуванні та обслуговуванні складних програм.

Архітектура мікросервісів: тут є розбивка програми на незалежні керовані служби. Це сприяє гнучкость масштабованості, гнучкості, незалежному розвитку та командній співпраці. Основним недоліком є той факт, що це додає складності, коли справа доходить до оркестровки та комунікації.

Архітектура клієнт-сервер: у цій архітектурі є клієнт (інтерфейс користувача), відокремлений від обробки даних (сервер). Підтримуються

масштабованість і централізоване керування, і всі зміни на стороні сервера впливають на всіх клієнтів.

Багаторівнева архітектура: тут система складається з 4 різних рівнів. Це включає рівні презентації, бізнес-логіки, стійкості та доступу до даних. Хоча підтримується ремонтопридатність, можуть виникнути деякі проблеми щодо складності міжрівневого зв’язку.

Архітектура, керована подіями: тут компоненти реагують на події, а не на прямі виклики, що забезпечує слабкий зв’язок і реакцію в реальному часі. Це відмінно підходить для розподілених систем, але проблеми з налагодженням і трасуванням можуть бути складнішими.

Безсерверна архітектура: тут дозволено створювати та запускати програми та служби без керування інфраструктурою. Це економічно вигідно та автоматично масштабується. Деякі проблеми можуть призвести до блокування постачальника та мають обмеження щодо локальної розробки та налагодження.

Вибір відповідного архітектурного патерну для системи – це терміновий процес, що вимагає врахування багатьох факторів, тому слід враховувати необхідні процедури.

2.5.2 Архітектура та взаємодія компонентів системи

Загальна архітектура системи вебскрапінгу для відстеження динаміки цін на товари та аналізу даних з використанням Spring побудована на основі трирівневої архітектури з чітким поділом на клієнтську та серверну частини з урахуванням принципів модульності, масштабованості та чіткого розділення відповідальностей між компонентами.

Трирівнева архітектура – це добре зарекомендувала себе архітектура програмних додатків, яка організує додатки в три логічні та фізичні обчислювальні рівні:

– рівень уявлення, або інтерфейс користувача;
– рівень програми, де обробляються дані;
– рівень даних, де дані програми зберігаються та управляються [24].

Застосування такого підходу дозволяє мінімізувати взаємозалежність між окремими рівнями, що спрощує процес оновлення та технічного обслуговування системи. На рівні обробки даних забезпечується виконання алгоритмів вилучення та первинної фільтрації інформації, отриманої з зовнішніх джерел. Взаємодія між структурними елементами здійснюється за допомогою стандартизованих протоколів.

Архітектурне рішення орієнтоване на клієнт-серверну модель із використанням сучасних вебтехнологій, контейнеризації та REST-підходу до організації взаємодії між підсистемами. На рисунку 2.3 зображено загальну архітектуру проєкту.

Клієнтська частина системи реалізована у вигляді вебзастосунку на основі React та TypeScript і використовується для взаємодії користувача з функціональністю системи. Обслуговування фронтенд-застосунку та маршрутизація запитів здійснюються за допомогою вебсервера Nginx, який також виконує функцію зворотного проксі, перенаправляючи API-запити до серверної частини. Такий підхід дозволяє відокремити інтерфейс користувача від серверної логіки, спростити мережеву конфігурацію та підвищити рівень безпеки.

Серверний рівень реалізовано з використанням фреймворку Spring Boot і призначено для обробки REST-запитів, реалізації бізнес-логіки, автентифікації користувачів та керування даними. У межах серверної частини функціонує модуль вебскрапінгу, який підтримує напівавтоматичний підхід до збору даних. Для статичних вебсторінок використовується бібліотека Jsoup, а для динамічних ресурсів – Selenium із віддаленим WebDriver. Вибір механізму збору даних визначається параметрами джерела, що зберігаються в базі даних, а запуск скрапінгу може здійснюватися як за розкладом, так і вручну через API. Використання багатопотоковості під час збору даних

дозволяє підвищити пропускну здатність системи та скоротити час обробки черги запитів. Вважається, що динамічне керування параметрами парсингу забезпечує адаптивність системи до змін у структурі цільових вебресурсів.

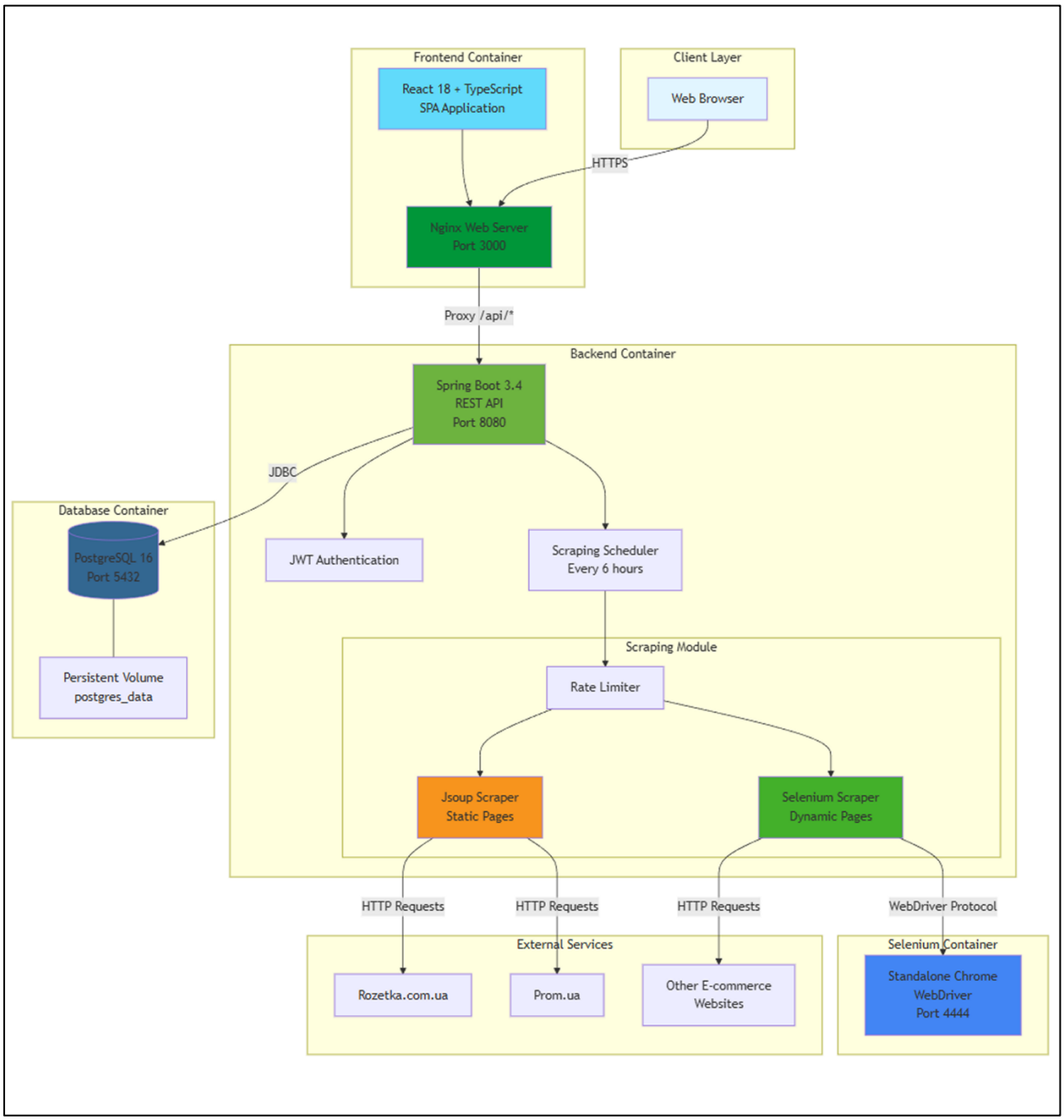


Рисунок 2.3 – Взаємодія різних рівнів системи

Збереження та обробка даних реалізовані на основі реляційної бази даних PostgreSQL, яка використовується для зберігання інформації про користувачів, товари, джерела скрапінгу, історію цін та повідомлення. Взаємодія з базою даних здійснюється через ORM-механізми Hibernate, що

спрощує роботу з даними на рівні об’єктної моделі. Для оптимізації доступу до великих масивів історичних даних передбачається індексування ключових полів і налаштування зв’язків між таблицями. Контейнеризація всіх компонентів системи, включно з базою даних та Selenium-середовищем, забезпечує ізоляцію, зручність розгортання та стабільність роботи системи, створюючи надійну основу для подальшого розвитку та масштабування.

Архітектура побудована за контейнерним підходом із застосуванням Docker та передбачає логічне розділення компонентів системи, що забезпечує масштабованість, ізоляцію та спрощення розгортання. Застосування оркестрації контейнерів сприяє автоматизації процесів моніторингу та відновлення працездатності окремих сервісів у разі виникнення збоїв. Така конфігурація дозволяє уніфікувати середовище розробки та промислової експлуатації, мінімізуючи ризики несумісності програмного оточення. На рисунку 2.4 зображено контейнерний зв’язок проєкту.

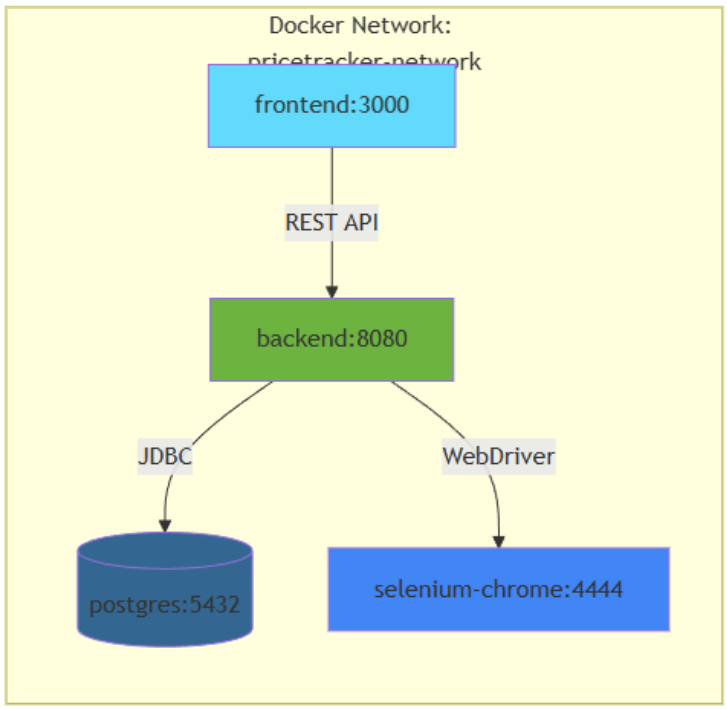


Рисунок 2.4 – Контейнерний зв’язок проєкту

У межах єдиної Docker-мережі взаємодіють фронтенд-застосунок, серверна частина, база даних та сервіс автоматизованого браузера. Фронтенд,

що працює на порту 3000, забезпечує користувацький інтерфейс і здійснює обмін даними з серверною частиною через REST API, що відповідає принципам клієнт-серверної архітектури.

Використання ізольованої віртуальної мережі дозволяє обмежити доступ до внутрішніх сервісів із зовнішнього середовища, підвищуючи загальний рівень безпеки. При цьому маршрутизація запитів між контейнерами здійснюється за внутрішніми іменами хостів, що нівелює залежність від динамічних IP-адрес. Для стабільної передачі даних між клієнтом і сервером застосовується протокол HTTP, який забезпечує стандартизовану обробку статусів відповідей. Вважається за доцільне впровадження механізмів CORS для контролю доступу до ресурсів серверної частини з боку веббраузера.

Серверна частина системи, реалізована на основі Spring та розгорнута на порту 8080, виконує роль центрального елемента обробки бізнес-логіки, управління процесами вебскрапінгу та аналізу даних. На цьому рівні реалізовано фільтрацію та трансформацію вхідних даних перед їх фіксацією в постійному сховищі.

Особлива увага приділяється обробці виняткових ситуацій, що виникають під час мережевих затримок або недоступності цільових вузлів. Для збереження результатів збору цін використовується реляційна база даних PostgreSQL, з якою взаємодія здійснюється через JDBC, що забезпечує надійний та стандартизований доступ до даних. Для оптимізації продуктивності застосовується механізм пулу з’єднань.

Процес вебскрапінгу реалізується за допомогою Selenium WebDriver, який працює у контейнері з браузером Chrome на порту 4444 та дозволяє автоматизовано взаємодіяти з вебсторінками електронної комерції. Така архітектурна організація сприяє чіткому розподілу відповідальностей між компонентами, підвищує стійкість системи та створює основу для подальшого розширення функціональних можливостей.

2.6 Проєктування модуля вебскрапінгу

Проєктування модуля вебскрапінгу в межах розроблюваної системи базується на принципах модульності та чіткого розподілу обов'язків між компонентами архітектури Spring.

Діаграми послідовностей UML – це діаграми взаємодії, які детально описують, як виконуються операції. Вони фіксують взаємодію між об'єктами в контексті співпраці. Діаграми послідовностей орієнтовані на час і візуально показують порядок взаємодії за допомогою вертикальної осі діаграми для представлення часу, часу надсилання повідомлень і часу їх надсилання [25].

На рисунку 2.5 зображено діаграму послідовності потоку даних. Процес обміну даними ініціюється через фронтенд-інтерфейс, звідки запити користувача спрямовуються через проксі-сервер Nginx до прикладного програмного інтерфейсу. На початковому етапі API здійснює звернення до бази даних для ідентифікації параметрів товарів, що підлягають моніторингу. Такий підхід забезпечує підготовку необхідного контексту для подальшої активації процедури збору даних, дозволяючи системі оперувати актуальними ідентифікаторами об'єктів у межах сесії.

Ключовим елементом модуля є підсистема скрапера, яка реалізує гнучку логіку обробки зовнішніх e-commerce ресурсів залежно від технології їхньої побудови. Діаграма послідовності відображає використання блоку альтернативного вибору для опрацювання двох типів джерел: статичних сторінок, що потребують прямого HTTP-запиту та парсингу за допомогою бібліотеки Jsoup, та динамічних сторінок. Це дозволяє ефективно вилучати дані з DOM-структури навіть у випадках складного клієнтського рендерингу.

Після завершення фази екстракції отримані результати повертаються до сервісного шару API для проведення процедур валідації та персистентності.

Процес обробки отриманих даних у базі даних структуровано за наступним алгоритмом:

– актуалізація поточної вартості товару в системі;

– створення нового запису в таблиці історії цін для формування часових рядів;
– автоматична перевірка встановлених користувачем правил та лімітів;
– генерація системного сповіщення у разі фіксації цінових коливань, що відповідають заданим тригерам.

Систематизація цих операцій дозволяє підтримувати високу цілісність даних та забезпечує основу для подальшого аналітичного опрацювання динаміки ринку.

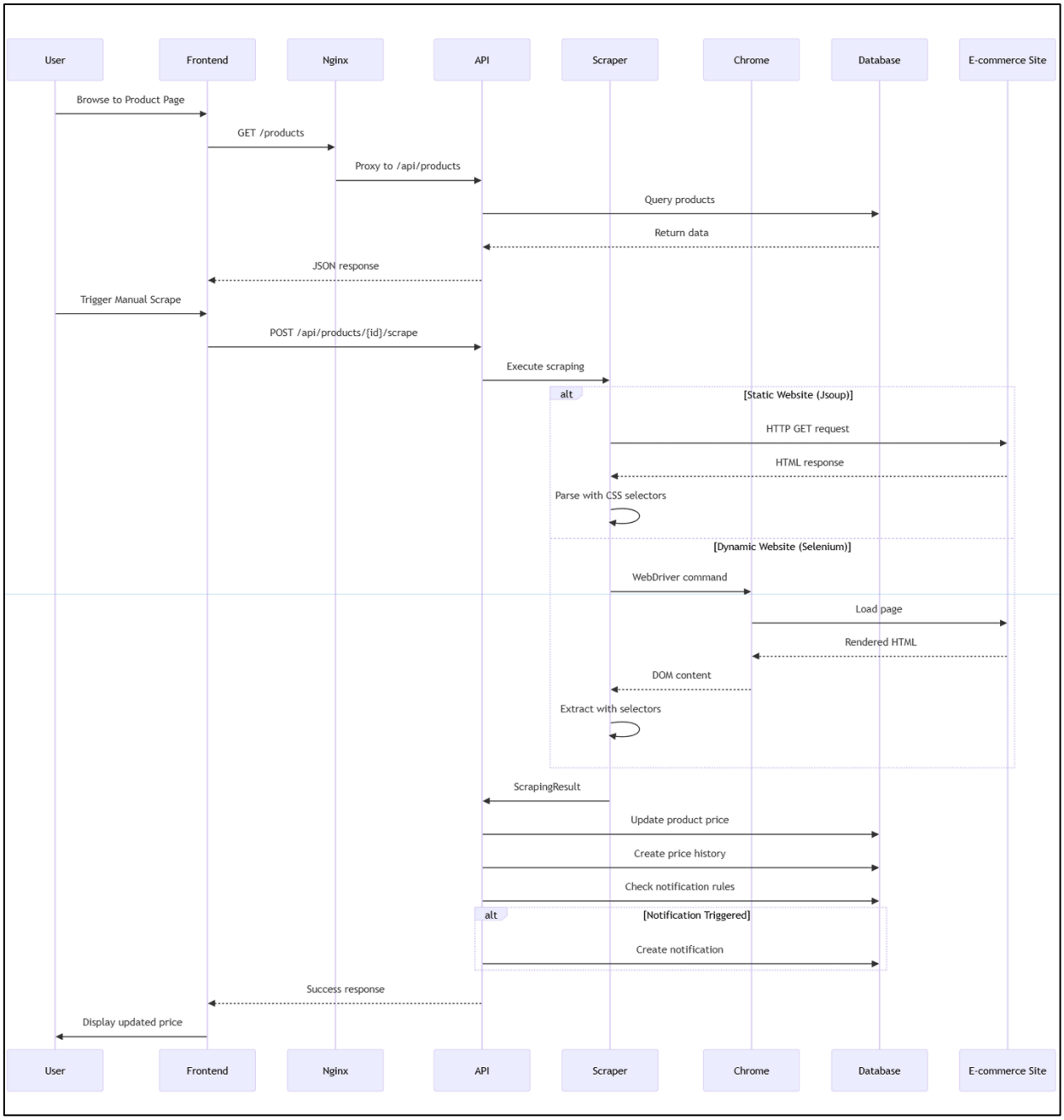


Рисунок 2.5 – Діаграма послідовності потоку даних

Завершальний етап життєвого циклу операції полягає у формуванні Success-відповіді, яка передається назад до фронтенд-частини застосунку. На основі отриманого підтвердження інтерфейс виконує асинхронне оновлення компонентів, демонструючи користувачеві актуальні цінові показники та оновлені графіки без необхідності повного перезавантаження сторінки.

Блок-схема – це діаграма, що зображує етапи процесу, робочого потоку, комп'ютерної програми чи системи. Блок-схеми складаються з фігур, з'єднаних лініями, і є покроковими процесами, що допомагають у прийнятті рішень, що зменшують невизначеність і підвищують ефективність робочих процесів [26]. На рисунку 2.6 зображено блок-схему алгоритму роботи модуля вебскрапінгу.

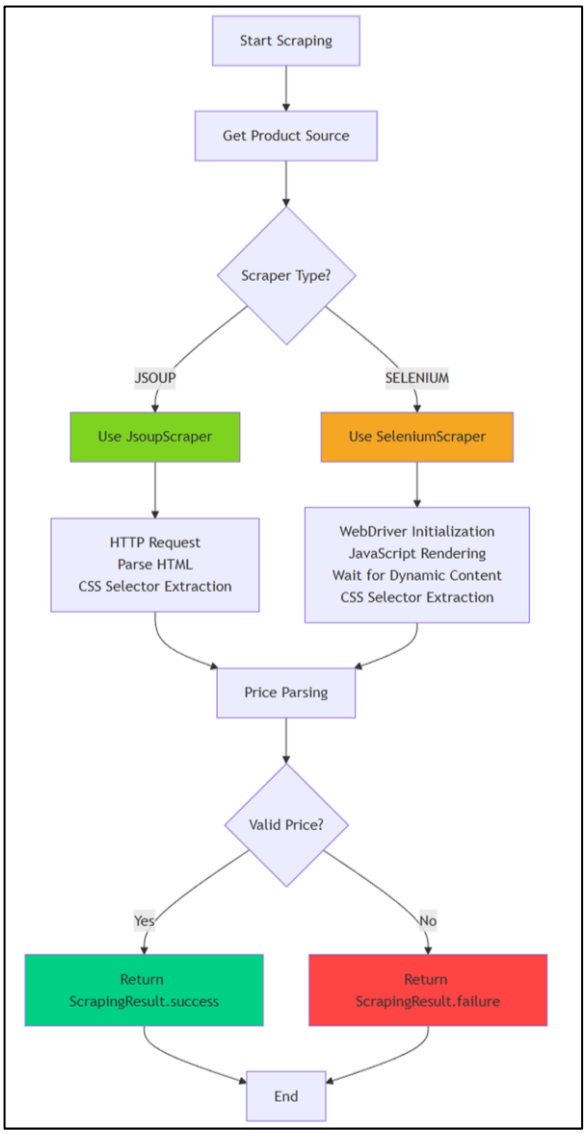


Рисунок 2.6 – Блок-схема алгоритму роботи вебскрапінгу

Проєктування алгоритмічної складової модуля вебскрапінгу базується на реалізації гнучкої логіки вибору інструментарію залежно від технічних характеристик цільового вебресурсу. На початковому етапі функціонування системи здійснюється ініціалізація процесу та отримання вихідних даних про товар, що дозволяє визначити оптимальний тип скрапера для конкретної сесії. Вибір між статичним та динамічним методами обробки сторінок є критичним етапом, оскільки він визначає подальшу стратегію взаємодії з об’єктом моніторингу та ефективність використання системних ресурсів.

Технологічний цикл збору даних структуровано за допомогою розгалуженого алгоритму, який забезпечує коректну обробку як спрощених HTML-структур, так і складних інтерфейсів з клієнтським рендерингом.

У межах цього процесу виконуються наступні операції:

– активація JsoupScraper для роботи зі статичними джерелами шляхом виконання прямих HTTP-запитів та парсингу DOM-структури;
– залучення SeleniumScraper для динамічних ресурсів, що передбачає ініціалізацію WebDriver, рендеринг JavaScript-сценаріїв та очікування завантаження контенту;
– екстракція цільових значень за допомогою уніфікованих CSS-селекторів;
– синтаксичний аналіз отриманих рядкових даних (Price Parsing) для приведення їх до числового формату.

Заключна фаза проєктування модуля зосереджена на механізмах валідації та контролю якості отриманої інформації. Після завершення етапу парсингу система виконує перевірку валідності ціни, що дозволяє відсіювати некоректні дані, спричинені змінами в структурі сайту-джерела або помилками з’єднання. Результат роботи модуля інкапсулюється в об’єкт ScrapingResult, який повертає статус успіху або помилки, забезпечуючи цілісність аналітичного контуру системи. Такий підхід гарантує високу точність відстеження динаміки цін та надійність формування масивів даних для подальшого аналізу.

2.7 Висновки до розділу 2

У другому розділі проведено комплексне проєктування системи автоматизованого збору та аналізу цін, що дозволило сформувати технічний фундамент для її розробки. Аналіз предметної області підтвердив актуальність системи для маркетингового моніторингу та SWOT-аналізу конкурентів, що забезпечує бізнесу стратегічні переваги, а споживачам – оптимізацію витрат. Формалізація вимог дозволила визначити пріоритетні функції: керування джерелами через CSS-селектори, автоматизований скрапінг, ведення історії цін та візуалізацію аналітики. При цьому нефункціональні вимоги зосереджені на продуктивності, безпеці даних та масштабованості архітектури.

На основі UML-моделювання та ER-діаграм розроблено логічну структуру бази даних, яка забезпечує цілісне зберігання інформації про товари, історію цін та налаштування користувачів. Обґрунтовано вибір технологічного стеку: Java та Spring Boot для бекенду, React та TypeScript для фронтенду, а також PostgreSQL як СУБД. Універсальність збору даних досягається поєднанням бібліотеки Jsoup для статичних сторінок та Selenium WebDriver для динамічного контенту, що дозволяє обробляти ресурси будь-якої складності.

Архітектура системи побудована за трирівневим принципом із використанням Docker-контейнеризації, що гарантує стабільність та зручність розгортання. Проєктування модуля вебскрапінгу та алгоритмів парсингу забезпечує гнучку обробку даних, їх валідацію та автоматичне сповіщення користувачів про цінові коливання. Запропоновані проєктні рішення є цілісними, логічно обґрунтованими та створюють надійну базу для програмної реалізації відмовостійкої системи моніторингу ринку.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ВЕБСКРАПІНГУ

3.1 Реалізація backend-частини

3.1.1 Базові налаштування

Розробка backend-компоненти системи здійснювалася з використанням Java 21 та Spring Boot 3.4, що забезпечує сучасний та надійний фундамент для побудови масштабованих вебдодатків. Вибір Java обґрунтовується її популярністю в корпоративному секторі, розвинутою екосистемою фреймворків та гарантіями кросплатформенної сумісності. Spring Boot 3.4 надає потужну абстракцію над Spring фреймворком, значно прискорюючи розробку через автоматичну конфігурацію та вбудовані стартери для типових завдань.

Управління залежностями реалізується через Apache Maven 3.4, що дозволяє декларативно визначити всі необхідні бібліотеки в єдиному місці. До ключових залежностей проєкту відносяться Spring Web для RESTful сервісів, Spring Data JPA для об’єктно-реляційного відображення, Spring Security для автентифікації та авторизації, PostgreSQL драйвер для роботи з БД, Jsoup та Selenium для вебскрейпінгу, JJWT для обробки JWT токенів та SpringDoc OpenAPI для автогенерування інтерактивної документації API.

Система використовує PostgreSQL 16 як основне сховище даних, обране за його надійністю, підтримкою складних запитів та оптимізацією для робочих навантажень. Підключення до БД конфігурується через файл application.yml, який зображено на рисунку 3.1.

Параметр hibernate.ddl-auto: update активує автоматичне створення та синхронізацію схеми БД на основі JPA сутностей, спрощуючи процес розгортання. Архітектура підключення передбачає використання connection pooling для оптимізації використання ресурсів та прискорення операцій з БД.

Файл application.yml містить також конфігурацію Jackson для серіалізації JSON та налаштування Hibernate для форматування SQL запитів в логах.

```yaml
spring:
  application:
    name: price-tracker

  datasource:
    url: jdbc:postgresql://${DB_HOST:localhost}:${DB_PORT:5432}/${DB_NAME:pricetracker}
    username: ${DB_USERNAME:postgres}
    password: ${DB_PASSWORD:postgres}
    driver-class-name: org.postgresql.Driver

  jpa:
    hibernate:
      ddl-auto: update
    show-sql: false
    properties:
      hibernate:
        dialect: org.hibernate.dialect.PostgreSQLDialect
        format_sql: true

  jackson:
    serialization:
      write-dates-as-timestamps: false
    default-property-inclusion: non_null

server:
  port: 8080

# JWT Configuration
jwt:
  secret: ${JWT_SECRET:your-256-bit-secret-key-here-must-be-at-least-32-characters-long}
  expiration: 86400000  # 24 hours in milliseconds

# Scraping Configuration
```

Рисунок 3.1 – Файл application.yml

Безпека системи реалізується через комплексний підхід на основі JWT (JSON Web Tokens) та Spring Security.

Конфігураційний клас SecurityConfig.java з **Додаток А** встановлює наступні правила:

– Stateless архітектура: використання SessionCreationPolicy.STATELESS означає, що кожен запит перевіряється незалежно, без збереження стану сесії на сервері, що критично важливо для масштабованості розподіленої системи;

– JWT фільтр: кожний запит перехоплюється фільтром JwtAuthenticationFilter, який валідує токен та встановлює контекст безпеки;

- CORS налаштування: явна конфігурація дозволяє запити від фронтенду на localhost:3000 та localhost:5173 з явним переліком дозволених методів та заголовків;
- захист CSRF: вимкнено для stateless API, оскільки CSRF атаки невекторні без сесій.

Клас JwtTokenProvider.java відповідає за весь цикл роботи з токенами. При реєстрації користувача генерується токен, який підписується HMAC-SHA256 алгоритмом з 256-бітним ключем з змінної середовища JWT_SECRET. Токен містить email користувача як subject та час закінчення дії (24 години). При кожному запиті токен валідується шляхом перевірки підпису та часу експірації. Реалізація такого механізму дозволяє забезпечити безстанну (stateless) автентифікацію, що є критично важливим для горизонтального масштабування серверної частини. Сервіс CustomUserDetailsService.java інтегрується зі Spring Security для завантаження деталей користувача з БД. Пароль користувача хешується за допомогою BCrypt з адаптивним числом раундів, що забезпечує захист від атак перебору. Застосування солі у процесі хешування гарантує унікальність результату навіть для однакових паролів, що нівелює ефективність використання райдужних таблиць.

Для забезпечення періодичного вебскрейпінгу використовується Spring Scheduling механізм. Клас SchedulerConfig.java конфігурує пул потоків розміром 5 для паралельного виконання завдань. Цей розмір обиралося як баланс між паралельністю та упередженням перевантаження системи. Кожний потік має дескриптивну назву для спрощення налагодження, а обробник помилок гарантує, що збої в одному завданні не вплинуть на виконання інших.

Використання виокремленого пулу потоків запобігає блокуванню основного потоку виконання програми тривалими операціями вводу-виводу. Клас ScrapingScheduler.java містить метод з анотацією @Scheduled, що виконується за крон-виразом, встановленим у конфігурації (0 0 */6 * * * означає виконання кожні 6 годин). При запуску витягуються всі активні продукти та виконується їхній скрейпінг в межах запланованого часу.

Автоматизація цього процесу дозволяє підтримувати актуальність аналітичних даних без безпосереднього втручання адміністратора системи.

3.1.2 Реалізація REST API

REST API система організована відповідно до REST принципів з використанням ресурс-орієнтованого підходу. Кожна основна сутність (користувач, продукт, джерело, сповіщення) має власний контролер, який обробляє HTTP запити. икористання стандартизованих методів передачі даних дозволяє забезпечити високий рівень сумісності системи з різноманітними типами клієнтських застосунків. Архітектура побудована за трирівневою моделлю: контролер, сервіс, репозиторій, що забезпечує розділення обов'язків.

Така декомпозиція сприяє підвищенню тестованості коду, оскільки бізнес-логіка відокремлена від механізмів обробки мережевих запитів та прямої взаємодії з базою даних.

Контролер AuthController.java надає три основні кінцеві точки:

– POST /api/auth/register – реєстрація нового користувача;
– POST /api/auth/login – автентифікація та отримання JWT токена;
– GET /api/auth/me – отримання даних поточного користувача.

Для захисту кінцевих точок, що потребують авторизації. На рисунку 3.2 зображено сервіс AuthService.java обробляє логіку аутентифікації, включаючи валідацію даних, хешування пароля за допомогою BCrypt та створення дефолтних параметрів сповіщень для нових користувачів.

Контролер ProductController.java надає повний набір CRUD операцій для управління продуктами:

– GET /api/products – список продуктів з фільтрацією, пагінацією та сортуванням;
– GET /api/products/{id} – отримання деталей продукту;

– POST /api/products – створення нового продукту;
– PUT /api/products/{id} – оновлення продукту;
– DELETE /api/products/{id} – видалення продукту;
– POST /api/products/{id}/scrape – ручне запуск скрейпінгу.

```java
@Transactional
public AuthResponse register(RegisterRequest request) {
    if (userRepository.existsByEmail(request.getEmail())) {
        throw new BadRequestException("Email is already registered");
    }

    // Get or create USER role
    Role userRole = roleRepository.findByName(Role.ROLE_USER)
            .orElseGet(() -> roleRepository.save(Role.builder().name(Role.ROLE_USER).build()));

    // Create user
    User user = User.builder()
            .email(request.getEmail())
            .password(passwordEncoder.encode(request.getPassword()))
            .firstName(request.getFirstName())
            .lastName(request.getLastName())
            .roles(Set.of(userRole))
            .build();

    user = userRepository.save(user);

    // Create default notification settings
    UserNotificationSettings settings = UserNotificationSettings.builder()
            .user(user)
            .build();
    notificationSettingsRepository.save(settings);

    log.info("User registered successfully: {}", user.getEmail());

    // Generate token
    String token = jwtTokenProvider.generateToken(user.getEmail());

    return AuthResponse.of(
            token,
            jwtTokenProvider.getJwtExpiration(),
            UserDto.fromEntity(user)
    );
}
```

Рисунок 3.2 – AuthService.java файл

На рисунку 3.3 зображено сервіс ProductService.java, реалізує складну логіку фільтрації за допомогою custom JPQL запитів.

Обробка помилок у REST API централізована через клас GlobalExceptionHandler, який перехоплює винятки та повертає користувачеві інформативні відповіді з відповідними HTTP-статус-кодами. Система

розрізняє різні типи помилок: BadRequestException (400), ResourceNotFoundException (404) та загальні помилки сервера (500).

```
@Transactional(readOnly = true)
public PageResponse<ProductDto> getUserProducts(String userEmail, ProductFilterRequest filter) {
    User user = getUserByEmail(userEmail);

    Sort sort = Sort.by(
            filter.getSortDirection().equalsIgnoreCase(anotherString: "ASC") ? Sort.Direction.ASC : Sort.Direction.DESC,
            filter.getSortBy()
    );
    Pageable pageable = PageRequest.of(filter.getPage(), filter.getSize(), sort);

    Page<Product> productPage;

    if (filter.getQuery() != null && !filter.getQuery().isBlank()) {
        productPage = productRepository.searchByUserIdAndQuery(user.getId(), filter.getQuery(), pageable);
    } else {
        productPage = productRepository.findByFilters(
                user.getId(),
                filter.getSourceId(),
                filter.getMinPrice(),
                filter.getMaxPrice(),
                filter.getIsActive(),
                pageable
        );
    }

    List<ProductDto> content = productPage.getContent().stream()
            .map(ProductDto::fromEntity)
            .collect(Collectors.toList());

    return PageResponse.of(productPage, content);
}

@Transactional(readOnly = true)
public ProductDto getProductById(Long id, String userEmail) {
    User user = getUserByEmail(userEmail);
    Product product = productRepository.findByIdAndUserId(id, user.getId())
            .orElseThrow(() -> new ResourceNotFoundException(resourceName: "Product", fieldName: "id", id));
    return ProductDto.fromEntity(product);
}
```

Рисунок 3.3 – JPQL запити фільтрації

Кожна відповідь про помилку містить стандартизований набір полів, включаючи часову мітку, повідомлення про причину виникнення проблеми та шлях запиту. Це забезпечує уніфікацію форматів обміну даними між сервером та клієнтом, що спрощує процес налагодження фронтенд-частини.

3.1.3 Робота з базою даних

База даних проєкту розбудована на PostgreSQL 16 та містить 7 основних таблиць, які пов’язані у складну схему відносин. Управління базою даних

здійснюється через Spring Data JPA, що забезпечує объектно-реляційне відображення ORM та автоматичне генерування SQL-запитів.

Для оптимізації запитів до бази даних застосовуються індекси на часто використовуваних полях: user_id, product_id, source_id та created_at. Це значно скорочує час виконання запитів при роботі з великими таблицями. Spring Data JPA автоматично генерує DDL-скрипти для створення таблиць на основі анотацій у класах сутностей.

Цей процес контролюється параметром spring.jpa.hibernate.ddl-auto у конфігурації, який може приймати значення create, create-drop, update або validate залежно від стадії розробки. Транзакційність операцій забезпечується через анотацію @Transactional у методах сервісу.

Це гарантує, що набір операцій над базою даних виконується як атомарна одиниця: якщо одна операція не вдається, всі попередні зміни скасовуються. Для впровадження функцій вебскрапінгу система використовує асинхронні операції з базою даних. Операції запису великих об’ємів даних (багаторазові вставлення записів price_history після скрапінгу) виконуються у окремих потоках, що не блокує основний потік обробки API-запитів. Цей підхід забезпечує високу доступність системи навіть при інтенсивному скрапінгу.

3.2 Реалізація frontend-частини

Frontend реалізовано на React 18 з TypeScript та збирається Vite. Структура проєкту організована за принципом розділення відповідальностей: шар представлення pages та components, шар контексту аутентифікації context/AuthContext.tsx, доступ до API src/api/*.ts та спільні типи src/types/index.ts. Готовий білд постачається через Nginx у контейнері, що визначено у Dockerfile та nginx.conf, а інтеграція з іншими сервісами виконується через docker-compose.yml, де фронтенд публікується на порту

3000 і залежить від backend-сервісу. На рисунку 3.4 зображено nginx.conf файл.

```
server {
    listen 80;
    server_name localhost;
    root /usr/share/nginx/html;
    index index.html;

    # Gzip compression
    gzip on;
    gzip_types text/plain text/css application/json application/javas

    # Handle React Router (SPA routing)
    location / {
        try_files $uri $uri/ /index.html;
    }

    # Proxy API requests to backend
    location /api {
        proxy_pass http://backend:8080;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Cache static assets
    location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg|woff|woff2)$ {
        expires 1y;
        add_header Cache-Control "public, immutable";
    }

    # Security headers
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header X-XSS-Protection "1; mode=block" always;
}
```

Рисунок 3.4 – Конфігурація nginx

Базове компонування реалізовано в DashboardLayout.tsx, де визначено каркас із хедером, боковою навігацією та областю контенту. Цей макет повторно використовується більшістю внутрішніх сторінок (DashboardPage.tsx, ProductsPage.tsx, NotificationsPage.tsx, SourcesPage.tsx, SettingsPage.tsx, ProductDetailPage.tsx), що мінімізує дублювання та забезпечує узгоджений UX. На рисунку 3.5 продемонстровано сторінку DashBoard.

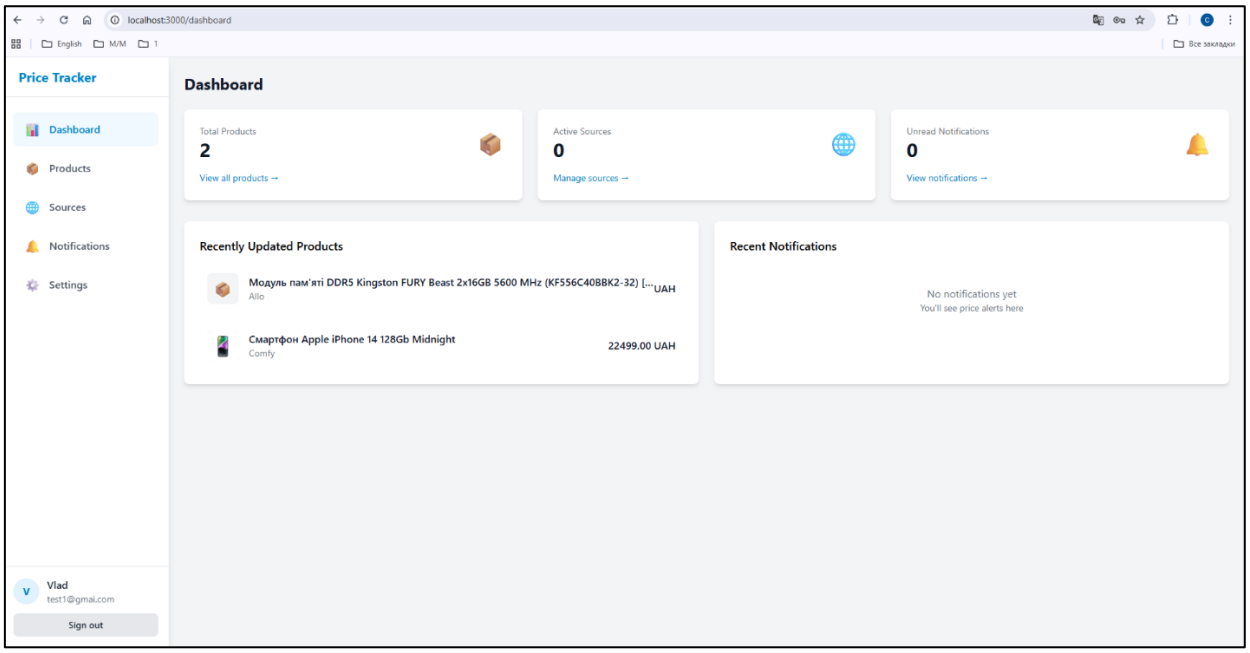
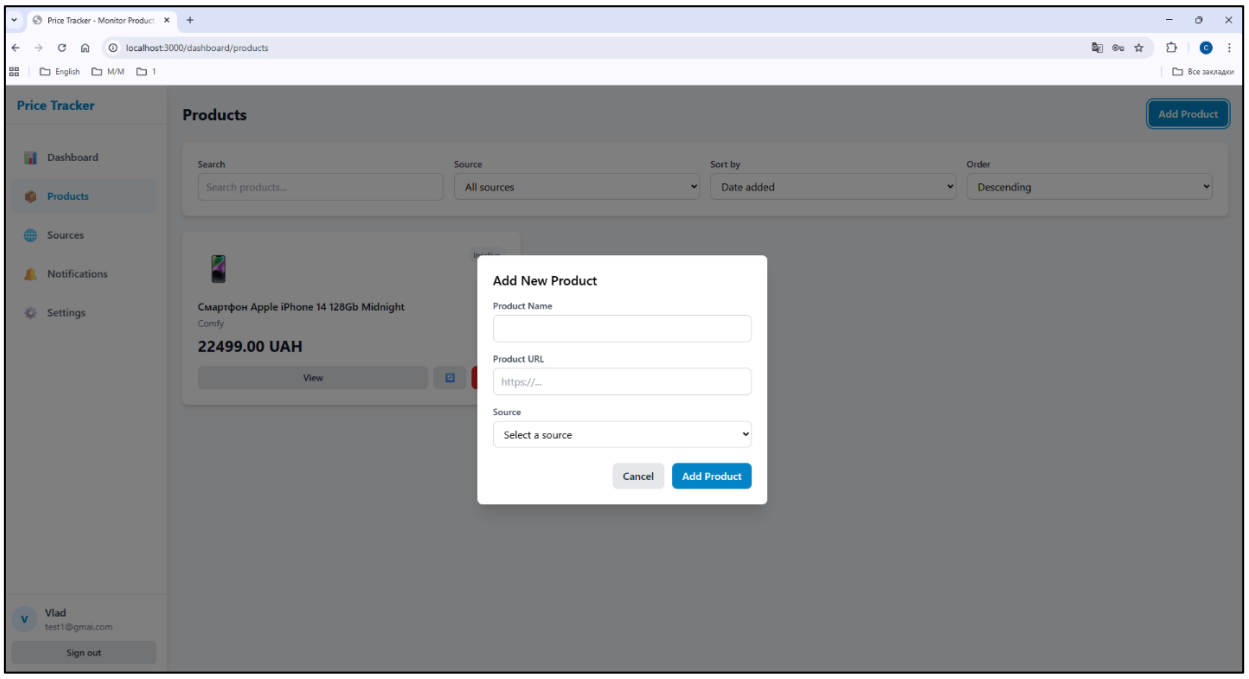


Рисунок 3.5 – Сторінка DashBoard

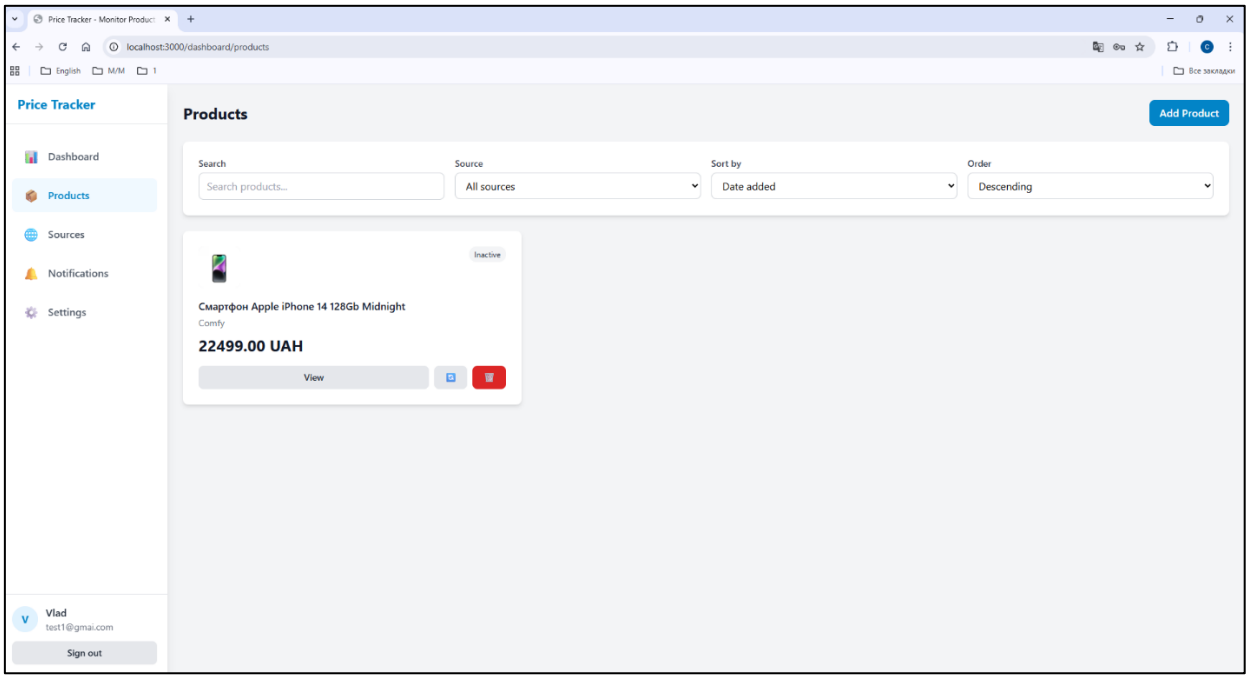
Сторінки входу й реєстрації, використовуючи контекст аутентифікації для виклику відповідних API-методів. Дані форм валідовано на рівні клієнта перед відправленням запиту, а стани завантаження й помилок відображаються користувачу через мінімалістичні компоненти інтерфейсу.

Файл DashboardPage.tsx з **Додаток Б** агрегує ключові показники та графіки, використовуючи переважно готові візуальні компоненти і легковагові віджети. Тут здійснюється початкове завантаження даних після авторизації та відображення основних метрик (кількість відстежуваних товарів, активних джерел, сповіщень).

Файл ProductsPage.tsx поєднує таблицю з переліком товарів, пагінацію та фільтри. Запити отримання списку, створення, оновлення й видалення делеговані до api/products.ts. Файл SourcesPage.tsx реалізує CRUD-операції над джерелами, спираючись на api/sources.ts. Інтерфейс дає змогу вмикати/вимикати джерела, змінювати CSS-селектори та тип скрапера (Jsoup/Selenium), що синхронізовано з бекенд-ентіті Source. Особлива увага приділяється забезпеченню реактивності інтерфейсу, що досягається шляхом використання хуків для синхронізації локального стану компонентів із даними. На рисунку 3.6 продемонстровано сторінку Products.



а) Додавання нового товару для скрапінгу



б) Відображення інформації про товар

Рисунок 3.6 – Сторінка Products

У docker-compose.yml фронтенд залежить від backend, публікується на 3000:80 та використовує спільну мережу pricetracker-network. Це забезпечує прямий доступ до API за внутрішнім DNS-ім’ям pricetracker-backend:8080 або через змінні середовища, визначені під час білду.

Типи у types/index.ts віддзеркалюють DTO бекенда: ProductDto, PriceHistoryDto, SourceDto, NotificationDto, PageResponse, що зменшує ризики розсинхронізації при змінах API. Під час додавання нових полів на бекенді розробник оновлює відповідні інтерфейси на фронтенді й отримує статичну перевірку коректності використання цих полів у компонентах.

Централізований інтерсептор у api/axios.ts перетворює мережеві збої та HTTP-помилки в уніфікований об’єкт помилки, який споживається у сторінках. Сторінки й компоненти відображають зрозумілі повідомлення (наприклад, при недоступності бекенда або втраті сесії). Для сценарію неавторизованого доступу здійснюється автоматичне перенаправлення на сторінку входу та очищення токену в контексті.

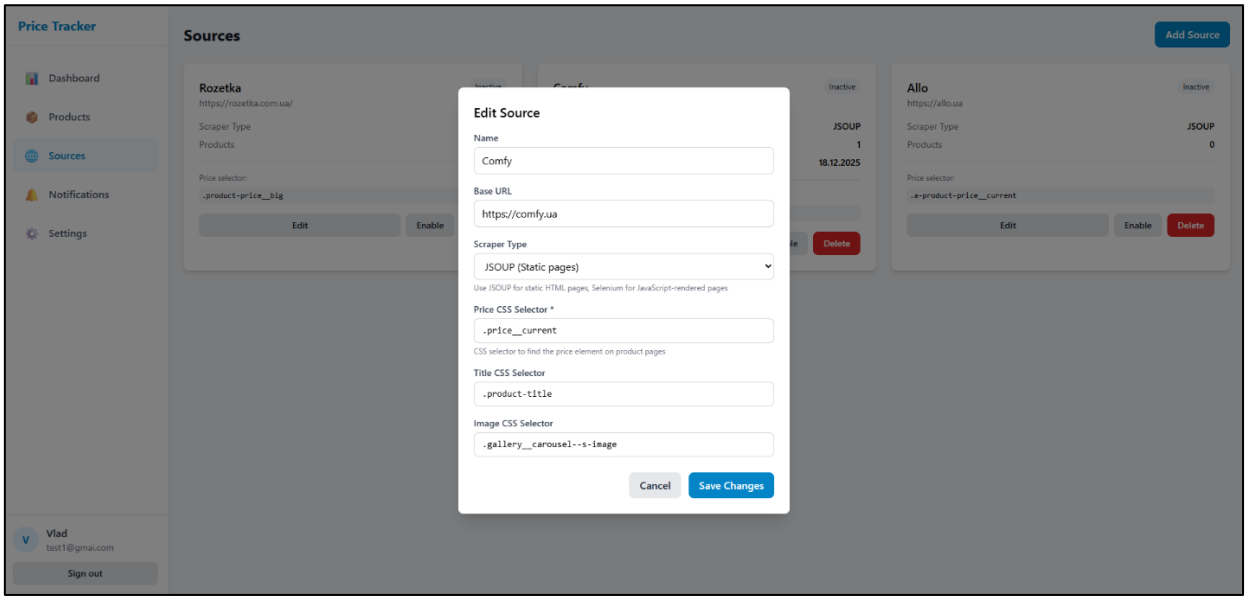
Frontend-частина побудована модульно: спільний макет, чіткі API-модулі, типізовані DTO та ізольований контекст аутентифікації. Це забезпечує узгодженість із REST API, простоту масштабування функціональності (додавання нових сторінок або візуалізацій) та передбачуваний користувацький досвід під час відстеження динаміки цін і аналітики.

3.3 Реалізація модуля вебскрапінгу

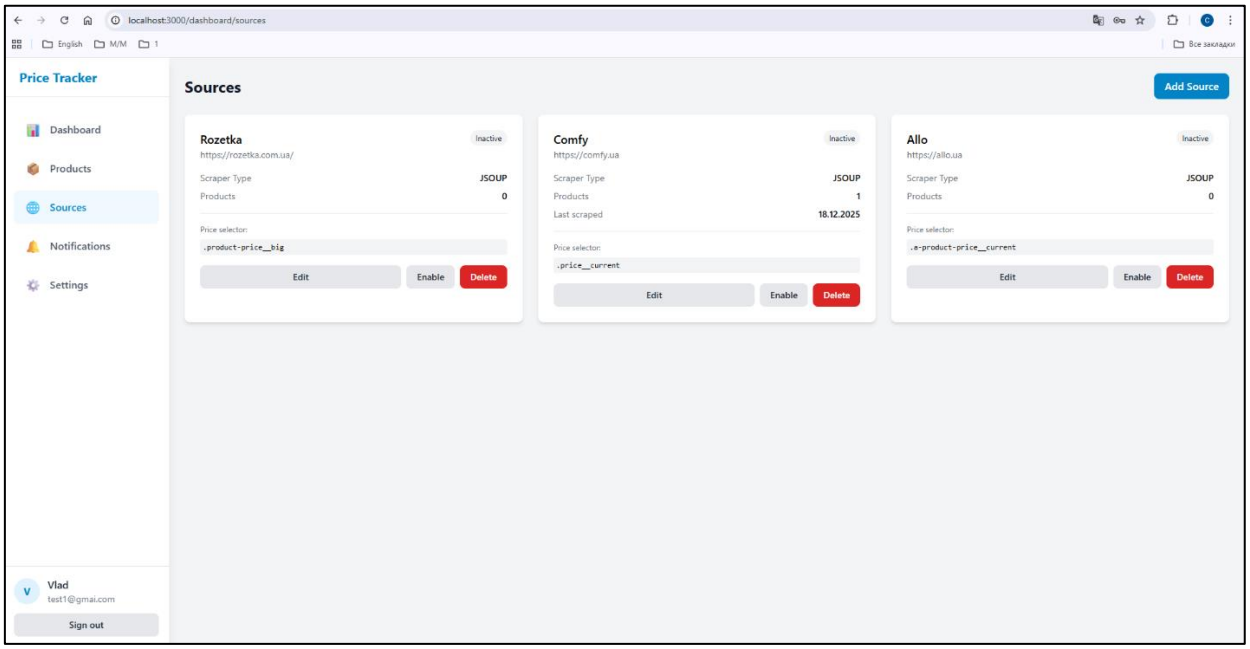
Модуль вебскрапінгу побудовано за принципом стратегії, що дозволяє динамічно обирати інструмент збору даних залежно від типу вебсайту. Інтерфейс Scraper визначає контракт для всіх реалізацій, а конкретні стратегії (JsoupScraper, SeleniumScraper) інкапсулюють логіку роботи з відповідними бібліотеками. Сервіс ScrapingService з Додаток В координує виконання операцій скрапінгу, вибір стратегії та обробку результатів.

Кожне джерело містить конфігурацію: базову URL, CSS-селектори для вилучення цін та назв продуктів, а також тип скрапера (ScraperType.JSOUP або ScraperType.SELENIUM). Така гнучкість дозволяє підтримувати як статичні сайти, так і динамічні одностарінкові застосунки (SPA) без зміни основної

логіки. На рисунку 3.7 продемонстровано сторінку Sources де користувач може додати сайт, з якого треба парсити данні.



а) Додавання нового джерела



б) Перелік джерел для вебскрапінгу

Рисунок 3.7 – Сторінка Sources

Планувальник завдань ініціалізує ThreadPoolTaskScheduler, що забезпечує паралельне виконання скрапінгу з кількох джерел. Автоматичний скрапінг запускається за розкладом кожні 6 годин, що налаштовується через

application.yml. Крім планового збору, користувачі можуть ініціювати ручний скрапінг через відповідний ендпоінт REST API.

Процес скрапінгу починається з підключення до URL з встановленим user-agent та таймаутом 10 секунд. Метод connect() повертає Document, з якого витягуються елементи за допомогою селекторів. Для отримання ціни використовується селектор priceSelector, для назви продукту – nameSelector. Якщо елемент знайдено, його текстовий вміст передається до PriceParser для нормалізації.

PriceParser обробляє різноманітні формати цін: видаляє символи валют, пробіли, коми, замінює нестандартні роздільники на крапку та перетворює результат у BigDecimal. Регулярні вирази виділяють числові значення з рядків типу «₴1 234,56», «€99.99», «$1,000.00». Така універсальність дозволяє працювати з вебсайтами різних країн без додаткової конфігурації. При успішному парсингу JsoupScraper повертає ScrapingResult з ціною та назвою продукту. У разі помилки (недоступність URL, відсутність елемента, невалідний формат ціни) результат містить статус помилки та опис проблеми. Це дозволяє системі сповіщень інформувати користувача про збої в роботі джерела.

Selenium WebDriver застосовується для сайтів з динамічним контентом, який завантажується JavaScript після початкового рендерингу сторінки. SeleniumScraper з **Додаток Г** керує headless-браузером Chrome через Selenium Grid, що розгорнуто в окремому контейнері selenium-chrome у docker-compose.yml.

Підключення до Selenium Grid здійснюється через Remote WebDriver з URL http://selenium-chrome:4444/wd/hub. Налаштування браузера включають headless-режим, вимкнення GPU-рендерингу та встановлення розміру вікна для коректного відображення адаптивних елементів. User-agent конфігурується для імітації звичайного браузера, що знижує ризик блокування з боку антискрапінгових систем.

Після завантаження сторінки SeleniumScraper очікує видимості елемента за селектором (за допомогою WebDriverWait та ExpectedConditions). Це

критично для SPA, де контент може з’являтися з затримкою. Таймаут очікування налаштовується через конфігураційний параметр (типово 15 секунд). Витягування даних виконується через findElement() з CSS-селекторами або XPath. Отриманий текст проходить ті самі етапи нормалізації через PriceParser, що і для Jsoup. Після завершення роботи WebDriver обов’язково закривається (driver.quit()) для звільнення ресурсів браузера в Selenium Grid.

3.4 Висновки до розділу 3

Модуль вебскрапінгу реалізовано за патерном стратегії: інтерфейс Scraper має дві основні реалізації – JsoupScraper для статичних сторінок і SeleniumScraper для динамічних SPA, що керується через Selenium Grid. Вибір стратегії виконується у сервісі скрапінгу на основі конфігурації джерела (тип скрапера, CSS-селектори), що зберігається в сутності Source. Нормалізація цін уніфікована через PriceParser, який очищує валютні символи й перетворює рядкові значення в числовий формат.

Для динамічних сайтів Selenium працює в headless-режимі з тайм-аутами очікування елементів і примусовим закриттям WebDriver після завершення. Планові запуски скрапінгу організовані через SchedulerConfig та пул потоків, а ScrapingRateLimiter обмежує частоту запитів до доменів, запобігаючи блокуванням. Результати фіксуються у таблиці price_history: при успіху зберігається ціна й мітка часу, при збоях – статус FAILED та повідомлення про помилку. Якщо зміна ціни перевищує заданий поріг користувача, генерується сповіщення; транзакційність (@Transactional) гарантує атомарність запису історії та створення сповіщень. Централізована обробка винятків і логування помилок спрощують діагностику та дозволяють за потреби тимчасово деактивувати проблемні джерела.

ВИСНОВКИ

У кваліфікаційній роботі було розглянуто основні етапи проєктування та реалізації системи вебскрапінгу для моніторингу цін “PriceMonitor”, розробленої засобами Spring Boot для серверної частини та React з TypeScript для клієнтського інтерфейсу.

Під час роботи було реалізовано такі задачі:

– проаналізовано існуючі рішення та технології вебскрапінгу та моніторингу цін;
– сформульовано функціональні та нефункціональні вимоги до системи;
– спроєктувати трирівневу архітектуру системи з чітким розмежуванням функціоналу;
– реалізовано компоненти серверної частини на Spring Boot;
– розроблено клієнтський інтерфейс з використанням React та TypeScript;
– створино систему вебскрапінгу на основі Jsoup та Selenium;
– протестовано компоненти та проведено комплексне тестування системи.

Під час виконання кваліфікаційної роботи реалізовано три розділи. У першому розділі наведено обґрунтування актуальності розробки системи в умовах висококонкурентного ринку електронної комерції. Шляхом дослідження потреб споживачів та аналізу існуючих аналогів визначено необхідність створення адаптивного рішення, що забезпечує точність даних та зручність через сучасні інтерфейси.

У другому розділі обґрунтовано вибір технологічного стеку (Spring Boot, React, PostgreSQL, Jsoup, Selenium), виконано проєктування архітектури системи: розроблено діаграми прецедентів для різних типів користувачів, спроєктовано ER-модель бази даних, розроблено діаграма послідовності

потоку даних, блок-схема алгоритму роботи вебскрапінгу, визначено трирівневу архітектуру та спроєктовано інтерфейси взаємодії між ними.

У третьому розділі наведено приклади реалізації ключових компонентів: реалізації backend-частини та frontend-частни, а також реалізації модуля вебскрапінгу.

Результати виконання роботи демонструють ефективність обраного технологічного підходу для створення сучасної системи моніторингу цін. Розроблена система здатна напівавтоматично збирати цінові дані з різних вебджерел, обробляти та аналізувати інформацію, надавати інтуїтивний інтерфейс для користувачів та забезпечувати миттєві сповіщення про зміни цін. Система демонструє високу продуктивність, масштабованість та безпеку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Mitchell R. Web Scraping with Python : 2nd edition. O'Reilly, 2018. 306 p.

2. Loyola A. Web Scraping: What It Is, How It Works, and Best Practices. *Browserless*. 2025. URL : https://www.browserless.io/blog/web-scraping-guide#:~:text=What%20Is%20Web%20Scraping?,skipping%20the%20repetitive%20grunt%20work (дата звернення: 27.03.2025).

3. Sahin K. Java Web Scraping Handbook : Learn advanced Web Scraping techniques. Lean Publishing, 2018. 115 p.

4. What Is Web Scraping, And Why Is It So Popular Nowadays? *EIMT*. URL : https://www.eimt.edu.eu/what-is-web-scraping-and-why-is-it-so-popular-nowadays (дата звернення: 29.03.2025).

5. Chen F. Research on Real-time E-commerce Price Comparison System Using Python Web Scraping Technology. *International Journal of Computer Science and Information Technology*. 2024. Vol. 4, No. 2. P. 127-136. URL : https://doi.org/10.62051/ijcsit.v4n2.18 (дата звернення: 29.03.2025).

6. Onyenwe I., Onyedinma E., Nwafor C., Agbata O. Developing Products Update-Alert System for e-Commerce Websites Users Using HTML Data and Web Scraping Technique. *International Journal on Natural Language Computing*. 2021. URL : https://doi.org/10.48550/arXiv.2109.00656 (дата звернення: 02.04.2025).

7. Ghose A., Sundararajan A. Evaluating Pricing Strategy Using e-Commerce Data: Evidence and Estimation Challenges. *Statistical Science*. 2006. Vol. 21, No. 2. P. 131-142. URL : https://doi.org/10.1214/088342306000000187 (дата звернення: 02.04.2025).

8. Haughey D. SWOT Analysis. *ProjectSmart*. 2021. URL : https://www.projectsmart.co.uk/tools/swot-analysis.php (дата звернення: 02.04.2025).

9. Functional and Non Functional Requirements. *GeeksforGeeks*. 2025. URL : https://www.geeksforgeeks.org/software-engineering/functional-vs-non-functional-requirements/ (дата звернення: 15.04.2025).

10. AltexSoft Editorial Team. Nonfunctional Requirements in Software Engineering: Examples, Types, Best Practices. *AltexSoft*. URL : https://www.altexsoft.com/blog/non-functional-requirements (дата звернення: 26.04.2025).

11. Baumann H., Grassle P., Baumann P. UML 2.0 in Action: A project-based tutorial. Packt Publishing, 2005. 248 p.

12. Taylor A. G. SQL All-in-One For Dummies : 4th ed. For Dummies, 2024. 800 p.

13. Java. URL : https://www.java.com/en (дата звернення: 15.05.2025).

14. Spring Boot Documentation. URL : https://docs.spring.io/spring-boot/index.html (дата звернення: 15.05.2025).

15. Apache Maven Project. URL : https://maven.apache.org (дата звернення: 15.05.2025).

16. Selenium Documentation. URL : https://www.selenium.dev/documentation (дата звернення: 15.05.2025).

17. Jsoup : Java HTML Parser. URL : https://jsoup.org (дата звернення: 15.05.2025).

18. Hibernate ORM. URL : https://hibernate.org/orm (дата звернення: 15.05.2025).

19. What is PostgreSQL? URL : https://www.postgresql.org/about (дата звернення: 15.05.2025).

20. TypeScript is JavaScript with syntax for types. URL : https://www.typescriptlang.org (дата звернення: 15.05.2025).

21. React Reference Overview. URL : https://react.dev/reference/react (дата звернення: 15.05.2025).

22. What is Docker? URL : https://docs.docker.com/get-started/docker-overview (дата звернення: 15.05.2025).

23. Choosing the best architecture for your software. URL: https://dev.to/yokwejuste/choosing-the-best-architecture-for-your-software-23h9 (дата звернення: 03.06.2025).

24. What is three-tier architecture? URL : https://www.ibm.com/think/topics/three-tier-architecture (дата звернення: 15.06.2025).

25. What is Sequence Diagram? URL : https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram (дата звернення: 02.07.2025).

26. What is a flowchart? URL : https://www.ibm.com/think/topics/flowcharthttps://www.ibm.com/think/topics/flowchart (дата звернення: 02.07.2025).

ДОДАТОК А

Конфігураційний клас SecurityConfig

```
public class SecurityConfig {
   private final JwtAuthenticationFilter jwtAuthenticationFilter;
   private final UserDetailsService userDetailsService;
   @Bean
   public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
      http
            .csrf(AbstractHttpConfigurer::disable)
            .cors(cors -> cors.configurationSource(corsConfigurationSource()))
            .sessionManagement(session -> session
                  .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(auth -> auth
                  // Public endpoints
                  .requestMatchers("/api/auth/**").permitAll()
                  .requestMatchers("/swagger-ui/**", "/swagger-ui.html", "/api-docs/**", "/v3/api-
docs/**").permitAll()
                  .requestMatchers(HttpMethod.OPTIONS, "/**").permitAll()
                  // All other endpoints require authentication
                  .anyRequest().authenticated()
            )
            .authenticationProvider(authenticationProvider())
            .addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class);
      return http.build();
   }
   @Bean
   public CorsConfigurationSource corsConfigurationSource() {
      CorsConfiguration configuration = new CorsConfiguration();
      configuration.setAllowedOrigins(List.of("http://localhost:3000", "http://localhost:5173"));
      configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "PATCH",
"DELETE", "OPTIONS"));
      configuration.setAllowedHeaders(Arrays.asList("Authorization", "Content-Type", "X-
Requested-With"));
      configuration.setExposedHeaders(List.of("Authorization"));
      configuration.setAllowCredentials(true);
      configuration.setMaxAge(3600L);
      UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
      source.registerCorsConfiguration("/**", configuration);
      return source;
   }
```

```
    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService);
        authProvider.setPasswordEncoder(passwordEncoder());
        return authProvider;
    }
    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration config)
throws Exception {
        return config.getAuthenticationManager();
    }
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
                                                    }
```

ДОДАТОК Б

Файл DashboardPage

```
import { useState, useEffect } from 'react';
import { Link } from 'react-router-dom';
import { productsApi } from '../api/products';
import { sourcesApi } from '../api/sources';
import { notificationsApi } from '../api/notifications';
import { Product, Source, Notification } from '../types';
const DashboardPage = () => {
  const [products, setProducts] = useState<Product[]>([]);
  const [sources, setSources] = useState<Source[]>([]);
  const [notifications, setNotifications] = useState<Notification[]>([]);
  const [isLoading, setIsLoading] = useState(true);
  useEffect(() => {
    const fetchData = async () => {
      try {
        const [productsRes, sourcesRes, notificationsRes] = await Promise.all([
          productsApi.getAll({ size: 5, sortBy: 'lastScrapedAt', sortDirection: 'DESC' }),
          sourcesApi.getAll(),
          notificationsApi.getAll(0, 5),
        ]);
        setProducts(productsRes.content);
        setSources(sourcesRes);
        setNotifications(notificationsRes.content);
      } catch (error) {
        console.error('Failed to fetch dashboard data', error);
      } finally {
        setIsLoading(false);
      }
    };
    fetchData();
  }, []);
  const totalProducts = products.length;
  const activeSources = sources.filter((s) => s.isActive).length;
  const unreadNotifications = notifications.filter((n) => !n.isRead).length;
  if (isLoading) {
    return (
      <div className="flex items-center justify-center h-64">
        <div  className="animate-spin  rounded-full  h-12  w-12  border-b-2  border-primary-
600"></div>
      </div>
```

```
  );
 }
 return (
  <div>
   <h1 className="page-title">Dashboard</h1>
   {/* Stats Cards */}
   <div className="grid grid-cols-1 md:grid-cols-3 gap-6 mb-8">
    <div className="card">
     <div className="flex items-center justify-between">
      <div>
       <p className="text-sm text-gray-500">Total Products</p>
       <p className="text-3xl font-bold text-gray-900">{totalProducts}</p>
      </div>
      <div className="text-4xl">📦</div>
     </div>
     <Link to="/dashboard/products" className="text-primary-600 text-sm mt-4 inline-block hover:underline">
      View all products →
     </Link>
    </div>
    <div className="card">
     <div className="flex items-center justify-between">
      <div>
       <p className="text-sm text-gray-500">Active Sources</p>
       <p className="text-3xl font-bold text-gray-900">{activeSources}</p>
      </div>
      <div className="text-4xl">🌐</div>
     </div>
     <Link to="/dashboard/sources" className="text-primary-600 text-sm mt-4 inline-block hover:underline">
      Manage sources →
     </Link>
    </div>
    <div className="card">
     <div className="flex items-center justify-between">
      <div>
       <p className="text-sm text-gray-500">Unread Notifications</p>
       <p className="text-3xl font-bold text-gray-900">{unreadNotifications}</p>
      </div>
      <div className="text-4xl">🔔</div>
     </div>
     <Link to="/dashboard/notifications" className="text-primary-600 text-sm mt-4 inline-block hover:underline">
      View notifications →
     </Link>
```

```
        </div>
      </div>
      {/* Recent Products */}
      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
        <div className="card">
          <h2 className="text-lg font-semibold mb-4">Recently Updated Products</h2>
          {products.length === 0 ? (
            <div className="text-center py-8 text-gray-500">
              <p>No products yet</p>
              <Link to="/dashboard/products" className="text-primary-600 hover:underline">
                Add your first product
              </Link>
            </div>
          ) : (
            <div className="space-y-4">
              {products.map((product) => (
                <Link
                  key={product.id}
                  to={`/dashboard/products/${product.id}`}
                  className="flex items-center p-3 rounded-lg hover:bg-gray-50 transition-colors"
                >
                  <div className="w-12 h-12 bg-gray-100 rounded-lg flex items-center justify-center
mr-4">
                    {product.imageUrl ? (
                      <img
                        src={product.imageUrl}
                        alt={product.name}
                        className="w-full h-full object-cover rounded-lg"
                      />
                    ) : (
                      <span className="text-2xl">📦</span>
                    )}
                  </div>
                  <div className="flex-1 min-w-0">
                    <p className="font-medium text-gray-900 truncate">{product.name}</p>
                    <p className="text-sm text-gray-500">{product.source.name}</p>
                  </div>
                  <div className="text-right">
                    <p className="font-semibold text-gray-900">
                      {product.currentPrice?.toFixed(2)} {product.currency}
                    </p>
                  </div>
                </Link>
              ))}
            </div>
```

```
          )}
        </div>
        {/* Recent Notifications */}
        <div className="card">
          <h2 className="text-lg font-semibold mb-4">Recent Notifications</h2>
          {notifications.length === 0 ? (
            <div className="text-center py-8 text-gray-500">
              <p>No notifications yet</p>
              <p className="text-sm">You'll see price alerts here</p>
            </div>
          ) : (
            <div className="space-y-4">
              {notifications.map((notification) => (
                <div
                  key={notification.id}
                  className={`p-3 rounded-lg ${notification.isRead ? 'bg-gray-50' : 'bg-primary-50'}`}
                >
                  <div className="flex items-start">
                    <span className="text-xl mr-3">
                      {notification.type === 'PRICE_DROP' && '📉'}
                      {notification.type === 'PRICE_INCREASE' && '📈'}
                      {notification.type === 'THRESHOLD_REACHED' && '🎯'}
                      {notification.type === 'SCRAPING_ERROR' && '⚠️'}
                    </span>
                    <div className="flex-1 min-w-0">
                      <p className="text-sm text-gray-800">{notification.message}</p>
                      <p className="text-xs text-gray-500 mt-1">
                        {new Date(notification.createdAt).toLocaleDateString()}
                      </p>
                    </div>
                  </div>
                </div>
              ))}
            </div>
          )}
        </div>
      </div>
    </div>
  );
};
export default DashboardPage;
```

ДОДАТОК В

Сервіс ScrapingService

```
package com.pricetracker.service;

import com.pricetracker.dto.ProductDto;
import com.pricetracker.entity.*;
import com.pricetracker.exception.ResourceNotFoundException;
import com.pricetracker.repository.*;
import com.pricetracker.scraper.*;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.time.LocalDateTime;
import java.util.List;
import java.util.Map;
import java.util.function.Function;
import java.util.stream.Collectors;
@Service
@RequiredArgsConstructor
@Slf4j
public class ScrapingService {
    private final ProductRepository productRepository;
    private final PriceHistoryRepository priceHistoryRepository;
    private final NotificationRepository notificationRepository;
    private final UserNotificationSettingsRepository notificationSettingsRepository;
    private final UserRepository userRepository;
    private final SourceRepository sourceRepository;
    private final List<Scraper> scrapers;
    private final ScrapingRateLimiter rateLimiter;
    private Map<ScraperType, Scraper> scraperMap;
    @jakarta.annotation.PostConstruct
    public void init() {
        scraperMap = scrapers.stream()
                .collect(Collectors.toMap(Scraper::getType, Function.identity()));
    }
    @Transactional
    public ProductDto scrapeProduct(Long productId, String userEmail) {
        User user = userRepository.findByEmail(userEmail)
```

```
            .orElseThrow(() -> new ResourceNotFoundException("User", "email", userEmail));
    Product product = productRepository.findByIdAndUserId(productId, user.getId())
            .orElseThrow(() -> new ResourceNotFoundException("Product", "id", productId));
    return scrapeAndUpdateProduct(product);
}
@Transactional
public ProductDto scrapeAndUpdateProduct(Product product) {
    Source source = product.getSource();
    Scraper scraper = scraperMap.get(source.getScraperType());
    if (scraper == null) {
        log.error("No scraper found for type: {}", source.getScraperType());
        return ProductDto.fromEntity(product);
    }
    // Apply rate limiting
    rateLimiter.waitForPermission(product.getProductUrl());
    SelectorConfig selectors = SelectorConfig.builder()
            .priceSelector(source.getPriceSelector())
            .titleSelector(source.getTitleSelector())
            .imageSelector(source.getImageSelector())
            .build();
    ScrapingResult result = scraper.scrape(product.getProductUrl(), selectors);
    if (result.isSuccess()) {
        BigDecimal oldPrice = product.getCurrentPrice();
        BigDecimal newPrice = result.getPrice();
        // Create price history entry
        PriceHistory priceHistory = PriceHistory.builder()
                .product(product)
                .price(newPrice)
                .currency(result.getCurrency())
                .scrapedAt(LocalDateTime.now())
                .build();
        priceHistoryRepository.save(priceHistory);
        // Update product
        product.setCurrentPrice(newPrice);
        product.setCurrency(result.getCurrency());
        product.setLastScrapedAt(LocalDateTime.now());
        if (result.getTitle() != null && !result.getTitle().isBlank()) {
            product.setName(result.getTitle());
        }
        if (result.getImageUrl() != null && !result.getImageUrl().isBlank()) {
            product.setImageUrl(result.getImageUrl());
        }
        product = productRepository.save(product);
        // Update source last scraped time
        source.setLastScrapedAt(LocalDateTime.now());
        sourceRepository.save(source);
```

```
            // Check if notification should be created
            if (oldPrice != null && newPrice != null) {
                checkAndCreateNotification(product, oldPrice, newPrice);
            }
            log.info("Successfully scraped product '{}': {} {}", product.getName(), newPrice,
result.getCurrency());
        } else {
            log.error("Failed to scrape product '{}': {}", product.getName(),
result.getErrorMessage());
            // Create error notification
            createErrorNotification(product, result.getErrorMessage());
        }
        return ProductDto.fromEntity(product);
    }
    private void checkAndCreateNotification(Product product, BigDecimal oldPrice, BigDecimal
newPrice) {
        UserNotificationSettings settings =
notificationSettingsRepository.findByUserId(product.getUser().getId())
                .orElse(null);
        if (settings == null) {
            return;
        }
        double percentageChange = calculatePercentageChange(oldPrice, newPrice);
        // Check for price drop
        if (settings.isPriceDropEnabled() && newPrice.compareTo(oldPrice) < 0) {
            if (Math.abs(percentageChange) >= settings.getDiscountThreshold()) {
                createPriceDropNotification(product, oldPrice, newPrice, percentageChange);
            }
        }

        // Check for price increase
        if (settings.isPriceIncreaseEnabled() && newPrice.compareTo(oldPrice) > 0) {
            createPriceIncreaseNotification(product, oldPrice, newPrice, percentageChange);
        }
        // Check for threshold reached
        if (settings.getMinPriceAlert() != null &&
newPrice.compareTo(settings.getMinPriceAlert()) <= 0) {
            createThresholdNotification(product, newPrice, "below", settings.getMinPriceAlert());
        }
        if (settings.getMaxPriceAlert() != null &&
newPrice.compareTo(settings.getMaxPriceAlert()) >= 0) {
            createThresholdNotification(product, newPrice, "above", settings.getMaxPriceAlert());
        }
    }
    private double calculatePercentageChange(BigDecimal oldPrice, BigDecimal newPrice) {
        if (oldPrice.compareTo(BigDecimal.ZERO) == 0) {
```

```
            return 0;
        }
        return newPrice.subtract(oldPrice)
                .divide(oldPrice, 4, RoundingMode.HALF_UP)
                .multiply(BigDecimal.valueOf(100))
                .doubleValue();
    }
    private void createPriceDropNotification(Product product, BigDecimal oldPrice, BigDecimal
newPrice, double percentageChange) {
        Notification notification = Notification.builder()
                .user(product.getUser())
                .product(product)
                .type(NotificationType.PRICE_DROP)
                .message(String.format("Price dropped for '%s' from %.2f to %.2f %s (%.1f%% off)",
                        product.getName(), oldPrice, newPrice, product.getCurrency(),
Math.abs(percentageChange)))
                .previousPrice(oldPrice)
                .newPrice(newPrice)
                .percentageChange(percentageChange)
                .build();
        notificationRepository.save(notification);
        log.info("Created price drop notification for product '{}'", product.getName());
    }
    private void createPriceIncreaseNotification(Product product, BigDecimal oldPrice,
BigDecimal newPrice, double percentageChange) {
        Notification notification = Notification.builder()
                .user(product.getUser())
                .product(product)
                .type(NotificationType.PRICE_INCREASE)
                .message(String.format("Price increased for '%s' from %.2f to %.2f %s (+%.1f%%)",
                        product.getName(), oldPrice, newPrice, product.getCurrency(),
percentageChange))
                .previousPrice(oldPrice)
                .newPrice(newPrice)
                .percentageChange(percentageChange)
                .build();
        notificationRepository.save(notification);
        log.info("Created price increase notification for product '{}'", product.getName());
    }
    private void createThresholdNotification(Product product, BigDecimal newPrice, String
direction, BigDecimal threshold) {
        Notification notification = Notification.builder()
                .user(product.getUser())
                .product(product)
                .type(NotificationType.THRESHOLD_REACHED)
                .message(String.format("Price for '%s' is now %s your threshold: %.2f %s (threshold:
```

```
%.2f)",
                    product.getName(), direction, newPrice, product.getCurrency(), threshold))
            .newPrice(newPrice)
            .build();
    notificationRepository.save(notification);
    log.info("Created threshold notification for product '{}'", product.getName());
  }
  private void createErrorNotification(Product product, String errorMessage) {
    Notification notification = Notification.builder()
            .user(product.getUser())
            .product(product)
            .type(NotificationType.SCRAPING_ERROR)
            .message(String.format("Failed to scrape '%s': %s", product.getName(), errorMessage))
            .build();
    notificationRepository.save(notification);
  }
  @Transactional
  public void scrapeAllActiveProducts() {
    List<Product> activeProducts = productRepository.findByIsActiveTrue();
    log.info("Starting scheduled scraping for {} active products", activeProducts.size());
    for (Product product : activeProducts) {
      try {
        scrapeAndUpdateProduct(product);
      } catch (Exception e) {
        log.error("Error scraping product '{}': {}", product.getName(), e.getMessage());
      }
    }
    log.info("Completed scheduled scraping");
  }
}
```

ДОДАТОК Г

Файл SeleniumScraper

```
package com.pricetracker.scraper;
import com.pricetracker.entity.ScraperType;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
import java.math.BigDecimal;
import java.net.URL;
import java.time.Duration;
@Component
@RequiredArgsConstructor
@Slf4j
public class SeleniumScraper implements Scraper {
    private final PriceParser priceParser;
    @Value("${scraping.selenium.url}")
    private String seleniumUrl;
    @Value("${scraping.selenium.timeout:30000}")
    private int timeout;
    @Override
    public ScrapingResult scrape(String url, SelectorConfig selectors) {
        WebDriver driver = null;
        try {
            log.info("Scraping URL with Selenium: {}", url);
            ChromeOptions options = new ChromeOptions();
            options.addArguments("--headless");
            options.addArguments("--no-sandbox");
            options.addArguments("--disable-dev-shm-usage");
            options.addArguments("--disable-gpu");
            options.addArguments("--window-size=1920,1080");
            options.addArguments("--user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36");
            driver = new RemoteWebDriver(new URL(seleniumUrl), options);
```

```
        driver.manage().timeouts().pageLoadTimeout(Duration.ofMillis(timeout));
        driver.get(url);
        WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
        // Extract price
        BigDecimal price = null;
        String currency = "UAH";
        if (selectors.getPriceSelector() != null && !selectors.getPriceSelector().isBlank()) {
            try {
                WebElement priceElement = wait.until(
                        ExpectedConditions.presenceOfElementLocated(By.cssSelector(selectors.getPriceSelector()))
                );
                String priceText = priceElement.getText();
                price = priceParser.parsePrice(priceText);
                currency = priceParser.extractCurrency(priceText);
                log.debug("Extracted price: {} {}", price, currency);
            } catch (Exception e) {
                log.warn("Failed to extract price element", e);
            }
        }
        // Extract title
        String title = null;
        if (selectors.getTitleSelector() != null && !selectors.getTitleSelector().isBlank()) {
            try {
                WebElement titleElement = driver.findElement(By.cssSelector(selectors.getTitleSelector()));
                title = titleElement.getText();
                log.debug("Extracted title: {}", title);
            } catch (Exception e) {
                log.warn("Failed to extract title element", e);
            }
        }
        // Extract image URL
        String imageUrl = null;
        if (selectors.getImageSelector() != null && !selectors.getImageSelector().isBlank()) {
            try {
                WebElement imageElement = driver.findElement(By.cssSelector(selectors.getImageSelector()));
                imageUrl = imageElement.getAttribute("src");
                if (imageUrl == null || imageUrl.isEmpty()) {
                    imageUrl = imageElement.getAttribute("data-src");
                }
                log.debug("Extracted image URL: {}", imageUrl);
            } catch (Exception e) {
                log.warn("Failed to extract image element", e);
            }
```

```
            }
            if (price == null) {
                return ScrapingResult.failure("Failed to extract price from page");
            }
            return ScrapingResult.success(price, currency, title, imageUrl);
        } catch (Exception e) {
            log.error("Failed to scrape URL with Selenium: {}", url, e);
            return ScrapingResult.failure("Scraping failed: " + e.getMessage());
        } finally {
            if (driver != null) {
                try {
                    driver.quit();
                } catch (Exception e) {
                    log.warn("Failed to quit WebDriver", e);
                }
            }
        }
    }
    @Override
    public ScraperType getType() {
        return ScraperType.SELENIUM;
    }
}
```