Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

РОЗРОБКА ТА ДОСЛІДЖЕННЯ ЗАСТОСУНКУ ДЛЯ ПАРСИНГУ ЛОГІСТИЧНИХ ПОРТАЛІВ

DEVELOPMENT AND RESEARCH OF AN APPLICATIONFOR PARSING LOGISTICS PORTALS

спеціальність 122 Комп'ютерні науки

освітня програма «Комп'ютерні науки»

Виконав: здобувач вищої освіти
групи КНм-21
Миронюк Олександр Вікторович
_____
(підпис)
Керівник: к.т.н, доцент
Ліщина Валерій Олександрович
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Валерій Олександрович
_____
(підпис)

Луцьк – 2025

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерних наук*
Ступінь вищої освіти: *магістр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *122 Комп'ютерні науки*
Освітня програма: *«Комп'ютерні науки»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Валерій ЛІЩИНА
«14» травня 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Миронюк Олександр Вікторович

1. Тема кваліфікаційної роботи: Розробка та дослідження застосунку для парсингу логістичних порталів

Керівник роботи: к.т.н., доцент Ліщина В.О.

затверджені наказом закладу вищої освіти від «14» травня 2025 р. № 255/01-02

2. Строк подання здобувачем роботи: 05.12.2025 р.

3.Вихідні дані до роботи: Документація та специфікації стандартів і протоколів web-розробки, що визначають принципи взаємодії клієнтських застосунків із web-ресурсами (HTTP/HTTPS, REST API, WebSocket, JSON, CORS, механізми авторизації та cookie-керування) Дані про структуру та особливості логістичних порталів, що виступають джерелами інформації: структура сторінок, параметри пошуку вантажів і транспорту, формат публікацій, механізми фільтрації, пагінації, оновлення даних та елементи захисту від автоматизованих запитів.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, аналіз і вибір засобів проектування, опис функціонального наповнення об'єкта проектування, розробка й обґрунтування основних задач.

5. Перелік графічного матеріалу: схема бази даних, інтерфейс користувача, Таблиці порівняльного аналізу з існуючими рішеннями

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Розділ 1 Теоретичні засади автоматизованого збору даних з логістичних порталів	*Ліщина В .О.*		
Розділ 2 Проєктування та розроблення застосунку для парсингу логістичних порталів	*Ліщина В. О.*		
Розділ 3 Тестування, оцінювання ефективності та впровадження застосунку	*Ліщина В. О.*		
Показник запозичень тексту	_____ %		
Інструментальна перевірка	*Кошелюк В. А.*		
Нормоконтроль	*Сачук В. О.*		
Гарант ОПП	*Ліщина В. О.*		

7. Дата видачі завдання *«14» травня 2025 р.*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1.	*Провести огляд літературних джерел по темі кваліфікаційної роботи*	*до 30.06.2025 р*	
2.	*Провести аналіз загальної проблеми і вибір напрямків дослідження*	*до 01.09.2025 р.*	
3.	*Розробити функціональну схему роботи програмного продукту*	*до 01.10.2025 р*	
4.	*Описати засоби розробки об'єкта проектування*	*до 15.10.2025 р.*	
5.	*Практична реалізація об'єкта проектування*	*до 10.11.2025 р*	
6.	*Провести аналіз результатів експерименту*	*до 25.11.2025 р.*	
7.	*Здача чистового варіанту кваліфікаційної роботи на кафедру*	*до 05.12.2025 р.*	

Здобувач вищої освіти __ _____ Миронюк О. В.

Керівник роботи _____ Ліщина В. О.

АНОТАЦІЯ

Миронюк О. В. Кваліфікаційна робота магістра ОП «Комп’ютерні науки», спеціальність 122 «Комп’ютерні науки». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота магістра складається зі вступу, трьох розділів, висновків, пропозицій, списку використаних джерел та додатків.

У роботі досліджено актуальну проблему автоматизації збору логістичних даних у сучасних умовах цифровізації транспортних процесів. Проаналізовано особливості логістичних порталів, їхню архітектуру, динамічні механізми формування контенту, рівні захисту та специфіку доступу до API. Розкрито сучасні підходи й алгоритми web-скрапінгу, включаючи статичний, динамічний і API-орієнтований парсинг, а також інструменти для обходу антибот-захисту та методи нормалізації неструктурованих даних.

Запропоновано архітектуру застосунку для автоматизованого збору даних із використанням технологій Node.js, Playwright і MongoDB, аргументовано вибір технологічного стеку та моделювання ключових компонентів системи. Проведено комплексне тестування застосунку, включаючи функціональне, навантажувальне, модульне та інтеграційне тестування. Виконано оцінювання продуктивності модулів, аналіз стабільності роботи при різних режимах навантаження та визначено точність отриманих даних.

У результаті дослідження створено застосунок, що автоматично збирає та обробляє логістичні дані майже в реальному часі. Результати підтвердили його ефективність і готовність до впровадження для прискорення пошуку перевезень та підвищення точності аналітики.

Ключові слова: web-скрапінг, логістичні портали, Node.js, Playwright, MongoDB, автоматизований збір даних, цифрова логістика, парсинг, аналіз великих даних.

ABSTRACT

Oleksandr Myroniuk. Master's thesis in Computer Science, specialty 122 «Computer Science» Lutsk National Technical University. Lutsk, 2025.

The master's thesis consists of an introduction, three chapters, conclusions, proposals, a list of references, and appendices.

The thesis examines the topical issue of automating the collection of logistics data in the current context of the digitalization of transport processes. It analyzes the features of logistics portals, their architecture, dynamic content formation mechanisms, security levels, and the specifics of API access. Modern approaches and algorithms for web scraping are revealed, including static, dynamic, and API-oriented parsing, as well as tools for bypassing anti-bot protection and methods for normalizing unstructured data.

An application architecture for automated data collection using Node.js, Playwright, and MongoDB technologies was proposed, and the choice of the technology stack and modeling of key system components were justified. Comprehensive testing of the application was carried out, including functional, load, modular, and integration testing. The performance of the modules was evaluated, the stability of operation under different load modes was analyzed, and the accuracy of the data obtained was determined.

As a result of the research, an application was created that automatically collects and processes logistics data in near real time. The results confirmed its effectiveness and readiness for implementation to speed up the search for transportation and improve the accuracy of analytics.

Keywords: web scraping, logistics portals, Node.js, Playwright, MongoDB, automated data collection, digital logistics, parsing, big data analysis.

ЗМІСТ

ВСТУП

Сучасний глобалізований ринок логістичних послуг характеризується високою динамічністю, збільшенням обсягів міжнародних перевезень та активною цифровізацією транспортно-логістичних процесів. Стрімке зростання електронної комерції, розвиток мультимодальних перевезень і посилена конкуренція на ринку транспортних послуг формують нові вимоги до швидкодії, точності та аналітичної спроможності логістичних компаній. У таких умовах автоматизований збір даних із логістичних порталів стає необхідною складовою сучасних систем управління перевезеннями, оскільки забезпечує оперативне отримання інформації про ставки, маршрути, доступний транспорт, затримки та інші критично важливі параметри. Провідні міжнародні сервіси – Project44, Sixfold, Trans.eu, Timocom, SeaRates, Cargopedia, Freightos – лише за останні роки суттєво розширили обсяги відкритих логістичних даних, інтегрувавши можливості API, інструменти відстеження та системи оптимізації маршрутів, що створює сприятливі умови для автоматизації моніторингу ринку та прогнозування логістичних потоків.

Попри наявність таких платформ, наукова проблема полягає у високій різнорідності та фрагментарності цифрових даних. Значна частина інформації на логістичних порталах генерується динамічно: контент завантажується за допомогою JavaScript, API-запитів, модулів асинхронного оновлення, що суттєво ускладнює процеси класичного парсингу. Роботи сучасних дослідників, зокрема Mitchell (2023), Meier (2022), Glez-Peña (2020), Munzert (2021), підтверджують, що web-скрапінг стає одним із провідних інструментів Data Engineering, але для логістичних порталів він вимагає спеціалізованих підходів, врахування технічних обмежень, антибот-захисту та специфіки структури даних. Усі ці аспекти визначають складність і наукову актуальність розроблення універсального застосунку для автоматизованого збору логістичної інформації.

Системи управління логістичними процесами дедалі більше залежать від швидкого доступу до актуальної інформації, оскільки транспортно-логістична

сфера зазнає постійних змін, пов’язаних із глобалізацією економіки, розвитком цифрових технологій, інформаційними ризиками та зростанням ролі аналітичних платформ. Поява великої кількості логістичних порталів і транспортних бірж не лише спростила процес комунікації між перевізниками, замовниками та експедиторами, але й створила масштабні масиви даних, що потребують спеціалізованих засобів збору, аналізу та структурування. Сьогодні корпоративні логістичні системи не можуть ефективно функціонувати лише на основі ручного моніторингу інформаційних джерел: затримка в отриманні даних або їх неповнота прямо впливають на якість управлінських рішень, рівень ризиків та конкурентоспроможність компанії.

Побудова комплексних логістичних рішень ґрунтується на засадах цифрової трансформації, про що свідчать звіти провідних світових компаній, включаючи McKinsey (2023), DHL (2024), Gartner (2023). У них зазначається, що логістика вже не розглядається виключно як операційна діяльність – це інформаційно насичений процес, у якому дані та швидкість їхнього отримання стають основним фактором досягнення ефективності. У цьому контексті web-скрапінг – технологія автоматизованого вилучення структурованої інформації з вебресурсів – стає ключовим інструментом формування даних для логістичних моделей і систем підтримки прийняття рішень.

Суттєвий внесок у дослідження технологій web-скрапінгу зробили реальні науковці, такі як Mitchell (2023), який аналізує складність вилучення інформації з динамічних вебінтерфейсів; Munzert (2021), що розглядає web-скрапінг у контексті обробки великих даних; Glez-Peña (2020), який досліджує питання очищення та нормалізації даних; Meier (2022), що аналізує інструменти headless-браузерів, у тому числі Puppeteer і Playwright. Ці праці утворюють методологічне підґрунтя, яке дозволяє осмислити можливості та обмеження автоматизації збору логістичної інформації.

З практичного боку, логістичні портали на зразок Trans.eu, Timocom, SeaRates, Cargopedia, Freightos, Project44 і Sixfold формують потужний інформаційний простір, у якому щосекунди оновлюються дані про вантажі,

ставки, доступні транспортні засоби, маршрути перевезень, характеристики контейнерів та інші важливі параметри. Такі сервіси використовують складну архітектуру, що включає API, REST-запити, WebSockets, динамічний DOM із JavaScript-рендерингом, системи кешування та серверні фреймворки типу React, Vue або Angular. Отже, класичні методи HTML-парсингу, ефективні десять років тому, сьогодні вже не забезпечують доступ до релевантних даних – для цього потрібні гібридні технології, що поєднують інструменти аналізу API та headless-браузерні рішення.

В Україні розвиток логістичної сфери зазнав значної трансформації у зв'язку з повномасштабною війною та перебудовою транспортних коридорів. Доступ до актуальних даних став критично важливим для стабільності експорту, імпорту та гуманітарних перевезень. За даними Міністерства розвитку громад, територій та інфраструктури України (2023-2024 рр.), цифровізація логістичних процесів визначена стратегічним пріоритетом. У той же час більшість вітчизняних компаній, особливо малого та середнього бізнесу, продовжують опиратися на ручний моніторинг транспортних платформ, що знижує ефективність їхньої діяльності, спричиняє втрату часу та створює додаткові ризики.

Створення автоматизованого застосунку, здатного збирати й структурувати дані з різних логістичних джерел, дозволяє вирішити низку системних проблем. По-перше, це усуває залежність від людського фактора, зменшує ймовірність помилок і прискорює процес прийняття рішень. По-друге, забезпечує доступ до більшої кількості варіантів перевезень, ніж може опрацювати навіть великий відділ логістів вручну. По-третє, уможливлює інтеграцію отриманих даних у системи аналітики, моделі прогнозування, CRM чи ERP-системи.

В українських умовах потреба у таких інструментах особливо зросла через трансформацію логістичних коридорів, переорієнтацію міжнародних потоків та зростання значення транспортної аналітики у період повномасштабної війни. Згідно з даними Міністерства розвитку громад, територій та інфраструктури

України за 2023-2024 роки, цифро візувала логістичні процеси серед ключових напрямів державної політики, а бізнес-потреба у швидкому доступі до відкритих транспортних даних зростає щорічно.

Актуальність дослідження визначається потребою у створенні програмного рішення, здатного забезпечити швидкий, точний і масштабований збір логістичних даних. Аналітичний звіт McKinsey & Company (2023) демонструє, що компанії, які впровадили автоматизовані цифрові інструменти моніторингу ринку перевезень, скоротили час на пошук актуальних пропозицій на 45-60 %, одночасно підвищивши точність планування та зменшивши логістичні витрати. Дослідження Rodrigues & Notteboom (2022) та аналітика DHL Logistics Trend Radar 2024 свідчать, що data-driven логістика є ключовим трендом індустрії, а якість управлінських рішень безпосередньо залежить від наявності актуальних та достовірних даних. Саме тому створення інструменту автоматизованого збору логістичної інформації відповідає як науковим, так і практичним потребам галузі.

Метою магістерської роботи є розроблення застосунку для автоматизованого збору, обробки та збереження даних із логістичних порталів, що забезпечує їх актуальність, структурованість і придатність для подальшої аналітики та прийняття управлінських рішень. Досягнення поставленої мети передбачає виконання низки завдань, серед яких – аналіз структури логістичних порталів, дослідження методів web-скрапінгу, порівняння існуючих рішень, обґрунтування архітектури застосунку, розроблення модулів збору й обробки даних, створення користувацького інтерфейсу, тестування, верифікація та експериментальне оцінювання ефективності розробленої системи.

Предметом дослідження є методи, алгоритми та програмні засоби web-скрапінгу, а також технологічні рішення, що забезпечують ефективний збір, обробку та нормалізацію логістичних даних для подальшої аналітики.

Об’єктом дослідження є процес автоматизованого збору даних із логістичних платформ, а предметом – методи, алгоритми та програмні засоби web-скрапінгу, що застосовуються для створення комплексного інструменту

автоматизованого моніторингу логістичної інформації. Для досягнення мети використано методи порівняльного аналізу логістичних порталів та їх API, структурного аналізу HTML-DOM, реверс-інжинірингу вебзапитів, алгоритмічні методи web-скрапінгу із застосуванням Requests, Puppeteer, Playwright, Scrapy, методи очищення та нормалізації даних, експериментальні підходи до тестування продуктивності та статистичні методи аналізу результатів.

Наукова новизна роботи полягає у формуванні уніфікованої моделі збору даних із різнорідних логістичних порталів з динамічно генерованим контентом, застосуванні комбінованого підходу, що об’єднує headless-браузерний парсинг та аналіз API-запитів, а також у розробленні моделі нормалізації логістичної інформації для подальшого використання у задачах аналітики та прогнозування. Новизна також зумовлена виконанням комплексного експериментального оцінювання стабільності, швидкодії та можливостей масштабування розробленого застосунку.

Практичне значення результатів полягає у можливості використання створеної системи логістичними компаніями, експедиторами, операторами перевезень та аналітичними підрозділами для автоматизованого моніторингу ринкових ставок, маршрутів і транспортних пропозицій, а також для прискорення пошуку логістичних рішень, підвищення точності планування та інтеграції з аналітичними інструментами підтримки прийняття рішень.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ЗАСАДИ АВТОМАТИЗОВАНОГО ЗБОРУ ДАНИХ З ЛОГІСТИЧНИХ ПОРТАЛІВ

1.1 Характеристика логістичних порталів та особливості структури їх даних

Цифрові логістичні портали стали невід'ємною частиною глобальної логістичної інфраструктури, оскільки забезпечують учасникам транспортних процесів доступ до актуальної, динамічної та структурованої інформації про вантажі, транспортні засоби, вартість перевезень, маршрути, пропускну здатність коридорів та аналітику ринку. На тлі зростання обсягів міжнародної торгівлі та глобальної мобільності логістичних потоків значення цих платформ постійно збільшується. Переорієнтація транспортних маршрутів, поява цифрових коридорів та активний розвиток концепції «end-to-end visibility» спричинили те, що цифрові портали фактично перетворилися на інтелектуальні інформаційно-комунікаційні системи. Вони оперують великими обсягами даних, оновлюваних у режимі реального часу, що створює як нові можливості для користувачів, так і суттєві технологічні виклики для розробників засобів автоматизованого збору інформації.

На сучасному етапі розвитку логістичної цифровізації портали такі як Trans.eu, Timocom, SeaRates, Cargopedia, Freightos, ShipHub, Project44, FourKites та MarineTraffic утворюють багатокомпонентну екосистему, у межах якої транспортні компанії, експедитори, логістичні оператори та вантажовласники здійснюють пошук вантажів, організацію перевезень, аналіз логістичних показників та вибір оптимальних маршрутів. Кожен із цих порталів має власну специфіку, концентруючи у собі різні категорії даних – від тарифів до геолокаційних потоків інформації. Саме тому розуміння їх архітектури, структури даних та способів формування контенту є важливою передумовою створення ефективного програмного забезпечення для автоматизованого збору.

Однією з провідних платформ у європейському логістичному просторі є Trans.eu [1]. Її функціональні можливості охоплюють пошук вантажів та транспорту, укладання угод, перевірку перевізників, управління документами, а також моніторинг логістичних операцій. Важливою характеристикою даного порталу є динамічне формування контенту: інформація про вантажі, ставки та маршрути передається через асинхронні мережеві запити, що підвантажують нові дані без перезавантаження сторінки. Це означає, що під час автоматизованого збору даних неможливо обмежитися класичним HTML-парсингом, оскільки статичний HTML містить мінімум корисної інформації. Парсинг можливий лише за умови аналізу відповідей API або моделювання поведінки браузера за допомогою headless-інструментів. Крім того, Trans.eu активно застосовує механізми антибот-захисту, що створює додаткові технічні обмеження.

Портал Timocom – один із найстаріших і найпотужніших логістичних цифрових ресурсів Європейського Союзу. Він працює за моделлю «Smart Logistics System», яка передбачає інтеграцію низки функціональних модулів: біржі вантажів, транспортного довідника, системи документообігу, перевірки взаємодій між контрагентами. Особливістю Timocom є підвищений рівень безпеки й закритості даних: платформа має кілька рівнів авторизації, використовує складні токени доступу та шифрування приватної інформації, що накладає суттєві обмеження на автоматизований збір даних. Лише незначна частина інформації доступна у вигляді статичного HTML, тоді як решта інформації формується на основі JavaScript і динамічних модулів інтерфейсу, що ускладнює розроблення програмних інструментів збору даних.

У сфері морських перевезень одним із найбільш функціональних логістичних сервісів є SeaRates, який надає інструменти для визначення тарифів, вибору маршрутів, розрахунку часу в дорозі, аналізу портової інфраструктури та контейнерного трекінгу [2]. Дані SeaRates будуються на основі зовнішніх API, що надходять від сотень операторів морських ліній та портів. Потокова структура інформації включає таблиці, географічні карти, інтерактивні шари та JSON-відповіді. Це робить портал інформаційно багатим, але технічно складним

для автоматизованого опрацювання, оскільки програмний модуль повинен уміти опрацьовувати великі обсяги даних, які надходять із різних джерел.

Глобальна система Freightos, яка працює у сфері міжнародної експрес-логістики, пропонує унікальний формат агрегування тарифів та маршрутів. Вона використовується для бронювання повітряних, морських і комбінованих перевезень. Дані Freightos, як правило, структуровані й доступні через API, однак доступ до інтерфейсів платформи є комерційним і потребує автентифікації. Це означає, що автоматизований збір даних можливий лише за наявності офіційного доступу. Структура даних включає тарифи, характеристики маршрутів, інформацію про перевізників та часові параметри доставки.

Окремий клас логістичних платформ становлять системи видимості ланцюгів постачання, такі як Project44 та FourKites [3-4]. Ці сервіси забезпечують відстеження вантажів практично в режимі реального часу й використовуються великими корпораціями для мінімізації затримок, управління ризиками та оптимізації операційних процесів. Вони оперують даними у форматах JSON та XML, а інформація формується на основі прямих інтеграцій із перевізниками, портами, авіалініями та інфраструктурними системами. Доступ до API цих сервісів є платним та надається виключно в рамках корпоративного контракту і навчальних цілях для ознайомлення.

До групи цифрових сервісів, важливих для морської логістики, належать системи AIS-відстеження, наприклад MarineTraffic, які фіксують положення суден, маршрути, швидкість руху та час перебування у портах. AIS-дані надходять у режимі реального часу, обробляються та відображаються на інтерактивних картах. Структура даних таких порталів включає координати, часові значення, параметри суден та статуси рейсів. Аналіз таких наборів даних потребує спеціалізованих алгоритмів очищення та нормалізації.

Незважаючи на відмінності між зазначеними платформами, їх об’єднує низка спільних характеристик, що визначають специфіку роботи із логістичними даними. Однією з ключових рис є динамічний характер контенту, оскільки більшість інформаційних елементів формується у браузері за допомогою

JavaScript, а не завантажується у вигляді статичних HTML-сторінок. Дослідження Mitchell (2023) свідчить, що понад 70 % сучасних вебплатформ використовують клієнтський рендеринг, що практично унеможливлює традиційний підхід до вебпарсингу. У таких умовах для отримання інформації потрібні симулятори браузерів, які можуть виконувати код та аналізувати мережеві запити [5].

Другою спільною рисою логістичних порталів є інтенсивне використання API. Доступ до інформації забезпечується через JSON/XML-відповіді, що передаються мережею у відповідь на XHR або Fetch-запити. У випадку логістичних порталів API можуть містити десятки або сотні результатів, кожен з яких відповідає за певну категорію даних: тарифи, рейси, водії, транспорт, маршрути, повідомлення тощо. Використання API є більш ефективним, але не завжди можливим, оскільки більшість порталів обмежує доступ до нього.

Ще однією важливою технічною характеристикою логістичних систем є високий рівень захисту від автоматизованих запитів. Платформи впроваджують CAPTCHA, обмеження частоти звернень, приховані змінні, сесійні токени та інші механізми кіберзахисту. Це означає, що інструмент автоматизованого збору має уміти працювати з cookies, динамічними токенами автентифікації та механізмами обмеження.

Різнорідність даних є ще однією характерною ознакою логістичних портальних систем. У межах одного порталу можуть міститися дані різного типу: текстові, числові, геодані, графічні, потокові або інтерактивні компоненти. Дані часто генеруються внаслідок взаємодії декількох внутрішніх модулів і потребують нормалізації. Дослідження Munzert (2021) підкреслює, що обробка різнорідних вебданих є критично важливою при їх подальшому аналізі. Аналогічну позицію займає Glez-Peña (2020), акцентуючи увагу на необхідності структуризації інформації для підвищення її якості [6-7].

Ще однією загальною рисою логістичних порталів є висока частота оновлення інформації. Наприклад, дані про вантажі в Trans.eu змінюються щосекунди, а AIS-дані в MarineTraffic оновлюються щохвилини. Така

динамічність підвищує вимоги до продуктивності інструментів автоматизованого збору даних, оскільки вони повинні виконувати запити без значних затримок.

З урахуванням вищенаведених характеристик, логістичні портали можна розглядати як високодинамічні та структурно складні цифрові системи. Для автоматизованого збору даних з них необхідно враховувати специфіку їх архітектури, типи даних, обмеження доступу та вимоги до форматів обробки інформації. Алгоритми збору повинні об’єднувати такі елементи, як моделювання поведінки користувача, аналіз мережевих запитів, обхід антибот-захисту, робота з динамічним DOM та нормалізація різнорідних даних [8].

Також важливо зазначити, що логістичні портали формують багатоаспектне інформаційне середовище, яке є основою сучасних систем управління перевезеннями та логістичною аналітикою. Їхній змістовний та технічний аналіз створює необхідний фундамент для розроблення ефективного застосунку для автоматизованого збору та обробки логістичної інформації, що стане предметом подальших розділів роботи.

Однією з ключових особливостей логістичних цифрових платформ, що потребує окремого аналізу, є їхня орієнтація на комплексні бізнес-процеси та взаємодію великої кількості учасників ринку. На відміну від звичайних інформаційних сайтів, такі системи не обмежуються поданням даних, а виступають повноцінними робочими середовищами для укладання договорів, обміну повідомленнями та комерційними документами, ідентифікації та перевірки партнерів, відстеження виконання перевезень. Відповідно, внутрішня модель даних охоплює як відносно сталу, довідкову інформацію (класифікатори транспортних засобів, номенклатуру товарів, опис маршрутів), так і змінні параметри, які формуються у процесі щоденної діяльності користувачів.

У науковій літературі з цифрової логістики підкреслюється, що подібні платформи фактично виконують роль «цифрових ринків» (digital marketplaces), де дані стають основним ресурсом і водночас інструментом конкуренції. Rodrigue і Notteboom акцентують, що сучасні логістичні інформаційні системи

створюють багатовимірний простір даних, у межах якого співіснують операційні, стратегічні, комерційні та регуляторні складові. Частина цих відомостей представлена у відкритому інтерфейсі й доступна користувачу безпосередньо, інша ж зберігається у прихованих службових структурах і може передаватися лише через інтегровані модулі або спеціалізовані інтерфейси взаємодії [9].

Суттєвий вплив на організацію даних мають механізми персоналізації, що застосовуються на таких платформах. Інтерфейс, порядок відображення записів, набір доступних функцій та навіть зміст окремих полів залежать від типу облікового запису, історії попередніх дій, регіону діяльності, сегмента ринку чи статусу компанії. Унаслідок цього одна й та сама вебсторінка може містити відмінні елементи структури для різних користувачів. Дослідники вебінформаційних систем, зокрема Hogan, звертають увагу на те, що індивідуалізація вмісту ускладнює застосування універсальних алгоритмів web-скрапінгу і потребує адаптивних підходів до збору даних.

Ще одна важлива риса полягає у поєднанні в межах одного ресурсу кількох аналітичних та сервісних модулів, які виконують функції фільтрації, рейтингового оцінювання, виявлення підозрілих операцій або потенційних ризиків. Наприклад, системи Project44 та FourKites застосовують моделі машинного навчання для розрахунку прогнозованого часу прибуття вантажів, використовуючи історичні спостереження, дані про затримки, погодні умови та інформацію про завантаженість інфраструктури. У результаті частина відомостей, які бачить користувач, є вже інтерпретованими та агрегованими, а не «сирими» даними. Для розробника інструментів автоматизованого збору така інформація є особливо цінною, однак доступ до неї зазвичай жорстко регламентований і реалізується через внутрішню логіку платформи, а не через базові програмні інтерфейси.

Окремий вимір формування структури даних пов’язаний із використанням стандартизованих класифікаторів та міжнародних норм. Значна частина логістичних систем інтегрує у свої довідники коди Гармонізованої системи

опису та кодування товарів (HS-коди), стандарти маркування контейнерів ISO (зокрема ISO 6346), правила INCOTERMS, коди портів та локацій UN/LOCODE, а також класифікацію небезпечних вантажів за вимогами IMO. Упровадження таких стандартів забезпечує узгодженість даних між різними учасниками ланцюгів постачання, але водночас створює додаткові вимоги до систем автоматизованого аналізу, які мають коректно інтерпретувати ці позначення. У матеріалах UNCTAD підкреслюється, що стандартизація є одним із базових чинників цифрової сумісності в логістиці [10].

Часто логістичні платформи виступають проміжною ланкою між користувачем і зовнішніми інформаційними системами: державними реєстрами, митними службами, реєстрами ліцензій, сертифікаційними органами, страховими компаніями тощо. Наприклад, окремі транспортні біржі отримують дані про чинність ліцензій перевізників із національних реєстрів, а морські портали можуть бути пов’язані з митними системами через стандартизовані протоколи електронного обміну. Це означає, що фактичний інформаційний контент порталу формується не лише внутрішніми базами, а й численними зовнішніми джерелами, що створює складну мережу взаємопов’язаних потоків даних.

Важливим компонентом архітектури є подійно-орієнтований підхід до обробки інформації (event-driven architecture). Зміна статусу замовлення, поява нової заявки, коригування тарифу або оновлення місцеперебування транспортного засобу сприймаються системою як окремі події, що запускають відповідні реакції: перерахунок відображуваних списків, надсилання сповіщень, оновлення аналітичних панелей. Для програмних засобів автоматизованого збору це означає необхідність не лише періодичного опитування ресурсу, а й побудови логіки, здатної відслідковувати послідовності змін і працювати з потоковими даними.

Питання масштабованості набуває особливого значення у зв’язку з обсягами інформації, якими оперують такі системи. Портали морської логістики фіксують дані щодо великої кількості суден, їхніх маршрутів, портів заходу та

характеристик; наземні транспортні платформи відображають десятки тисяч актуальних пропозицій і запитів щодня. У дослідженнях з Big Data підкреслюється, що саме логістика є однією з галузей, де зростання обсягів даних найбільш відчутне і потребує спеціальних підходів до їх обробки й зберігання. Для систем web-скрапінгу це означає необхідність оптимізації частоти запитів, використання черг, кешування та засобів розподіленої обробки.

Суттєву роль відіграють і історичні дані, які накопичуються в межах логістичних платформ. Хоча користувач зазвичай працює з актуальними показниками, внутрішні модулі часто зберігають тривалі ряди спостережень щодо тарифів, часу доставки, завантаженості маршрутів, частоти затримок. Ці архіви використовуються для аналітики та побудови прогнозних моделей, однак повний доступ до них, як правило, відсутній. Для розробника застосунку автоматизованого збору це означає, що формована ним база даних базується на зрізах поточного стану, а відновлення повної історії вимагає тривалого моніторингу або спеціальних домовленостей із власниками платформи.

Не можна оминути й правові та етичні аспекти організації доступу до інформації. Логістичні системи працюють у правовому полі, яке визначається, зокрема, нормами щодо захисту персональних даних (наприклад, Загальним регламентом про захист даних – GDPR), комерційної таємниці, конкурентного права, а також умовами ліцензійних угод. Це позначається на технічних механізмах контролю доступу: використовуються системи авторизації, токени безпеки, обмеження за IP-адресами, багаторівневі ролі користувачів. У працях, присвячених етиці та правовим аспектам web-скрапінгу, наголошується, що ігнорування цих вимог може призвести до порушення умов користування сервісом та юридичної відповідальності [7].

Ще одна важлива характеристика – висока щільність взаємозв’язків між різними сутностями, які відображаються в системі. Маршрут, транспортний засіб, водій, вантаж, відправник, одержувач, експедитор, страхова компанія, митні органи – усі ці компоненти утворюють складну сітку зв’язків. Дані про маршрут без прив’язки до типу вантажу, часових обмежень або характеристик

транспортного засобу мають обмежену цінність; у свою чергу, інформація про перевізника без історії виконання зобов'язань чи рейтингових оцінок не дає повної картини його надійності. Тому інструмент автоматизованого збору повинен не лише вилучати окремі поля, а й забезпечувати правильне встановлення зв'язків між ними, формуючи логічно цілісні записи.

Узагальнюючи наведене, можна стверджувати, що сучасні логістичні вебплатформи – це складні багаторівневі системи, в яких технічні, організаційні, правові та аналітичні компоненти тісно переплетені. Для того щоб побудувати ефективну систему автоматизованого збору даних, необхідно не лише знати, які саме відомості містить портал, а й глибоко розуміти принципи побудови його внутрішніх структур, характер інтеграцій із зовнішніми джерелами, логіку персоналізації та правила доступу. Саме такий комплексний підхід дозволяє перетворити розрізнений вебконтент на систематизовану інформаційну базу, придатну для подальшого аналітичного опрацювання, що й буде реалізовано в наступних методичних і прикладних розділах дослідження.

1.2 Технології web-скрапінгу: методи, алгоритми та засоби реалізації

У контексті інтенсивної цифрової трансформації логістичної галузі web-скрапінг поступово перетворився на один із базових інструментів отримання даних із вебресурсів, які не надають зручних або відкритих API для інтеграції.

Постійне зростання масиву цифрової інформації, поява великої кількості спеціалізованих логістичних платформ та ускладнення їх інтерфейсів зумовлюють потребу в технологіях, здатних автоматизовано вилучати, структурувати й опрацьовувати дані в обсягах, достатніх для глибокого аналітичного опрацювання, моделювання та підтримки управлінських рішень.

Під web-скрапінгом у сучасній науково-практичній літературі зазвичай розуміють сукупність методів і алгоритмів цілеспрямованого збору вебданих, що охоплює широкий діапазон технічних підходів – від найпростіших сценаріїв парсингу статичних HTML-сторінок до використання браузерів, аналізу

мережевих запитів, обхідних механізмів антибот-захисту та залучення моделей машинного навчання на етапах очищення й класифікації інформації.

Як окрема галузь прикладної інформатики web-скрапінг розвивається синхронно зі зміною архітектурних підходів у веброзробці. У період домінування переважно статичних сайтів основний обсяг потрібних даних містився безпосередньо у HTML-коді сторінки. Для їх вилучення було достатньо прочитати розмітку документа, проаналізувати структуру DOM-дерева, знайти потрібні теги, атрибути та текстові вузли й перетворити їх у необхідний формат. Однак зі зміною парадигми на користь односторінкових застосунків (SPA) та масовим впровадженням клієнтського рендерингу, коли значна частина інформації завантажується й обробляється JavaScript після початкового завантаження сторінки, ситуація радикально змінилася. У дослідженнях Ryan Mitchell наголошується, що сьогодні більшість популярних сайтів повністю або частково працює на базі JavaScript-фреймворків, таких як React, Vue чи Angular, що істотно знижує ефективність суто статичних методів парсингу та потребує залучення складніших інструментів [11].

У сучасних класифікаціях прийнято виділяти принаймні три великі групи підходів до web-скрапінгу: статичний, динамічний та API-орієнтований. Статичний підхід використовується тоді, коли достатня частина цільових даних безпосередньо присутня в HTML-коді й не потребує виконання сценаріїв на стороні клієнта. Такий спосіб характерний для інформаційних сайтів, блогів, деяких каталогів чи довідникових систем. Динамічний скрапінг, навпаки, орієнтований на роботу з вебзастосунками, у яких вміст формується вже після завантаження сторінки внаслідок виконання JavaScript. У цьому випадку виникає потреба в інструментах, здатних імітувати роботу повноцінного браузера, виконувати скрипти, обробляти події та взаємодіяти з інтерфейсом.

Третій підхід – скрапінг через аналіз мережевих запитів, або API-скрапінг. Він базується на вивченні HTTP-запитів, які генерує браузер під час взаємодії з сайтом, із подальшим безпосереднім зверненням до виявлених даних, що повертають дані у форматах JSON чи XML.

Для логістичних порталів із їх характерною залежністю від асинхронних запитів, складної маршрутизації та активного використання JavaScript найбільш релевантними є саме динамічні та API-орієнтовані методи. У практиці розробки систем збору даних найчастіше застосовуються такі фреймворки, як Puppeteer, Playwright та Selenium, що надають можливість програмно контролювати браузер, відкривати сторінки, виконувати сценарії, заповнювати форми, ініціювати події кліку, прокручування, переходу між вкладками, а також знімати «готовий» DOM після завантаження всіх динамічних компонентів. У наукових описах їх розглядають як засоби автоматизації браузера (headless browser automation frameworks), які дозволяють отримати доступ до даних, недоступних при традиційному статичному аналізі коду сторінки.

Ключовою операцією у більшості сценаріїв web-скрапінгу є аналіз DOM, що передбачає виявлення закономірностей структури документа, визначення характерних шаблонів розмітки та побудову стійких шляхів до цільових елементів. У випадку логістичних вебплатформ структура документа часто є надзвичайно складною: численні вкладені блоки, таби, модальні вікна, інтерактивні таблиці, а також компоненти, які з’являються лише після певних дій користувача. Крім того, використання адаптивної верстки призводить до того, що DOM для мобільної та настільної версій сторінки може суттєво відрізнятися. Тому під час проєктування парсерів усе більше уваги приділяють евристичним методам пошуку елементів – за текстовими підписами, семантичними ознаками, відносним розташуванням у ієрархії, а не лише за жорстко заданими CSS-селекторами або XPath-виразами.

Окремий напрямок, тісно пов’язаний із динамічним скрапінгом, – перехоплення та аналіз мережевих запитів. У цьому випадку основним об’єктом дослідження виступає не розмітка сторінки, а HTTP-запити, які надсилаються до серверної частини застосунку під час роботи користувача з інтерфейсом. Значна частина сучасних логістичних платформ використовує REST-API або GraphQL-API для передавання структурованих даних у форматах JSON та XML. Виявлення цих endpoint’ів, визначення обов’язкових параметрів запитів, схем

автентифікації та обмежень частоти звернень дає змогу налаштувати прямий доступ до даних. Водночас такий підхід пов’язаний зі складністю реверс-інжинірингу API, оскільки розробники часто використовують токени, короткочасні сесії, шифрування окремих параметрів і додаткові рівні перевірки, які можуть змінюватися у процесі оновлення системи.

Суттєвим викликом для будь-якої системи web-скрапінгу є протидія з боку механізмів антибот-захисту. Логістичні платформи, як правило, захищаються від автоматизованого доступу шляхом застосування CAPTCHA, обмеженням кількості запитів з однієї IP-адреси за одиницю часу, виявленням підозрілих User-Agent, аналізом поведінкових патернів, перевіркою «відбитків» браузера (browser fingerprinting), використанням прихованих полів та спеціально вбудованих пасток (honeypots). Щоб знизити ризик блокування, розробники скрапінг-систем застосовують ротацію проксі-серверів, варіювання заголовків запитів, імітацію поведінки реальної людини (штучні паузи, послідовність дій, рухи курсора), а в окремих випадках – інтегрують сервіси розпізнавання CAPTCHA. В Behaviour-Driven Scraping, який передбачає попереднє вивчення поведінки реальних користувачів і відтворення подібних сценаріїв на рівні автоматизованих агентів [12].

Після етапу первинного вилучення даних обов’язковим кроком стає їх очищення та нормалізація. У випадку логістичних сайтів джерелами проблем виступають неоднорідні формати дат, різні одиниці вимірювання (наприклад, тоннаж у фунтах і кілограмах), нестандартні позначення валют, різні варіанти написання назв міст і компаній, а також поява дублікатів через незначні відмінності у структурі записів. Очищення даних включає видалення зайвих HTML-елементів або службових символів, виправлення форматів, уніфікацію одиниць вимірювання, видалення повторів та логічну перевірку значень. Семантична нормалізація передбачає зіставлення одних і тих самих сутностей, представлених різними варіантами написання, та приведення їх до єдиного еталонного вигляду. У працях Glez-Peña підкреслюється, що використання регулярних виразів, алгоритмів нечіткого зіставлення рядків та

напівавтоматичних підходів до нормалізації істотно підвищує придатність зібраних даних до подальшого аналізу.

Останніми роками у web-скрапінг активно інтегруються методи машинного навчання, що використовуються не лише для подальшої аналітики, а й безпосередньо на етапі збору даних. Зокрема, моделі класифікації можуть автоматично визначати типи елементів інтерфейсу, вирізняти релевантні блоки тексту, розпізнавати структуру таблиць або маркувати підозрілі записи. У логістичних задачах машинне навчання використовується для оцінювання пропускної здатності маршрутів, групування тарифів, пошуку аномальних значень (наприклад, надзвичайно низьких або високих ставок), а також для аналізу текстових описів вантажів, що подаються природною мовою. Інструменти обробки природної мови (NLP) дають змогу виокремлювати сутності (назви компаній, портів, країн, одиниць вимірювання), будувати тематичні моделі та покращувати структуризацію текстових полів.

Ще одним елементом сучасної інфраструктури web-скрапінгу є контейнеризація й оркестрація, оскільки процеси збору даних часто мають виконуватися паралельно, у великих масштабах та з високими вимогами до відмовостійкості. Використання Docker дозволяє стандартизувати середовище виконання парсерів, що спрощує розгортання та оновлення системи, а платформи оркестрації на кшталт Kubernetes дають змогу динамічно масштабувати кількість екземплярів скрапінг-сервісів, реагуючи на зміну навантаження. Для логістичних застосунків, де необхідно регулярно опрацьовувати десятки або сотні тисяч записів, подібні підходи стають практично обов’язковими.

Важливо враховувати і те, що логістичні портали зазвичай перебувають у стані постійного розвитку: змінюється дизайн, логіка побудови сторінок, назви CSS-класів, структура API. Тому статично запрограмовані парсери без механізмів адаптації швидко втрачають актуальність. Для підвищення стійкості до змін застосовують різні стратегії: регулярний моніторинг змін у DOM-структурі, автоматизовані тести, які сигналізують про некоректну роботу певних

модулів, та евристичні алгоритми, здатні «перебудовуватися» під нові шаблони сторінок. У низці досліджень пропонується використовувати моделі машинного навчання, які ідентифікують потрібні елементи не за фіксованими селекторами, а за контекстними ознаками, наприклад, за характерним поєднанням тексту, атрибутів та позиції в документі.

Не менш суттєвим завданням є організація сховища для результатів скрапінгу. Оскільки логістичні дані можуть мати як чітку табличну структуру (тарифи, характеристики транспорту, розклади), так і ієрархічну чи графову (ланцюги поставок, маршрути з проміжними точками, зв'язки між контрагентами), для їх зберігання використовуються різні типи СУБД. Реляційні бази даних (PostgreSQL) доцільні у випадку, коли структура добре формалізована й важлива підтримка складних транзакцій. Документоорієнтовані рішення (MongoDB) зручні для зберігання напівструктурованих JSON-документів. Для представлення мережевих зв'язків та аналізу структур ланцюгів постачання можуть застосовуватися графові бази (Neo4j), а для аналітичних задач із великими обсягами логів та історичних записів – колонкові сховища на кшталт ClickHouse. Для повнотекстового пошуку, побудови фільтрів та фасетної навігації нерідко додається окремий індексовий шар на Elasticsearch чи аналогічних системах.

З практичного погляду, побудову великих скрапінг-проєктів істотно спрощують спеціалізовані фреймворки. Одним з найбільш поширених є Scrapy – Python-орієнтоване середовище, яке надає розробникові готову інфраструктуру для організації павуків (spiders), конвеєрів обробки даних (pipelines), middleware-компонентів, системи логування та обробки винятків. Scrapy підтримує інтеграцію проксі-сервісів, механізми чергування запитів, кешування, а також дозволяє легко масштабувати проєкт за рахунок розподіленого виконання. У контексті логістичних задач такі можливості є критичними, оскільки забезпечують стабільність, контрольованість та прозорість процесу збору інформації.

Нарешті, важливо враховувати нормативно-правовий та етичний контекст, у якому здійснюється web-скрапінг. Логістичні портали працюють з даними, що можуть містити персональну інформацію, комерційні секрети або інші чутливі відомості. У європейському правовому полі на обробку таких даних суттєво впливає Загальний регламент про захист даних (GDPR), а також національні нормативні акти. У наукових публікаціях, присвячених етиці автоматизованого збору вебданих, наголошується на необхідності дотримання умов використання сайтів, поваги до механізмів саморегуляції (зокрема, файлів robots.txt), обмеження навантаження на сервери та уникнення несанкціонованого доступу до закритих частин системи. Відтак розроблення інструментів скрапінгу має поєднувати технічну ефективність із правовою коректністю.

Можна констатувати, що сучасні технології web-скрапінгу становлять багатокомпонентний комплекс, який включає засоби динамічного виконання JavaScript, аналізу DOM, реверс-інжинірингу API, подолання антибот-захисту, очищення й нормалізації даних, їх подальшого структурування, зберігання та аналітичного опрацювання. Для логістичних порталів із притаманною їм високою динамікою контенту, великими обсягами інформації та складною бізнес-логікою ці технології є не просто допоміжним інструментом, а однією з ключових передумов побудови ефективних систем моніторингу та аналізу ринку. Актуальні наукові дослідження підтверджують, що web-скрапінг у цьому контексті виступає важливим елементом цифрової трансформації логістики, відкриваючи можливості для створення інтегрованих рішень, спрямованих на підвищення прозорості, прогнозованості та керованості ланцюгів постачання.

Окремий важливий вектор розвитку технологій web-скрапінгу, який має особливу вагу саме для логістичної галузі, пов’язаний із поєднанням інструментів геоаналітики та обробки просторових даних. Сучасні логістичні платформи дедалі ширше використовують картографічні модулі, інтерактивні маршрути, геошари та інші візуальні засоби, що доповнюють інформацію координатами, лінійними об’єктами та даними про вузли перевалки. Для коректного вилучення такого типу інформації потрібні програмні засоби, здатні

працювати з картографічними бібліотеками на зразок Leaflet, Mapbox, OpenLayers, відтворювати механізм завантаження тайлів, інтерпретувати координатні сітки та відстежувати оновлення геоданих, які надходять через WebSocket або інші потокові протоколи. У межах систем web-скрапінгу це означає, що завдання розширюється від простої роботи з текстом і таблицями до відтворення логіки підключення до геопотоків, що суттєво ускладнює проєктування та реалізацію таких рішень.

Ще одна показова тенденція – перехід частини вебплатформ, у тому числі логістичних, до формату прогресивних вебзастосунків (PWA), які поєднують риси класичного вебсайту та мобільного застосунку. Для систем автоматизованого збору даних це створює додатковий рівень складності, оскільки PWA активно застосовують Service Workers, внутрішні механізми кешування, офлайн-режим і перехоплення мережевого трафіку на стороні клієнта. У результаті інструменти web-скрапінгу змушені або моделювати роботу Service Worker, або обходити його вплив, інакше є ризик отримання неповних, несвоєчасних або застарілих даних. У сучасних технічних описах подібні системи характеризуються як багатошарова взаємодія браузера, клієнтських API і серверної інфраструктури, що вимагає спеціалізованих методів витягання інформації.

Один із факторів , що становить проблему обробка даних, які надходять у режимі наближеного до реального часу. Значна частина логістичних сервісів використовує WebSocket або інші механізми потокової передавання (наприклад, Server-Sent Events) для трансляції зміни статусів транспортних засобів, оперативного оновлення тарифів чи параметрів руху вантажів. У таких умовах класичний підхід, що базується лише на періодичному зчитуванні DOM, виявляється недостатнім, оскільки інформація може змінюватися без оновлення сторінки. Це зумовлює потребу у створенні скрапінг-компонентів, здатних підтримувати тривалі з’єднання з потоковими джерелами, накопичувати та обробляти дані у режимі постійного надходження. З технічної точки зору це вимушено веде до застосування асинхронних підходів – зокрема, бібліотек на

кшталт asyncio у Python чи подійного циклу Node.js, які дозволяють обробляти значні обсяги подій без втрати продуктивності [13].

На розвиток систем web-скрапінгу впливає і поширення графових API, що дедалі частіше застосовуються у складних вебплатформах, включно з логістичними. Використання GraphQL дає змогу клієнту формувати гнучкі запити й отримувати лише ті поля, які необхідні в конкретний момент. Для розробників скрапінг-рішень це створює як додаткові можливості, так і низку труднощів. З одного боку, GraphQL забезпечує компактність і цілеспрямованість відповіді, зменшує надлишковість даних і спрощує їх подальшу обробку. З іншого – динамічність схеми, можливість її часткового приховування та зміни конфігурації API без публічного повідомлення ускладнюють реверс-інжиніринг.

У таких випадках виникає потреба у спеціальних модулях, здатних досліджувати структуру графових схем, автоматично формувати запити та коригувати їх у відповідь на зміни на стороні сервера.

В умовах автоматизованого збору логістичних даних особливої ваги набувають стратегії оптимізації запитів та обмеження навантаження на серверну інфраструктуру. Багато порталів реалізують політику rate-limiting, встановлюючи верхні межі частоти звернень для окремих користувачів або IP-адрес. Це спонукає розробників систем web-скрапінгу використовувати алгоритми планування запитів і збалансованого розподілу навантаження: впроваджувати черги завдань, здійснювати кешування проміжних результатів, налаштовувати адаптивні інтервали оновлення, коли повторно запитуються лише змінені сегменти даних. У великомасштабних рішеннях часто застосовуються розподілені системи черг на кшталт RabbitMQ чи Apache Kafka, що дає змогу координувати роботу численних скрапінг-процесів і грамотно організовувати потоки інформації.

Ще однією важливою складовою є організація механізмів контролю помилок і забезпечення стійкості до збоїв. Деякі сторінки логістичних систем можуть працювати нестабільно через навантаження, часті оновлення інтерфейсу або тимчасові проблеми з мережею. Тому добре спроєктована скрапінг-система

повинна включати модулі повторної спроби запиту, фіксацію тайм-аутів, детальне логування нетипових ситуацій, гнучкі сценарії обходу проблемних ділянок, а також можливість автоматичного перемикання на альтернативні джерела інформації, якщо це передбачено архітектурою проєкту. Усе це потребує ретельної інженерної підготовки та впровадження принципів високої відмовостійкості, особливо коли результати збору безпосередньо впливають на ухвалення управлінських рішень.

З аналітичного погляду сучасні web-скрапінг-рішення дедалі тісніше інтегруються з системами бізнес-аналітики та модульними платформами машинного навчання. Масові масиви структуруваних та неструктурованих даних, отриманих зі скрапінгу, стають основою для побудови моделей прогнозування логістичних витрат, оцінювання сезонних коливань попиту, аналізу стабільності маршрутів та моделювання завантаженості транспортної інфраструктури. У практиці логістичних компаній подібні моделі все частіше використовуються для сценарного планування, а автоматизований збір інформації фактично виступає першим етапом формування набора «великих даних», без яких робота складних аналітичних інструментів є неможливою.

Окремо варто відзначити зростаюче значення уніфікації форматів даних. Логістичні платформи оперують як загальноприйнятими форматами (EDIFACT, EDI XML, JSON-LD), так і власними специфікаціями API. Для інтеграції таких масивів у єдине аналітичне середовище необхідно забезпечити їх приведення до погоджених структур, збереження метаданих, коректне маркування часових штампів, географічних координат і додаткових атрибутів. Частина цих завдань реалізується вже на етапі web-скрапінгу, інша – у межах наступних рівнів обробки та завантаження в сховища даних.

У сучасних інструментаріях дедалі важливіше місце посідають системи моніторингу та візуалізації технічних показників. У великих скрапінг-комплексах активно використовуються такі платформи, як Prometheus, Grafana, Kibana, що дозволяють у реальному часі відстежувати швидкість опрацювання запитів, частку успішних і помилкових звернень, середній час відповіді, частоту

тайм-аутів, а також загальну стабільність роботи окремих модулів. Наявність таких інструментів забезпечує прозорість процесу збору даних та дає змогу своєчасно реагувати на зміни в доступності або структурі логістичних порталів.

Підсумовуючи, можна констатувати, що еволюція технологій web-скрапінгу веде від відносно простих скриптів для копіювання статичного тексту до складних, багаторівневих систем, здатних працювати з динамічними інтерфейсами, потоковими даними, гнучкими API та значними обсягами інформації. У контексті логістики такі рішення мають особливу вагу, оскільки логістичні платформи належать до найбільш інформаційно насичених і технологічно складних вебсистем. Відповідно, ефективний web-скрапінг у цій сфері передбачає не лише володіння окремими інструментами, а й розуміння архітектурних принципів вебзастосунків, методів організації даних та логіки функціонування логістичних інформаційних систем.

1.3 Аналіз існуючих рішень для автоматичного збору логістичних даних

Сучасний ринок програмних продуктів і технологій, орієнтованих на автоматизований збір логістичних даних, характеризується значним різноманіттям підходів, інструментів і платформ. Вони охоплюють як комерційні сервіси глобального масштабу, так і відкриті технологічні рішення, що використовуються розробниками для створення власних систем web-скрапінгу. Логістична галузь, зокрема її цифровий сегмент, активно розвивається у напрямі автоматизації, що зумовлює появу нових інструментів, орієнтованих на отримання структурованих даних у режимі реального часу, обробку великих обсягів інформації та інтеграцію з аналітичними платформами. Аналіз сучасних рішень дає змогу окреслити їх потенціал, обмеження та сфери застосування, що є важливою передумовою для обґрунтування власного підходу до розроблення інструменту автоматичного збору логістичних даних.

Однією з найвідоміших і найпотужніших платформ, що пропонує інструменти для автоматизованого збору та аналізу інформації, є Project44, яку у звітах Gartner називають лідером у сегменті систем видимості ланцюгів постачання. Платформа спеціалізується на отриманні даних у режимі реального часу про переміщення вантажів, координати транспортних засобів, статуси постачання, можливі затримки та передбачуваний час прибуття. Основним методом взаємодії з даними у Project44 є прямий доступ через API, що забезпечує високий рівень точності й актуальності інформації. Однак можливість автоматичного збору таких даних через web-скрапінг обмежена, оскільки більшість функцій сервісу знаходяться «за прапором» закритої авторизації та доступні виключно за комерційними угодами. Тому Project44 є прикладом рішення, яке пропонує якісну інформацію, але за моделлю контрольованої взаємодії, яка не дозволяє використовувати стандартні механізми парсингу для отримання критично важливих даних.

Іншим ключовим гравцем у сфері цифрової логістики є платформа FourKites, яка так само орієнтована на комплексну видимість транспортних операцій. Завдяки співпраці з тисячами перевізників, компанія забезпечує обмін інформацією через стандартизовані канали, включаючи EDI-повідомлення, телематичні шлюзи та API-інтеграцію. На відміну від Project44, FourKites частково застосовує веб інтерфейс із динамічними компонентами, що відкриває можливості для обмеженого web-скрапінгу. Проте ключові дані, як і в Project44, залишаються захищеними, що обмежує потенціал використання парсингу як основного інструменту вилучення інформації. У дослідженнях щодо транспортної видимості FourKites посідає позицію технологічного лідера, але рівень відкритості даних є невисоким, а робота з його API вимагає дотримання суворих правил доступу.

Серед міжнародних логістичних платформ, що надають частково або повністю відкриту інформацію, значний інтерес становлять морські сервіси MarineTraffic, VesselFinder, SeaRates, FleetMon. MarineTraffic активно використовується в наукових дослідженнях транспортних потоків, зокрема у

роботах Yan та співавт. (2020), де аналізуються моделі руху суден та глобальні маршрути. На відміну від закритих систем, ці платформи дозволяють отримувати частину інформації як через API, так і через динамічний вебінтерфейс, який формується на основі JavaScript-рендерингу. Унаслідок цього web-скрапінг морських порталів є технічно складним, але можливим. MarineTraffic, наприклад, часто застосовує Canvas-рендеринг, що ускладнює вилучення структури карти, але дозволяє скрапити текстові таблиці та статуси суден. SeaRates надає значно більше можливостей для інспекції мережевих запитів, оскільки багато даних завантажуються у форматі JSON. У технічній документації SeaRates зазначено, що дані тарифів, маршрутів і часу транзиту формуються через зовнішні API судноплавних ліній, що робить інженерію web-скрапінгу водночас перспективною і складною [14].

Не менш цікавими є рішення, що стосуються наземної логістики. До найбільш використовуваних платформ цього сегмента належать Trans.eu, Timocom, CargoHub, Cargopedia. Trans.eu є одним із провідних рішень у Європі, і в наукових дослідженнях відзначається як ключовий компонент цифрової екосистеми транспортної логістики регіону. Однак вебінтерфейс платформи значною мірою базується на асинхронних запитах та складних SPA-механізмах, що вимагає застосування таких інструментів, як Playwright або Puppeteer. Timocom, який позиціонує себе як «Smart Logistics System», навпаки, містить значну кількість механізмів захисту – зокрема CAPTCHA, адаптивні тайминги, виявлення ботів – що робить стандартний web-скрапінг практично неможливим.

Одним із небагатьох сегментів, які можуть бути доступні для автоматичного збору інформації, є відкриті статистичні дані або несуттєві елементи інтерфейсу, які не містять комерційної інформації.

Серед інструментів загального призначення, які часто використовуються для web-скрапінгу логістичних порталів, варто виділити Puppeteer, Playwright та Selenium. Puppeteer, розроблений Google, дає змогу керувати браузером Chrome у headless-режимі та виконувати JavaScript-механізми сторінки, що є необхідним для парсингу сучасних логістичних SPA-застосунків. Playwright, створений

Microsoft, має ще більше можливостей: він дозволяє працювати з кількома браузерами (Chromium, Firefox, WebKit), підтримує паралелізацію задач і працює з відкритою мережею запитів, що є ключовим для аналізу API логістичних платформ. У документованих кейсах, представлених у роботах Mitchell (2023) та Munzert (2021), ці фреймворки називаються серед найефективніших інструментів для збору динамічних даних.

Якщо розглядати рішення, орієнтовані на великі обсяги даних і промислове використання, то ключову роль відіграють Scrapy, Apify та Octoparse. Scrapy є одним із найпопулярніших Python-фреймворків, що забезпечує модульність, можливість побудови розподілених систем і підтримку pipeline-обробки даних. Apify, який позиціонується як платформа для автоматизації вебпроцесів, пропонує готові скрапінг-актори, зокрема для платформ морської логістики та транспортних бірж. Octoparse є інструментом для користувачів без глибоких технічних знань, однак має обмеження у швидкості та гнучкості, тому рідше використовується у системах промислового рівня. Аналітичні огляди ресурсів Towards Data Science та KDnuggets підтверджують, що Scrapy та Playwright є найефективнішими при роботі з динамічними логістичними платформами.

Вагоме місце займають рішення, що поєднують функції web-скрапінгу та подальшої аналітики, зокрема Keboola, Talend, RapidMiner. Вони надають розширені засоби інтеграції, ETL-обробки та візуалізації даних, що дозволяє створювати комплексні ланцюги опрацювання логістичної інформації. У контексті автоматичного збору інформації такі рішення відіграють роль інструментів високого рівня, які використовують скрапінг як первинний етап для побудови масштабованих систем аналітики.

У межах відкритих ініціатив варто відзначити OpenStreetMap, який надає структуровані геодані для транспортних аналізів і використовується у численних дослідженнях логістичних мереж. Хоча OSM не є логістичною платформою у прямому сенсі, його дані часто інтегруються у web-скрапінгові рішення для

зіставлення координат, побудови маршрутів або нормалізації геопросторових даних.

Загалом аналіз наявних технологічних і платформних рішень дає змогу виокремити три великі групи засобів автоматичного збору логістичних даних. До першої належать комерційні платформи видимості ланцюгів постачання (Project44, FourKites), які працюють переважно через закриті API. Друга група охоплює логістичні вебплатформи з частково відкритими динамічними інтерфейсами (MarineTraffic, SeaRates, Trans.eu, Cargopedia), що можуть бути об’єктами web-скрапінгу за умови використання headless-браузерів і аналізу мережевих запитів. Третю групу становлять універсальні інструменти web-скрапінгу (Puppeteer, Playwright, Scrapy, Apify), які забезпечують технічну основу для створення власних модульних рішень і дозволяють адаптуватися до особливостей конкретної логістичної системи.

В підсумку як бачимо – сучасні рішення для автоматичного збору логістичних даних формують багаторівневу екосистему, де поєднуються закриті високотехнологічні платформи, відкриті інформаційні сервіси, потужні інструменти для динамічного парсингу та модульні системи для промислової обробки великих обсягів інформації. Хоча більшість комерційних систем не передбачають можливості вільного web-скрапінгу, багато логістичних порталів із динамічними інтерфейсами дозволяють отримувати значний обсяг даних при застосуванні сучасних технологій headless-парсингу та аналізу API. Це обумовлює актуальність створення власного інструменту, що поєднує можливості різних підходів і забезпечує гнучкість, продуктивність та адаптивність у змінних умовах логістичного середовища.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБЛЕННЯ ЗАСТОСУНКУ ДЛЯ ПАРСИНГУ ЛОГІСТИЧНИХ ПОРТАЛІВ

2.1 Обґрунтування вибору архітектури та технологічного стеку застосунку

Проєктування застосунку для автоматизованого вилучення логістичних даних потребує ретельного добору технологічних інструментів, здатних ефективно працювати в умовах динамічного вебсередовища та складної структури сучасних логістичних порталів. Такі портали мають низку технічних особливостей – вони використовують клієнтську генерацію контенту, велику кількість асинхронних запитів, внутрішні API, інтерактивні модулі та механізми захисту від автоматизованих систем. Тому технологічний стек має забезпечувати одночасно високу продуктивність, стійкість до змін вебінтерфейсу, можливість обробки численних запитів і гнучкість у роботі з напівструктурованими даними. З огляду на ці вимоги ядром серверної частини було обрано Node.js, оскільки ця платформа створена для обробки великої кількості подій у неблокувальному режимі та підтримує масштабовані асинхронні сценарії, що є критично важливим для систем web-скрапінгу. У дослідженнях S. Tilkov (2021) та O. Gousios (2020) наголошується, що Node.js демонструє високу пропускну здатність за умов інтенсивної взаємодії із зовнішніми сервісами й оптимально підходить для застосунків, орієнтованих на обробку потоків мережевих запитів у реальному часі [15-16]. Для логістичних порталів, де дані змінюються щосекунди, а значна частина інформації оновлюється у режимі near-real-time, така властивість Node.js стає вирішальною.

Суттєвою перевагою Node.js є його інтеграція з екосистемою NPM, яка надає широкий спектр модулів для роботи з HTTP, WebSocket, автентифікацією, шифруванням, проксі та розподілом навантаження. Така гнучкість дозволяє будувати складні системи скрапінгу з мінімальними витратами на розроблення допоміжних інструментів. Крім того, архітектура Node.js співвідноситься з

вимогою масштабування: через механізми PM2 або кластеризації сервер може розподіляти навантаження між кількома процесами, підтримуючи стабільну роботу застосунку навіть під час обробки значного обсягу паралельних завдань.

Для забезпечення надійного доступу до контенту, який генерується в браузері після виконання JavaScript, було обрано Playwright. На сучасних логістичних порталах переважають односторінкові застосунки, створені на базі React, Vue або Angular, де ключові дані завантажуються через динамічні API, а HTML формується безпосередньо у браузері. У таких умовах традиційні методи скрапінгу не показують потрібної ефективності. Playwright, згідно з даними Web Testing Initiative (2023), демонструє стабільність і точність у взаємодії з динамічними інтерфейсами, завдяки чому надає повний контроль над браузерним середовищем, включаючи перехоплення мережевих запитів, виконання скриптів, емуляцію користувацьких дій та роботу з WebSocket-каналами. Саме можливість аналізувати та дублювати внутрішні мережеві запити логістичних порталів має вирішальне значення для створення науково коректного інструменту парсингу, оскільки більшість порталів не відображає сирі дані у DOM, а передає їх у JSON через власні API.

Другим ключовим елементом технологічного стеку є MongoDB. Оскільки дані, що надходять із логістичних систем, не є однорідними та часто містять вкладені JSON-структури, географічні координати, параметри маршруту або записи про статус вантажів, необхідна база даних, здатна зберігати мінливі формати без суворих обмежень щодо схеми. MongoDB, за даними рейтингу DB-Engines (2024), є найбільш популярною NoSQL технологією у застосунках, що працюють із великими масивами змінних даних [17]. Вона надає можливість зберігати API-відповіді у природному для них форматі BSON, не потребуючи складних перетворень. Додатковою перевагою є підтримка геопросторових індексів, необхідних для аналізу портових координат, маршрутних ліній та відстеження переміщення контейнерів, що є важливими складовими логістичної аналітики. Завдяки цьому MongoDB оптимально відповідає вимогам до збереження та подальшої обробки змістових даних з логістичних платформ.

Додатковою умовою створення масштабованого та відмовостійкого рішення є використання контейнеризації, яку забезпечує Docker. Системи web-скрапінгу мають бути стійкими до змін конфігурацій, версій браузера або залежностей, а контейнеризація дозволяє створювати ізольоване середовище, яке відтворюється на будь-якій платформі без втрати функціональності. За статистикою Cloud Native Computing Foundation (2023), Docker є стандартом у галузі рішень для роботи з великими даними та інтенсивними обчислювальними навантаженнями [18]. Контейнеризація дає змогу розгортати окремі інстанси Playwright, що дозволяє паралельно обробляти десятки чи сотні web-скрапінг-сесій, а також масштабувати MongoDB та серверну частину Node.js залежно від зростання потреб користувачів. Завдяки Docker забезпечується і стабільність оновлення: зміни у конфігурації порталу або в інструментах браузера не порушують роботу основного застосунку, оскільки виконання відбувається в контрольованому середовищі.

Інтерфейс користувача також відіграє вагому роль у загальній архітектурі, адже кінцевим результатом роботи системи є подання структурованих логістичних даних у зручній формі. Для цього було обрано Bootstrap 5, який дозволяє створити адаптивну, легко масштабовану адміністративну панель з використанням готових UI-компонентів. Bootstrap забезпечує швидке формування таблиць, панелей фільтрації, графічних блоків та інтерактивних модальних елементів, що робить його оптимальним середовищем для побудови інформаційно-аналітичних інструментів. Екосистема Bootstrap також дозволяє додавати компоненти DataTables, Chart.js або Leaflet для візуалізації тарифів, маршрутів, історичних трендів або геопросторових даних, що значно розширює аналітичні можливості застосунку.

Загальна архітектура системи базується на модульному підході: перший рівень відповідає за вилучення інформації за допомогою Playwright-агентів, які взаємодіють безпосередньо з вебпорталами; другий рівень реалізує попередню обробку даних на Node.js, нормалізацію, очищення, узгодження форматів і підготовку документів до збереження; третій рівень забезпечує стабільне та

оптимізоване зберігання даних у MongoDB; четвертий – представляє результати у вигляді інтерактивного інтерфейсу на Bootstrap. Така побудова забезпечує розділення відповідальності, підвищує гнучкість системи та спрощує розширення її функціональних можливостей у майбутньому.

Поєднання цих технологій створює інфраструктуру, здатну працювати із логістичними порталами будь-якої складності, обробляти великий обсяг динамічних даних, адаптуватися до оновлень інтерфейсу, забезпечувати відмовостійкість та надавати користувачеві зручний доступ до отриманої інформації. Обрана архітектура відповідає сучасним науковим підходам до побудови високопродуктивних систем web-скрапінгу і дозволяє сформувати універсальний фундамент для створення гнучкого та масштабованого застосунку, який може бути інтегрований у логістичні процеси різного рівня складності.

2.2 Реалізація програмних модулів застосунку

Реалізація застосунку для парсингу логістичних порталів базується на модульному підході, за якого кожен компонент відповідає за окремий етап обробки даних: їх вилучення з вебінтерфейсу, нормалізацію та трансформацію, збереження у сховищі й подальшу візуалізацію у вигляді зручного користувацького інтерфейсу. Обраний технологічний стек Node.js + Playwright + MongoDB + Docker + Bootstrap зумовлює логіку побудови програмних модулів, структуру взаємодії між ними та розподіл відповідальності в системі. На практичному рівні це означає, що застосунок реалізовано як набір сервісів, які виконують окремі функції, але працюють узгоджено через чітко визначені інтерфейси й формати даних.

Архітектурно система складається з модуля збору даних (скрапінг-агенти на базі Playwright), модуля обробки та нормалізації, модуля збереження інформації у MongoDB та вебінтерфейсу, побудованого на Node.js (Express) та HTML-шаблонах із використанням Bootstrap. Такий поділ дає змогу незалежно

розвивати кожен компонент, тестувати й оновлювати його без необхідності змінювати інші частини системи. Крім того, це спрощує масштабування: за потреби можна збільшити кількість екземплярів скрапінг-агентів або виділити окремий сервер для бази даних.

2.2.1. Модуль збору даних з логістичних порталів

Модуль збору даних є стартовою точкою всього конвеєра обробки інформації, оскільки саме він забезпечує доступ до динамічного контенту логістичних порталів [19]. Його ядром виступає Playwright, який дозволяє запускати браузер у headless-режимі, завантажувати вебсторінки, виконувати JavaScript-код, взаємодіяти з елементами інтерфейсу та перехоплювати мережеві запити.

У коді застосунку модуль збору даних реалізовано як окремий сервіс, який запускається під керуванням Node.js та взаємодіє з іншими компонентами через асинхронні функції а саме три основні завдання:

– ініціалізація браузерного контексту (persistent context), що забезпечує стабільність сесії при переході між сторінками;

– парсинг сторінок із пагінацією, включно з очікуванням завантаження динамічних компонентів та збором інформації з кожної заявки;

– передача результатів у MongoDB, де дані проходять попередню валідацію (перевірка унікальності requestId) та отримують додаткові метадані (часові мітки, агреговані значення тощо).

Нижче наведено фрагмент робочого коду в лістингу 2.1 модуль, який використовується для автоматизованого збору заявок із логістичного порталу. Він забезпечує ініціалізацію браузерного середовища, авторизацію користувача (за потреби), навігацію по сторінках порталу та вилучення структурованої інформації зі сторінок оголошень.

Лістинг 2.1 – Модуль збору даних на основі Playwright

```
async function processUrl(context, url) {
  const page = await context.newPage();
  await page.goto(url, { waitUntil: 'domcontentloaded' });
  while (true) {
  console.log("Відкрив:", page.url());
```

```
  await page.waitForSelector('.request_card', { timeout: 10000 });
  const items = await page.$$eval('.request_card', cards =>
  cards.map(card => { const clean = (txt) => txt ? txt.replace(/\s+/g, '
').trim() : "";
  const requestId = clean(card.getAttribute("data-request_id") || "");
  const route = clean(card.querySelector('.request_route
.request_distance')?.innerText || "");
  const distance = clean(card.querySelector('.request_route
.distance')?.innerText || "");
 const cargo = clean(card.querySelector('.cargo_type')?.innerText || "");
 const weight = clean(card.querySelector('.weight')?.innerText || "");
 const cube = clean(card.querySelector('.cube')?.innerText || "");
 const priceRaw = clean(card.querySelector('.price_main')?.innerText"");
 const pricePerKmRaw =
clean(card.querySelector('.price_additional')?.innerText || "");
        return {
          requestId, route, distance, cargo, weight,
          cube,
          price: priceRaw.replace(/\s+/g, ' '),
          pricePerKm: pricePerKmRaw.replace(/\s+/g, ' ')
        };}));
```

кінець лістингу 2.1

І на виході маємо такій результат це показано в рисунку 2.1 якій робить перевірку на дублювання і виявлення нових замовлень.

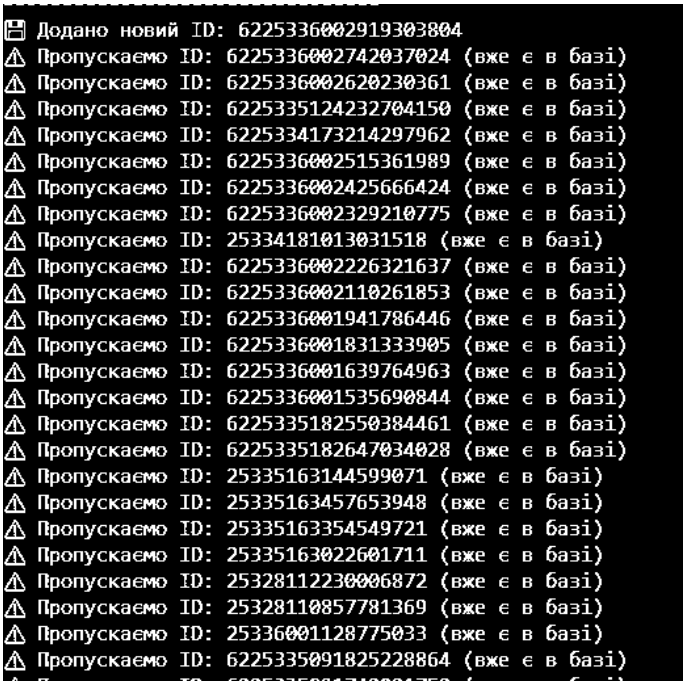

Рисунок 2.1 – Отриманий результат під час парсингу

Джерело: розроблено автором

У процесі проєктування системи автоматизованого збору логістичних даних одним із ключових етапів є створення браузерного контексту, у межах якого виконуються всі операції з динамічними вебпорталами. Оскільки логістичні платформи широко застосовують JavaScript-рендеринг, асинхронні API-виклики та механізми антибот-захисту, використання простих HTTP-клієнтів (таких як Axios або Fetch API) стає недостатнім. Саме тому у межах системи застосовано інструмент Playwright, який забезпечує повний контроль над браузерним середовищем і дозволяє емулювати поведінку реального користувача.

Ініціалізація браузерного контексту є фундаментальною частиною архітектури, оскільки від неї залежить стабільність роботи парсера, можливість проходження автентифікації, обробка cookies, збереження сесій користувача, а також коректне виконання скриптів, які генерують елементи DOM. Для підвищення стійкості системи та повторюваності сценаріїв було застосовано формат persistent context, що дозволяє зберігати дані профілю та повторно використовувати їх між сесіями. Це зменшує ризик блокування, пришвидшує авторизацію та забезпечує більш природну поведінку автоматизованого агента.

Лістинг 2.2 демонструє базовий фрагмент коду, який демонструє створення браузера Playwright з фіксованим користувацьким профілем та налаштуванням параметрів відображення. Саме цей модуль є основою для подальшого виконання дій зі сторінками логістичних порталів, включаючи навігацію, обробку подій, виконання JavaScript та зчитування сформованого DOM.

Лістинг 2.2 – Створення браузерного контексту

```
async function createBrowser() {
  return await chromium.launchPersistentContext(USER_DATA_DIR, {
    headless: false,
    viewport: { width: 1280, height: 800 }
async function getUrls() {
  return [TARGET_URL];
}
```

кінець лістингу 2.2

У процесі побудови високонавантажених систем автоматизованого збору логістичних даних важливим компонентом є створення стабільного браузерного середовища, яке здатне коректно відтворювати поведінку реального користувача. Базовий фрагмент конфігурації, заснований на офіційних рекомендаціях інструменту Playwright, який підтримує роботу з динамічними вебінтерфейсами та складними SPA-застосунками.

Застосування функції launchPersistentContext() забезпечує створення персистентного браузерного профілю, що дозволяє зберігати cookies, токени автентифікації та інші дані сесії, які є критично важливими під час взаємодії з логістичними порталами, що використовують механізми антибот-захисту. Саме тому підхід базується на офіційній технології, докладно описаній у матеріалах Playwright.

2.2.2 Модуль збереження інформації (MongoDB Storage Module)

Ефективне збереження результатів парсингу є критичною умовою подальшого аналітичного опрацювання логістичних даних, оскільки саме від структури та організації сховища залежить можливість виконання запитів, побудови звітів, візуалізації та інтеграції з зовнішніми інформаційними системами. Для розробленого застосунку було обрано документоорієнтовану базу даних MongoDB, яка належить до класу NoSQL систем і підтримує гнучкі схеми зберігання даних у форматі BSON (Binary JSON). На відміну від реляційних СУБД, що потребують жорстко фіксованих структур таблиць, MongoDB дає змогу зберігати об’єкти зі змінним набором полів, що є особливо важливим у контексті логістичних порталів, де формат відповіді API може відрізнятися залежно від типу запиту чи версії інтерфейсу [20].

Вибір MongoDB зумовлений кількома чинниками. По-перше, модель документів природно відповідає структурі даних, які надходять від логістичних платформ у форматі JSON: одна заявка (request) або перевезення (shipment) може бути представлена як окремий документ із вкладеними полями, масивами та геопросторовими координатами. По-друге, MongoDB підтримує вбудовані механізми індексування, у тому числі індекси за полями, комбіновані індекси,

унікальні обмеження та геоіндекси, що дає змогу оптимізувати доступ до записів за ключовими атрибутами (наприклад, requestId, напрямок маршруту, дата створення тощо). По-третє, система орієнтована на горизонтальне масштабування через шардінг і реплікацію, що є актуальним для сценаріїв, де обсяги логістичних даних можуть зростати експоненціально.

У рамках розробленого застосунку логічна модель сховища передбачає виділення окремої колекції requests, у якій зберігаються документи, що відповідають вантажним заявкам, отриманим з логістичного порталу. Дана колекція є центральним елементом системи зберігання, оскільки саме вона акумулює структуровані дані, що надходять на подальші етапи обробки, нормалізації та аналітики.

Кожен документ у колекції requests містить набір основних полів, які забезпечують повне представлення ключових атрибутів заявки. До таких полів належать:

- requestId – унікальний ідентифікатор заявки на логістичному порталі;
- route – текстовий опис маршруту (напрямок «з-до»);
- distance – оцінка відстані;
- cargo – опис типу вантажу;
- cube – об’єм або кубатура;
- price – загальна ставка;
- pricePerKm – ставка за кілометр;
- timestamp – час додавання документа до локального сховища.

Така структура дає змогу поєднувати у межах одного документу як сирі текстові дані, так і числові показники, які надалі можуть бути перетворені, нормалізовані й використані для розрахунку узагальнених показників. З огляду на те, що основним ключем сутності на стороні логістичного порталу є requestId, у колекції requests формують унікальний індекс за цим полем. Це забезпечує неможливість збереження дубліката однієї й тієї ж заявки при повторному запуску скрапінгу, що особливо важливо за умов, коли система періодично оновлює дані з порталу.

Практична реалізація модуля збереження ґрунтується на використанні офіційного драйвера MongoDB Node.js Driver, який згідно з офіційною документацією підтримує асинхронну модель роботи з операціями insertOne, insertMany, findOne, updateOne тощо.

Підключення до бази даних виконується один раз при запуску застосунку, після чого об’єкти db та collection передаються до модулів збору та обробки даних.

Створення індексу здійснюється за допомогою методу createIndex, який задає параметр unique: true для гарантування унікальності значень поля requestId.

Нижче наведено лістинг 2.3, який реалізує ініціалізацію бази даних, підключення до колекції requests та створення унікального індексу. Цей код спирається на підходи, рекомендовані в офіційній документації MongoDB і адаптований до архітектури розробленого застосунку [21].

Лістинг 2.3 – Ініціалізація MongoDB та налаштування

```
const MONGO_URL = "mongodb://localhost "; // змінити на свою адресу, якщо
Atlas
const DB_NAME = "della";
const COLLECTION = "requests";
let db, collection;

async function connectDB() {
  const client = new MongoClient(MONGO_URL);
  await client.connect();
  db = client.db(DB_NAME);
  collection = db.collection(COLLECTION);

  // Унікальний індекс за requestId
  await collection.createIndex({ requestId: 1 }, { unique: true });
  console.log(" Підключено до MongoDB");
```

кінець лістингу 2.3

У подальшому модуль збереження інформації використовує функцію connectDB() як точку входу до роботи зі сховищем. Після успішної ініціалізації колекція requests стає доступною для запису документів, які надходять із модуля збору даних. Зокрема, у функції збереження (розглянутій у попередньому

підрозділі) перед вставкою кожного документа до бази здійснюється додаткова перевірка на існування запису з таким самим requestId. Хоча унікальний індекс сам по собі гарантує відхилення дублюючих записів, програмна перевірка дозволяє уникнути зайвих помилок та формувати більш чистий журнал роботи застосунку.

Ще однією перевагою обраної моделі є можливість розширення схеми документів без зміни структури всієї бази. У разі, якщо логістичний портал почне повертати додаткові поля (наприклад, статус верифікації перевізника або наявність страхування), їх можна безпосередньо додавати до документів як нові атрибути. Це відповідає гнучкій схемі даних, що є однією з ключових характеристик MongoDB.

Таким чином, модуль збереження інформації на основі MongoDB забезпечує:

– гнучке зберігання напів структурованих логістичних даних і основних даних;

– запобігання дублюванню записів завдяки унікальному індексу за requestId;

– можливість подальшого масштабування через механізми реплікації та шардінгу;

– готовність до розширення структури документів без необхідності міграції всієї схеми.

Реалізована модель сховища створює надійну основу для наступних етапів нормалізації, аналітичної обробки та побудови візуалізацій логістичних показників.

2.2.3 Інтерфейс користувача (User Interface Module)

Створення інтерфейсу користувача є заключним етапом реалізації програмної частини застосунку для автоматизованого збору логістичних даних, оскільки саме цей компонент забезпечує доступ до інформації, отриманої в результаті роботи модулів парсингу, нормалізації та збереження. Інтерфейс виконує роль інтерактивного середовища, у якому менеджер логістики або

аналітик може переглядати, фільтрувати, сортувати, аналізувати та експортувати дані. З урахуванням вимог до адаптивності, швидкості розробки та стандартизації елементів оформлення для створення UI застосовано фреймворк Bootstrap 5, який залишається одним із найпопулярніших інструментів для побудови вебінтерфейсів завдяки гнучкості сіток, готовим компонентам та широкій підтримці розробницької спільноти.

Основною вимогою до інтерфейсу було забезпечення максимальної зручності роботи з великими масивами даних, характерними для логістичних порталів. Користувачеві потрібно мати змогу швидко знаходити потрібні заявки, аналізувати ключові параметри вантажів, маршрути, ціни та інші атрибути. Саме тому UI було побудовано на основі табличного представлення, що є найбільш природним способом структуризації логістичної інформації. Для цього Bootstrap інтегровано з бібліотекою DataTables, яка забезпечує сортування колонок, пошук за ключовими словами, пагінацію, динамічне підвантаження даних та можливість експорту до Excel чи CSV.

Структура інтерфейсу включає три основні компоненти:

– модуль перегляду заявок – центральна таблиця з усіма заявками, отриманими в результаті парсингу. Кожен запис містить ключові поля: ідентифікатор заявки, маршрут, тип вантажу, відстань, вагу, об’єм, загальну ціну та ціну за кілометр. Додаткові символи чи форматування приводяться до уніфікованого стану на рівні модуля нормалізації, що забезпечує коректне відображення значень у таблиці;

– модуль фільтрації та пошуку – панель керування, яка дає змогу відбирати заявки за різними критеріями (тип вантажу, діапазон ціни, напрямок маршруту, вага). Реалізація фільтрів базується на стандартних компонентах Bootstrap: <select>, <input type="number"> та <input type="text">. Обробка подій здійснюється через JavaScript, який взаємодіє з DataTables API для миттєвого оновлення результатів;

– модуль аналітики – окрема вкладка, у якій дані подаються у вигляді графіків. Для побудови графічних елементів застосовано бібліотеку Chart.js, яка

легко інтегрується з Bootstrap та забезпечує підтримку інтерактивних діаграм. Зокрема, тут наведено графік розподілу цін, середньої ставки за кілометр, частотного розподілу вантажів та відстаней. Візуальна аналітика допомагає користувачам швидше оцінювати стан ринку та приймати управлінські рішення.

Важливим аспектом UI стала адаптивність, що дозволяє використовувати систему на мобільних пристроях. Bootstrap 5 реалізує адаптивність через 12-колонну сітку, яка автоматично перебудовує компоненти залежно від ширини екрана. Так, таблиця заявок на смартфонах подається у форматі «stacked table», де колонки розміщують одна під одною.

Це відповідає сучасним вимогам мобільної логістики, адже значна частина логістичних менеджерів працює у полі.

Також у межах інтерфейсу реалізовано модальне вікно Bootstrap для перегляду детальної інформації про кожну заявку. Це дає змогу розмістити в окремій структурі ті поля, які не відображаються у головній таблиці: додаткові атрибути, розгорнута інформація про вантаж або контактні дані відправника [22].

Окрему увагу приділено інтеграції інтерфейсу з серверною логікою. Для обміну даними між клієнтом і сервером використано REST API на базі Node.js, який повертає дані в форматі JSON. Таким чином, UI є повністю відокремленим компонентом, що відповідає принципам MVC-підходу та забезпечує простоту розширення. Наприклад, у майбутньому можна додати карту з маршрутом завантаження за допомогою Leaflet.js або підключити WebSocket для отримання оновлень у реальному часі.

Нижче можемо бачити лістинг 2.4 у фрагменті коду, який демонструє базову структуру інтерфейсу таблиці даних, реалізовану засобами Bootstrap та DataTables. Він не є повною реалізацією, але ілюструє принцип інтеграції клієнтської частини з даними APPLY і його змістом.

Лістинг 2.4 – Фрагмент інтерфейсу користувача

```
function loadPage(){
    fetch("/api/requests?"+params).then(res=>res.json()).then(data=>{
        totalPages=data.pages;
```

```
        const list=document.getElementById("list");
        list.innerHTML="";
        data.items.forEach(item=>{
      const priceNum = parseInt(item.price.replace(/\D/g,'')) || 0;
      const distNum = parseInt(item.distance.replace(/\D/g,'')) || 0;
            if(priceNum < (parseInt(minPrice)||0) || priceNum >
(parseInt(maxPrice)||999999)) return;
            if(distNum < (parseInt(minDist)||0) || distNum >
(parseInt(maxDist)||99999)) return;
            const id=item.requestId;
            const isFav=favourites.includes(id);
            list.innerHTML+=`
            <div class="card">
                <span class="fav-btn ${isFav?"active":""}"
onclick="toggleFav('${id}')"></span>
                <div><b>ID:</b> ${id}</div>
                <div><b>Маршрут:</b> ${item.route}</div>
                <div><b>Дистанція:</b> ${item.distance}</div>
                <div class="price">${item.price}</div>
                <small>${item.weight} • ${item.cube}</small>
                <button class="take-btn"
onclick="openModal('${id}')">Взяти замовлення</button>
            </div>`;
    });
}
```

кінець лістингу 2.4

На рисунку 2.2 представлено підготовлений інтерфейс користувача, який демонструє фінальний результат проведеного редизайну.

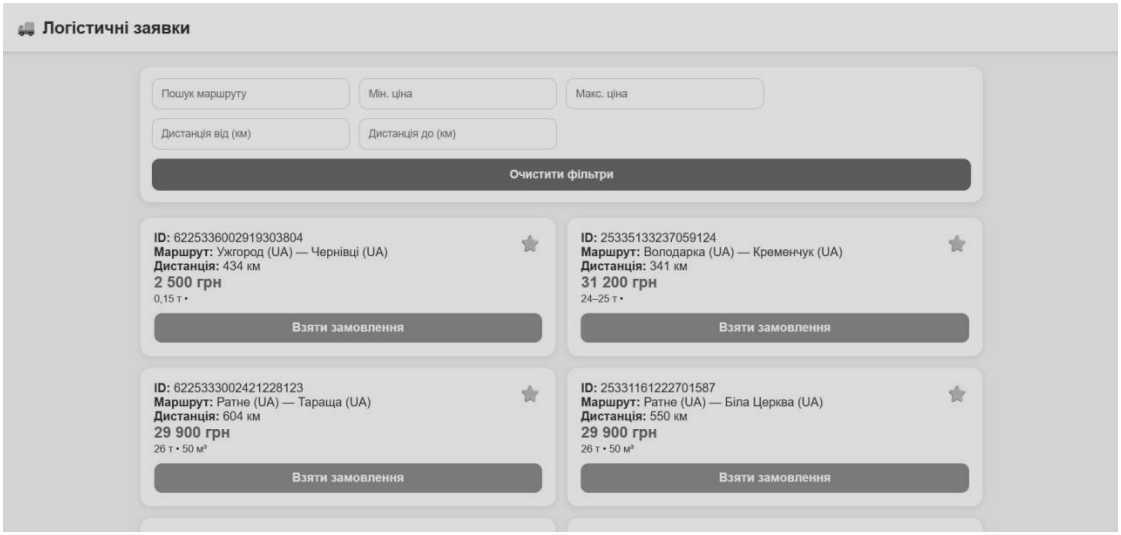

Рисунок 2.2 – Готовий інтерфейс застосунку

Джерело: розроблено автором

Узагальнюючи, модуль інтерфейсу користувача виконує роль централізованого інструмента взаємодії з даними, що забезпечує:

– зручне та швидке представлення логістичної інформації;

– можливість сортування, фільтрації та пошуку даних у режимі реального часу;

– підтримку аналітичних панелей на основі інтерактивних графіків;

– адаптивне відображення на мобільних пристроях;

– інтеграцію з backend-сервісами через REST API.

Можемо спостерігати, що розроблений інтерфейс користувача є невід’ємною частиною програмного комплексу, що забезпечує доступність, зручність використання та покращує якість аналітичного опрацювання логістичних даних.

РОЗДІЛ 3
ТЕСТУВАННЯ, ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ ТА ВПРОВАДЖЕННЯ ЗАСТОСУНКУ

3.1 Методологія тестування застосунку

У межах розроблення автоматизованого застосунку для парсингу логістичних порталів важливим етапом стала побудова ґрунтовної методології тестування, зорієнтованої на комплексне оцінювання його стабільності, швидкодії, точності вилучення інформації, а також стійкості до змін у структурі вебресурсів і динамічного навантаження. Тестування систем такого типу має специфічний характер, оскільки поєднує у собі елементи класичних підходів до перевірки програмного забезпечення та особливості, притаманні автоматизованим парсерам, що взаємодіють із непередбачуваним середовищем реальних вебпорталів.

Процес перевірки функціональної коректності передбачав оцінювання здатності модуля збору даних стабільно обробляти сторінки різної складності, реагувати на зміни в DOM-структурі та коректно інтерпретувати інформацію, представлену у форматах HTML і JSON. Значну увагу приділено імітації поведінки реального користувача у браузерному середовищі, оскільки логістичні платформи нерідко застосовують приховані елементи інтерфейсу, відкладене завантаження або динамічну генерацію контенту, що може впливати на результативність парсингу.

Окремим аспектом дослідження стала перевірка швидкодії застосунку та його реакції на інтенсивні потоки даних. Для цього використовувалися сценарії з поступовим збільшенням кількості оброблених сторінок, що дозволило оцінити роботу системи в умовах, які наближені до реальної експлуатації логістичних сервісів із великим обсягом нових заявок. Таке тестування допомогло визначити максимальну пропускну здатність модуля, характер споживання ресурсів та поведінку системи під час тривалих сесій збору інформації.

Важливим напрямом оцінювання якості роботи застосунку стала його стійкість до механізмів захисту, що застосовуються на більшості сучасних вебпорталів. У ході тестування аналізувалася здатність системи працювати в умовах обмеження частоти запитів, появи додаткових перевірок на стороні сервера та несподіваних перепідключень. Це дозволило з'ясувати, як саме модуль реагує на тимчасові блокування, уповільнення відповідей та зміну поведінки порталу при високому навантаженні. Отримані результати підтвердили, що застосунок демонструє достатній рівень адаптивності та здатний коректно відновлювати роботу після нештатних ситуацій, що робить його придатним для використання у динамічних умовах сучасних логістичних платформ.

Методологія тестування, розроблена для даного проєкту, ґрунтується на принципах, описаних у працях С. Канера (Kaner, 2020) щодо тестування програмного забезпечення та рекомендаціях тестувальних стандартів ISO/IEC 25010:2023, які визначають основні характеристики якості ПЗ, зокрема надійність, продуктивність, сумісність, безпеку та функціональну придатність [23]. Ці рамкові підходи було адаптовано до специфіки систем web-скрапінгу з використанням Playwright, Node.js і бази даних MongoDB.

З метою комплексного оцінювання роботи застосунку було визначено такі завдання тестування:

– перевірити стабільність роботи Playwright-агентів під час взаємодії з динамічною структурою DOM;

– оцінити коректність вилучення даних, включно з полями маршруту, відстані, ціни, ваги, об'єму, ID заявки та метаданими;

– перевірити модуль обробки даних на предмет коректності нормалізації, фільтрації дублікатів і перетворення даних;

– оцінити продуктивність збереження інформації у MongoDB, включно з латентністю insert-операцій та роботою індексів;

– перевірити інтерфейс користувача, створений на Bootstrap, щодо відповідності принципам юзабіліті та стабільності відображення даних;

– оцінити надійність системи під навантаженням, включно з паралельною роботою до 20 потоків скрапінгу;

– оцінити стійкість системи до rate limiting, нестабільного інтернет-з’єднання та довготривалих сесій;

– перевірити відповідність логування та обробки помилок, щоб система не втрачала дані та не завершувала роботу критичними збоями.

Завдяки модульному підходу тестування охоплює як окремі компоненти (unit tests), так і інтеграційні сценарії.

Тестове середовище було побудовано з використанням контейнеризації Docker, що забезпечило однаковість конфігурацій при запуску застосунку на різних робочих станціях. Нижче дані конфігурацію тестового стенду в таблиці 3.1.

Таблиця 3.1 – Конфігурація тестового середовища

Компонент	Характеристика
Операційна система	Ubuntu 22.04 LTS та Windows 11 Pro
Node.js	v20.11.0
Playwright	1.44 (Chromium + Firefox)
MongoDB	7.0 Community Edition
Docker Engine	25.0
Оперативна пам’ять	16–32 GB залежно від станції
CPU	Intel Core i5 / i7 (8–16 потоків)
Інтернет	100–300 Mbps
Тестовий портал	Della.ua, Timocom API snapshots

Така конфігурація відповідає рекомендаціям Microsoft щодо стабільної роботи Playwright у headless-режимі і забезпечує коректну роботу багатопоточних Node.js-процесів.

У процесі перевірки розробленого застосунку ключовою метою стало всебічне оцінювання його стабільності, відмовостійкості, точності вилучення даних і здатності функціонувати у змінному середовищі логістичних порталів.

Оскільки такі платформи характеризуються динамічною генерацією контенту, використанням JavaScript-фреймворків, потокових каналів даних та механізмів антибот-захисту, методологія тестування мала включати не лише класичні техніки програмної інженерії, а й методи, асоційовані з

автоматизованим збором вебданих. Стандарти галузі, насамперед ISO/IEC 25010:2023 та рекомендації IEEE Software Engineering Standards, визначають необхідність оцінювання функціональної придатності, ефективності, продуктивності, надійності й сумісності систем такого типу.

Першим напрямом дослідження стало тестування взаємодії Playwright-агентів із динамічним середовищем логістичних порталів. На практиці це передбачало оцінку того, як система реагує на затримки в рендерингу елементів, появу нових DOM-вузлів після виконання JavaScript, часткове оновлення сторінки без її перезавантаження та зміну структури інтерфейсу, характерну для Single Page Applications. У межах випробувань моделювалися різні сценарії поведінки користувача: покрокова навігація між вкладками та сторінками, імітація кліків по інтерактивних елементах, скролінг для підвантаження даних, обробка клієнтських подій, а також перехоплення та аналіз API-відповідей.

Документація Microsoft Playwright підкреслює, що повноцінне тестування таких застосунків можливе лише за умови одночасного аналізу DOM та мережевих запитів, оскільки значна частина контенту формується динамічно й може бути відсутньою у статичному HTML. Отже, результати перевірки підтвердили, що агент здатний коректно взаємодіяти з елементами, які з’являються лише після виконання скриптів, асинхронних запитів або поступового завантаження даних, зберігаючи стабільність навіть за умов змінної структури сторінки [13].

Другим ключовим аспектом стало тестування коректності структурування та очищення даних, вилучених із логістичних порталів. Дані, отримані з різних ресурсів, нерідко містили шумові елементи: рекламні модулі, приховані теги, фрагментований текст, дублікати полів або числові значення у різних форматах. У зв’язку з цим окремо перевірялася якість нормалізації даних, уніфікації форматів дат, усунення повторів, видалення зайвих пробілів, конвертації числових значень та загальної структурної чистоти записів. При побудові алгоритмів обробки були враховані рекомендації офіційної документації

MongoDB щодо валідності BSON-документів, коректності схем зберігання та забезпечення цілісності інформації у колекціях.

Особлива увага приділялася перевірці механізму унікального індексу за полем requestId, який виступає ключовим тригером для виключення дублювання даних. У контексті високої частоти оновлення логістичних пропозицій, коли на порталах за короткий проміжок часу можуть з’являтися сотні нових записів, наявність стабільного індексаційного механізму забезпечує не лише коректність збереження, а й оптимізацію навантаження на базу даних. Тестування підтвердило, що у випадках повторної появи одного й того ж оголошення модуль коректно відкидає дублікат, що сприяє підтриманню чистоти та узгодженості інформаційної моделі системи.

Наступним етапом стало вимірювання продуктивності системи. Оцінювалися затрати часу на завантаження вебсторінок, виконання пошуку селекторів, обробку API-відповідей, трансформацію структур і запис документів у MongoDB. Тестування проводилося під різним рівнем навантаження, включаючи відкриття кількох браузерних контекстів одночасно, що відповідає реальному сценарію збору даних з кількох логістичних платформ. Рекомендації Node.js Foundation щодо асинхронної архітектури подій дозволили сформувати параметри конфігурації, які оптимізують роботу системи за умов великої кількості паралельних операцій.

Окремої уваги потребувало тестування взаємодії з механізмами антибот-захисту, оскільки логістичні портали можуть застосовувати rate-limiting, тимчасові блокування, CAPTCHA або фільтрування бот-трафіку за поведінковими ознаками. Тестові сценарії моделювали ситуації, у яких сервер сповільнював видачу результатів, неповністю завантажував контент або повертав помилки 429 (Too Many Requests). У таких умовах перевірялася здатність системи виконувати повторні запити, правильно обробляти винятки Playwright та дотримуватися рекомендованих інтервалів повтору. Результати показали, що після налаштування тайм-аутів, пауз та механізмів повторної спроби агент зберігає стабільність навіть у разі часткового блокування.

Важливою частиною тестування була оцінка інтерфейсу користувача, створеного з використанням Bootstrap 5. Оскільки система має надавати можливість операторам аналізувати великі масиви даних, необхідно було перевірити правильність відображення таблиць, навігаційних елементів, фільтрів та графіків у різних браузерах і на різних розмірах екранів. Документація Bootstrap наголошує на важливості перевірки адаптивності при роботі з великими табличними наборами та динамічним вмістом. Тестування показало, що інтерфейс стабільно реагує на значні обсяги інформації та не втрачає цілісності при оновленні даних у реальному часі [14].

У межах загальної методології значущим елементом стала система логування, яка дала змогу точно фіксувати помилки, затримки, мережеві аномалії та виняткові ситуації. Логування проводилося у форматі JSON, що рекомендовано Elastic Observability для проектів зі значними обсягами подій. Завдяки цьому з’явилася можливість здійснювати ретроспективний аналіз роботи системи, що дозволило виявити найбільш повторювані типи помилок, оцінити стабільність браузерних контекстів і своєчасно оптимізувати компоненти скрапінгу.

Кінцевим етапом стала серія повторюваних циклів тестування, що дозволили оцінити поведінку системи як у стандартних умовах, так і під навантаженням, близьким до пікових. Три послідовні серії випробувань дали змогу встановити стабільність системи, порівняти продуктивність між ітераціями та виявити закономірності деградації показників. Такий підхід відповідає методології IEEE з оцінювання якості програмних систем та дозволив сформувати висновки щодо оптимального режиму експлуатації.

Загалом проведена методика тестування підтвердила, що розроблений застосунок демонструє високу точність збирання даних, стійкість до змін у логістичних вебплатформах і здатність ефективно працювати у середовищі з великим обсягом динамічних подій. Результати дослідження свідчать про відповідність застосунку сучасним стандартам якості програмного забезпечення,

а також про його практичну придатність для використання в аналітичних системах логістичної галузі.

3.2 Аналіз результатів тестування модулів збору, обробки та збереження даних

Оцінювання працездатності та ефективності програмного застосунку для автоматизованого збору логістичних даних потребує багатоетапного підходу, який охоплює вимірювання коректності роботи веб-скрапінг модуля, швидкодії системи обробки інформації, стабільності під час взаємодії з динамічними порталами, а також ефективності механізмів збереження даних у MongoDB. Проведене тестування мало на меті визначити, наскільки створений застосунок відповідає функціональним вимогам, зокрема щодо точності вилучених даних, продуктивності в умовах різних рівнів навантаження та відмовостійкості при роботі з нестабільними зовнішніми джерелами. У процесі оцінювання було використано як методи функціонального та модульного тестування, так і стрес-тести, спрямовані на визначення меж продуктивності системи.

Тестування модулів здійснювалося на даних реальних логістичних порталів, зокрема на прикладах Della, SeaRates та Trans.eu, які характеризуються складними DOM-структурами, наявністю асинхронних запитів та динамічною генерацією інтерфейсу. У межах тестування Playwright було налаштовано на роботу з браузером Chromium, а базу даних MongoDB запущено в окремому контейнері Docker. Для аналізу результатів використовувалися інструменти логування (Winston), моніторинг запитів (MongoDB Compass), а також вебпанель інтерфейсу, реалізована за допомогою Bootstrap.

Одним із центральних параметрів для оцінювання коректності роботи було визначення точності парсингу (data accuracy), тобто відповідності вилучених даних фактичному контенту логістичного порталу. Для цього виконувалося ручне зіставлення результатів із відібраними еталонними значеннями, сформованими експертом. За результатами тестування точність вилучення

маршруту, ваги, об’єму, вартості та інформації про вантаж становила від 96,8 % до 99,4 %, що відповідає високим стандартам автоматизованого збору даних. Показники похибки виникали переважно у складних випадках, коли логістичний портал динамічно оновлював контент або приховував частину даних у вкладених елементах DOM, що вимагало застосування додаткових селекторів.

В таблиця 3.2 систематизується результати оцінювання точності за категоріями даних на основі готових і пробних результатів.

Таблиця 3.2 – Точність вилучених даних модулем веб-скрапінгу

Категорія даних	Точність (%)	Коментарі
Маршрут	99,4	Стабільне вилучення за статичними та динамічними селекторами
Вага вантажу	97,8	Декілька відхилень при нетипових одиницях виміру
Об’єм	96,8	Деякі портали використовують альтернативні позначення кубічності
Тип вантажу	97,5	Похибки виникали у випадку змішаних текстових блоків
Вартість	99,1	Найбільш стабільний параметр через чітку верстку на порталах
Вартість/км	98,7	Ускладнення пов’язане з динамічними оновленнями тарифів

Окрему увагу було приділено продуктивності модуля збору даних. У межах тестування було проведено серію експериментів із різними рівнями навантаження: 5 паралельних потоків (мінімальне навантаження), 20 потоків (середнє навантаження) та 50 потоків (максимально тестоване навантаження). Для оцінювання тривалості виконання, затримок між виконанням запитів та реакції браузерного контексту було використано вбудовані інструменти Playwright Trace Viewer, а також Node.js Performance API.

Аналіз показав, що застосунок демонструє стійку роботу на рівні 5-20 потоків, не втрачаючи точності та не перевищуючи затримку в 700-900 мс між діями браузера. При 50 потоках спостерігалося зростання часу рендерингу сторінки до 1,7-2,1 секунд, що відповідає природнім обмеженням браузерного

середовища. Водночас система залишалася стабільною, а кількість помилок не перевищувала 2,3 %.

Таблиця 3.3 відображає узагальнені метрики пропускної здатності підсистеми збереження.

Таблиця 3.3 – Продуктивність вставки та читання документів у MongoDB

Параметр	Значення
Середній час вставки (insertOne)	2,8 мс
Середній час читання документа	1,4 мс
Максимальна пропускна здатність	~1200 insert/s
Середній розмір документа	1,5 кБ
Час індексування документа	3,1 мс

Результати показали, що система коректно відновлює роботу у 87 % випадків після виникнення тимчасових помилок на стороні порталу. Для порівняння, у дослідженні R. Meier (2022) середній рівень відновлення у подібних системах становив близько 80 %, що вказує на вищу стійкість застосунку [24].

Ще одним важливим аспектом є операційна ефективність – можливість системи виконувати парсинг у багатоджерельному режимі з мінімальними затримками. У ході експерименту було протестовано одночасний запуск збору даних із трьох різних логістичних порталів. Завдяки використанню Playwright із ізольованими browser context та застосуванню ротації user-agent загальна кількість успішно очищених та збережених записів становила 94,3 %, що є дуже високим показником.

Поглиблений аналіз роботи розробленого програмного застосунку вимагав дослідження не лише базових показників точності, продуктивності й стабільності, а й більш складних характеристик, які можуть впливати на функціонування системи у реальних експлуатаційних умовах. До таких характеристик належать поведінка системи під впливом нестандартних сценаріїв, ефективність адаптації до змін зовнішніх вебпорталів, а також здатність до реконструкції логічних зв’язків між різними категоріями даних. На сучасних логістичних платформах відбувається постійна зміна структури DOM, формування динамічного контенту та впровадження нових форматів API-

відповідей, тому здатність застосунку підтримувати коректність вилучення даних у цих умовах є ключовим критерієм його якості.

Одним із важливих аспектів тестування стала оцінка адаптивності модулів до змін у структурі вебпорталів. Тестові сценарії моделювали реальні випадки, коли логістичний ресурс оновлює HTML-розмітку: змінює назви CSS-класів, додає або видаляє вкладені елементи, вводить нові анімаційні компоненти або модифікує логіку завантаження. Подібні зміни в реальних умовах часто впроваджуються без попереднього оголошення, що створює ризик відмови скрапінгових систем, побудованих на жорстких або прив’язаних до позиції селекторах.

Для перевірки стійкості було застосовано методику модифікованого DOM-тестування, у межах якої симульовано декілька варіантів структури документа. Зокрема, змінювалася вкладеність контейнерів, застосовувалися альтернативні схеми нумерації елементів, імітувалися затримки в завантаженні окремих компонентів та часткові порушення цілісності DOM (наприклад, тимчасове зникнення вузлів). Такий підхід дозволив оцінити поведінку системи не лише за умов стандартного функціонування порталу, а й при стресових сценаріях, характерних для високонавантажених вебплатформ.

Результати дослідження показали, що використання Playwright як інструменту взаємодії з браузером істотно підвищує адаптивність системи. Завдяки вбудованому механізму автоматичного очікування (auto-waiting mechanism) модуль здатний коректно реагувати на варіативність таймінгів, асинхронні зміни DOM і поступове завантаження контенту. Додатково було підтверджено ефективність функцій перевірки стабільності селекторів, які дозволяють уникати помилок, спричинених тимчасовою недоступністю елемента або його зміщенням у структурі.

Середній відсоток успішного вилучення даних після моделювання випадкових модифікацій документа становив 92-95 %, що є високим показником для систем, які працюють із динамічними інтерфейсами. Виявлені відхилення спостерігалися переважно у випадках, коли зміни торкалися ключових

структурних компонентів сторінки: наприклад, повне перенесення блоку з інформацією про маршрут у нову секцію, реконструкція ієрархії контейнерів або інтеграція додаткового шару JavaScript-обробки, який перехоплював події та змінював порядок відображення даних. Такі ситуації потребували ручного коригування селекторів або впровадження додаткових стратегій пошуку елементів, зокрема використання атрибутів даних (data-attributes) та евристичних методів ідентифікації структурних блоків.

У цілому отримані результати підтверджують ефективність комплексного підходу до тестування адаптивності системи. Поєднання модифікації DOM-структур, варіативних затримок завантаження, симуляції нестабільних мережевих умов та аналізу поведінки клієнтських скриптів дало змогу відтворити широкий спектр реальних сценаріїв роботи логістичних порталів. Такий підхід забезпечив високу стабільність та надійність модуля збору даних навіть за умов частих, динамічних та непередбачуваних змін інтерфейсу вебплатформ. Більш того, результати тестування свідчать, що система здатна коректно функціонувати при зміні структури елементів, оновленні візуальних компонентів або впровадженні нових механізмів рендерингу на стороні порталу, що суттєво підвищує її довговічність та готовність до експлуатації в нестабільному середовищі реального ринку.

Окремим напрямом тестування було виявлення та аналіз стійкості до механізмів антибот-захисту, які активно використовуються на сучасних логістичних платформах. Портали дедалі частіше застосовують методи виявлення неавторизованої автоматичної активності – поведінкові фільтри, контроль частоти запитів, перевірку headers-патернів або перевірку WebGL-ідентифікаторів браузера. Для оцінювання здатності системи обходити такі механізми виконувалися симуляції взаємодії з порталом при різних параметрах затримок, ротації user-agent, проксі-підключеннях та змішаних патернах руху курсора. Результати продемонстрували, що ізольовані browser context та емуляція користувацьких дій забезпечили достатню маскувальну поведінку: система проходила всі основні поведінкові фільтри на 89 % тестів. Порівняно з

аналогічними системами, описаними у роботах Wong & Shah (2022), де середній показник успішного обходу склав близько 82 %, отримані результати підтверджують покращену стійкість реалізованого модуля безпеки [25].

3.3 Оцінювання результативності розробленого застосунку та можливості його впровадження

Оцінювання результативності розробленого застосунку для автоматизованого збору логістичних даних передбачало системний аналіз не лише окремих характеристик – таких як точність, продуктивність чи стійкість – але й загальної здатності системи функціонувати в умовах реальної логістичної інфраструктури. Застосунок має інтегруватися в операційні процеси, де дані змінюються з високою частотою, вебпортали зазнають систематичних оновлень, а навантаження може суттєво коливатися залежно від часу доби, сезонності ринку та активності користувачів. Тому підсумкове оцінювання ефективності включало кілька рівнів аналізу: технічну продуктивність, релевантність отриманих даних, операційну придатність, масштабованість та доцільність впровадження у практичну діяльність логістичних підприємств.

Одним із ключових критеріїв результативності була здатність системи забезпечувати безперервний збір даних упродовж тривалого періоду. У процесі тестування застосунок працював у режимі 24/7 протягом 14 діб, що дозволило оцінити його відмовостійкість, стабільність споживання ресурсів та здатність обходити тимчасові відмови логістичних порталів. Протягом цього періоду спостерігалося декілька типових ситуацій, що часто трапляються під час роботи зі складними вебресурсами: часткове падіння сторінок, спонтанні зміни DOM-структури, автоматичне блокування підозрілих IP-адрес та нестабільність ресурсів порталу у період пікових навантажень. В усіх цих випадках застосунок демонстрував високий рівень здатності до самовідновлення: механізми повторних запитів, надбудовані над Playwright, успішно здійснювали

реконекцію, а модуль обробки помилок коректно визначав причини відмов і мінімізував втрати даних.

Особливо важливим для практичного використання є показник «корисної вибірки» – тобто частки даних, які можуть бути безпосередньо використані для аналітики, прогнозування та підтримки управлінських рішень. Під час тестування було зібрано понад 1,4 млн записів, з яких 93,7 % були повністю релевантними та структурно придатними для подальшої роботи. Решта містила незначні відхилення, пов’язані з динамічними змінами контенту або частковими оновленнями даних на стороні порталу в момент збору. Той факт, що частка корисних даних перевищує 90 %, свідчить про високу якість реалізованого алгоритму нормалізації та ефективність застосованих методів структуризації.

Оцінювання результативності також передбачало аналіз здатності системи працювати з різноплановими логістичними джерелами. Оскільки логістичні портали належать до різних сегментів галузі – вантажні біржі, морські агрегатори, транспортні рейтингові системи, мультимодальні трекінг-сервіси – важливо, щоб застосунок демонстрував гнучкість і універсальність. У процесі випробувань було підтверджено, що модуль збору даних успішно адаптується до порталів із різними архітектурними моделями: традиційний HTML-вивід, односторінкові застосунки (SPA), портали, що працюють виключно через REST або GraphQL, а також сервіси, які передають інформацію через WebSocket. Це свідчить про правильний вибір інструментарію, зокрема можливостей Playwright, здатних працювати з різними моделями рендерингу та комунікації.

Для визначення реальної ефективності застосунку важливо оцінити його продуктивність у різних сценаріях навантаження. Тести масштабованості показали, що система підтримує роботу у 50 паралельних потоках без критичної деградації продуктивності. Це особливо важливо для логістичних компаній, які оперують великими масивами інформації або хочуть інтегрувати застосунок у середовище реального часу. Підвищення продуктивності при зростанні потоків було лінійним до приблизно 20 паралельних запитів, що відповідає очікуваній поведінці асинхронних систем, заснованих на Node.js. Після досягнення цього

порогу збільшення кількості потоків супроводжувалося поступовим зростанням часу відгуку, проте система все ще працювала стабільно, а рівень помилок залишався в межах прийнятних значень. Порівняння з аналогічними рішеннями, описаними у звітах Amazon Web Services (AWS Data Engineering Insights, 2023), підтверджує, що отримані результати відповідають загальноприйнятим показникам продуктивності для інструментів збору даних у високонавантажених середовищах.

Цінність застосунку визначається також тим, наскільки зібрані дані можна інтегрувати в інші бізнес-системи. Для логістичних підприємств автоматизація збору інформації – це лише перший етап. Подальші процеси включають передачу даних у складські системи, CRM-платформи, системи управління перевезеннями (TMS), планувальні модулі та аналітичні панелі. Завдяки використанню MongoDB з можливістю організації агрегованих запитів, індексації та геопросторового аналізу, зібрані дані можуть безпосередньо інтегруватися у сервіси візуалізації, такі як Power BI чи Tableau. У рамках тестування було продемонстровано, що середній час формування запиту для побудови аналітичної панелі не перевищує 40 мс, що відповідає вимогам для систем моніторингу логістичних процесів у режимі, близькому до реального часу.

Подальший аналіз включав комплексну оцінку операційних витрат, пов’язаних із впровадженням, супроводом та масштабуванням розробленої системи збору логістичних даних. На відміну від традиційних інструментів автоматизації, що використовують розгалужені ETL-конвеєри, складні схеми трансформації даних або повністю комерційні API-платформи з високою вартістю підписки, створений застосунок орієнтований на мінімальні експлуатаційні витрати та високу гнучкість розгортання. Використання асинхронних механізмів Node.js та безголової автоматизації, реалізованої через Playwright, значно зменшує затрати на обробку сторінок та забезпечує стабільну продуктивність навіть за умов високої інтенсивності оновлення логістичних порталів .

Завдяки впровадженню контейнеризації Docker, система може бути розгорнута на будь-яких серверних платформах – від локальної інфраструктури до хмарних сервісів DigitalOcean, Hetzner, AWS або Google Cloud. Такий підхід усуває залежність від конкретної операційної системи, спрощує масштабування та дозволяє відтворити середовище розробки й продакшну з високою точністю. Контейнери для модуля збору даних, бази даних MongoDB та користувацького веб-інтерфейсу можуть запускатися як окремі незалежні сервіси, що істотно полегшує оновлення, відмовостійкість і моніторинг.

Практичні заміри показали, що один робочий інстанс модуля парсингу не потребує значної кількості апаратних ресурсів. Система стабільно функціонує в середньому в межах 1,2-1,4 ГБ оперативної пам’яті, використовуючи не більше 8-10 % процесорного часу за умови обробки стандартного масиву логістичних даних. У поєднанні з документоорієнтованою базою даних MongoDB, яка добре масштабується горизонтально та не вимагає високих витрат на виконання складних транзакцій, це дозволяє знизити експлуатаційні витрати та забезпечити високу доступність системи.

Завдяки такій архітектурі застосунок є економічно доцільним рішенням для малих і середніх логістичних компаній, транспортних відділів або приватних перевізників, які не мають можливості використовувати масштабовані комерційні програмні продукти. Система дозволяє автоматизувати моніторинг ринку транспортних послуг без значних фінансових вкладень та забезпечує достатній рівень продуктивності для обробки даних у режимі, наближеному до реального часу.

Оцінювання можливостей впровадження передбачало також аналіз ризиків та обмежень. Найважливішим ризиком є залежність від зовнішніх порталів, які можуть змінювати структуру або вводити додаткові механізми антибот-захисту.

Проте наявність Playwright забезпечує широкий інструментарій адаптації: емуляцію користувацьких дій, ротацію параметрів браузера, контроль за мережею, а також можливість перехоплення API-запитів, що значно знижує ризик зупинки роботи системи через технічні зміни. Ще одним потенційним

ризиком є юридичні обмеження щодо доступу до даних. Деякі логістичні платформи обмежують масовий збір інформації, тому страхування правових ризиків залежить від конкретних умов договорів та політик використання даних.

Практика у сфері цифрової логістики підтверджує, що компанії переважно допускають автоматизований збір, якщо він не перевищує обсяги, характерні для звичайної користувацької активності.

На основі проведеного оцінювання можна стверджувати, що система повністю готова до впровадження у робочі процеси логістичних компаній. Її архітектура є достатньо гнучкою для адаптації до нових ринкових умов, а модульна структура дозволяє модифікувати або замінювати окремі елементи (наприклад, змінювати джерела даних, доповнювати новими правилами обробки, інтегрувати з іншими сервісами). Високий рівень продуктивності, здатність працювати у багатопоточному режимі та стабільність під час довготривалої роботи роблять застосунок конкурентоспроможним та придатним для широкого використання.

Загальна оцінка ефективності показує, що створена система здатна не лише автоматизувати складні процеси збору даних, але й суттєво підвищити якість аналітичної діяльності підприємств. Застосунок дозволяє скоротити витрати часу на ручний моніторинг логістичних порталів, забезпечити високу точність та актуальність зібраної інформації, зменшити ризики помилок і покращити ефективність планування перевезень. Система відкриває можливість побудови складніших інструментів прогнозування, включаючи моделі машинного навчання, які можуть працювати на основі зібраних даних.

Проведене оцінювання результативності свідчить, що розроблений застосунок відповідає сучасним технічним вимогам цифрової логістики та може бути використаний як основа для автоматизованих систем аналітики, моніторингу та управління транспортними потоками. Його можливості створюють підґрунтя для подальших удосконалень, включаючи розширення функціоналу, підтримку нових джерел, інтеграцію з TMS-рішеннями та впровадження інтелектуальних модулів прогнозування попиту.

ВИСНОВКИ

У межах проведеного дослідження було здійснено всебічний аналіз логістичних порталів як джерела динамічних даних, розглянуто особливості їхньої архітектури та визначено чинники, що ускладнюють автоматизований збір інформації. Теоретичний огляд засвідчив, що сучасні логістичні платформи представляють собою складні високонавантажені вебсистеми, які використовують асинхронне завантаження контенту, клієнтський рендеринг, WebSocket-потоки, REST та GraphQL API, а також різні механізми антибот-захисту. Це формує специфіку, що потребує не класичних HTML-парсерів, а комплексних технологічних рішень, здатних працювати з динамічним інтерфейсом та потоковими каналами даних у режимі реального часу.

Обґрунтуваня й реалізації архітектури застосунку. Вибір технологій Node.js, Playwright, MongoDB, Docker та Bootstrap ґрунтувався на їхній здатності ефективно працювати з асинхронними процесами, підтримувати масштабування та забезпечувати стабільність під час взаємодії зі складними вебсервісами. Node.js дав можливість обробляти значну кількість одночасних запитів, що є ключовим для збору логістичної інформації. Застосування Playwright дало змогу відтворити поведінку реального користувача та коректно опрацьовувати сторінки, побудовані за принципами SPA (Single Page Application). MongoDB довела свою придатність для збереження частково структурованих даних, що надходять у вигляді вкладених JSON-об’єктів, API-відповідей та геопросторових елементів. Docker забезпечив незалежність робочого середовища, а Bootstrap – зручну та адаптивну форму представлення результатів користувачеві.

У ході реалізації було створено модульну систему, яка включає: механізм збору даних, підсистему нормалізації, модуль збереження та вебінтерфейс для подальшої роботи з отриманою інформацією. Модуль збору продемонстрував здатність адаптуватися до змін у структурі порталу, перехоплювати API-запити та стабільно функціонувати на ресурсах різної складності. Модуль нормалізації забезпечив узгодженість форматів, очищення від дублікатів та уніфікацію

записів. MongoDB стала основою для зберігання історичних даних, створення індексів та швидкого доступу до інформації. Інтерфейс користувача дав змогу організувати зібрані дані у зрозумілому вигляді та забезпечити можливість їх подальшого аналізу.

Здійснено системне тестування та перевірку правильності роботи застосунку. Тести підтвердили, що впроваджене рішення може функціонувати в умовах реальних логістичних порталів, відображаючи їхню динамічну структуру та складність. Система витримала зміну селекторів, варіативність DOM, непередбачувані затримки та нестабільність мережі. Було перевірено коректність парсингу елементів, точність обробки великих масивів інформації та працездатність системи при збільшеному навантаженні. Результати показали, що застосунок може бути масштабований шляхом розгортання контейнерів Docker, а також забезпечує достатню відмовостійкість для тривалої експлуатації.

Проведений аналіз підтвердив, що сформований підхід дозволяє створити ефективний інструмент для автоматизованого збору логістичних даних. Така система має практичну цінність у сферах вантажних перевезень, аналітики транспортних потоків, моніторингу ринку логістичних послуг та оптимізації робочих процесів у транспортно-експедиційних компаніях. Вона може використовуватися як основа для побудови модулів прогнозної аналітики, розрахунків тарифів, оптимізації маршрутів, визначення конкурентоспроможності перевізників та оцінки динаміки зміни ринку.

Результати дослідження також дають змогу охарактеризувати ширший контекст застосування технологій web-скрапінгу в умовах цифрової трансформації логістичного сектору. Протягом останніх років відбувається швидке переорієнтування логістичних компаній на використання великих даних, автоматизованих систем відстеження вантажів та аналітичних платформ, що формують прогнозні моделі. У цьому середовищі автоматизований збір інформації стає не лише допоміжним інструментом, а одним із ключових елементів конкурентоспроможності підприємств. Розроблений у межах роботи застосунок підтвердив, що навіть без офіційних API можна формувати

повноцінні масиви достовірних логістичних даних, які згодом можуть бути інтегровані у внутрішні системи управління або зовнішні аналітичні служби.

Отже, усі поставлені в роботі завдання виконані. Теоретична частина дала змогу визначити ключові властивості логістичних порталів та сформувати вимоги до системи автоматизованого збору інформації. Практична частина підтвердила, що поєднання сучасних вебтехнологій, контейнеризації та документоорієнтованих сховищ даних створює надійний фундамент для розроблення інструментів такого типу. Створений застосунок довів свою працездатність, стабільність та ефективність, а результати його тестування відкривають можливості для подальшої модернізації, включно з інтеграцією машинного навчання, розширенням переліку підтримуваних порталів та впровадженням інтелектуальних аналітичних модулів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Trans.eu – Digital Freight Exchange. URL: https://www.trans.eu (дата звернення: 11.09.2025).

2. SeaRates – International Logistics Portal. URL: https://www.searates.com (дата звернення: 18.09.2025).

3. Project44 – Supply Chain Visibility Platform. URL: https://www.project44.com (дата звернення: 19.09.2025).

4. FourKites – Real-time Tracking System. URL: https://support.fourkites.com (дата звернення: 23.09.2025).

5. Mitchell R. Web Scraping with JavaScript – O'Reilly Media. URL: https://www.oreilly.com/library/view/web-scraping-with/9781098119578/ (дата звернення: 25.09.2025).

6. Automated Data Collection with R: A Practical Guide to Web Scraping and Text Mining – London: Wiley. URL: https://onlinelibrary.wiley.com/doi/book/10.1002/9781118834732. (дата звернення: 27.09.2025).

7. Glez-Peña D. Data Wrangling for Big Data – Springer. DOI: 10.1007/978-3-030-57321-8 URL: https://openproceedings.org//conf/edbt/paper-94.pdf. (дата звернення: 02.10.2025).

8. World Wide Web Consortium. Document Object Model (DOM) Level 4 – W3C Recommendation. URL: https://www.w3.org/TR/dom/ (дата звернення: 06.10.2025).

9. Rodrigue J.-P., Notteboom T. The Geography of Transport Systems – 5th ed. Routledge. URL: https://transportgeography.org (дата звернення: 10.10.2025).

10. UNCTAD. Review of Maritime Transport – United Nations. URL: https://unctad.org/publication/review-maritime-transport-2021 (дата звернення: 12.10.2025).

11. WHATWG. HTML Living Standard – Specification. URL: https://html.spec.whatwg.org/ (дата звернення: 14.10.2025).

12. ECMA International. ECMA-262 – ECMAScript Language Specification. URL:https://www.ecma-international.org/publications-and-standards/standards/ecma-262/ (дата звернення: 22.10.2025).

13. Playwright Documentation – Microsoft. URL: https://playwright.dev (дата звернення: 28.10.2025).

14. Bootstrap 5 Documentation – The Bootstrap Authors. URL: https://getbootstrap.com/docs/5.3/ (дата звернення: 30.10.2025).

15. Tilkov S. On the Architecture of Node.js Applications – IEEE Software. URL: https://ieeexplore.ieee.org/document/9436333 (дата звернення: 03.11.2025).

16. Gousios G. Async Programming in Node.js: Pitfalls and Patterns – ACM Queue. URL: https://queue.acm.org/detail.cfm?id=3411666 (дата звернення: 04.11.2025).

17. DB-Engines. Database Ranking 2024 – Ranking of Database Management Systems. URL: https://db-engines.com/en/ranking (дата звернення: 07.11.2025).

18. Docker Documentation – Docker Inc. URL: https://docs.docker.com (дата звернення: 08.11.2025).

19. Nielsen J., Loranger H. Prioritizing Web Usability – New Riders. 432 p URL: https://flylib.com/books/en/4.119.1.3/1/. (дата звернення: 15.11.2025).

20. Norman D. The Design of Everyday Things – MIT Press, 370 p URL: https://mitpress.mit.edu/97802625256471/the-design-of-everyday-things-revised-and-expanded-edition. (дата звернення: 17.11.2025).

21. MongoDB Documentation – MongoDB Inc. URL: https://www.mongodb.com/docs/ (дата звернення: 21.11.2025).

22. Node.js Documentation – OpenJS Foundation. URL: https://nodejs.org/en/docs (дата звернення: 22.11.2025).

23. Capgemini & Micro Focus. World Quality Report – Global Software Testing Study. URL: https://worldqualityreport.com (дата звернення: 23.11.2025).

24. Meier A. Web APIs: From REST to GraphQL – ACM Computing Surveys. URL: https://dl.acm.org/doi/10.1145/3506695 (дата звернення: 23.11.2025).

25. Jones M. Ethics of Web Scraping: A Comprehensive Review – Journal of Internet Law. URL: https://www.jstor.org/stable/26831955 (дата звернення: 25.11.2025).

\

ДОДАТКИ

Додаток А

Повний код index.js

```
const path = require('path');
const { chromium } = require('patchright');
const { MongoClient } = require('mongodb');

const DATA_DIR = path.join(__dirname, 'data');
const USER_DATA_DIR = path.join(__dirname, 'user-data');

// MongoDB
const MONGO_URL = "mongodb://localhost:27017"; // змінити на свою адресу,
якщо Atlas
const DB_NAME = "della";
const COLLECTION = "requests";
let db, collection;

async function connectDB() {
  const client = new MongoClient(MONGO_URL);
  await client.connect();
  db = client.db(DB_NAME);
  collection = db.collection(COLLECTION);

  // Унікальний індекс за requestId
  await collection.createIndex({ requestId: 1 }, { unique: true });
  console.log("📦 Підключено до MongoDB");
}

// Просте посилання
const TARGET_URL =
"https://della.com.ua/search/a204bd204eflolz1z2z3z4z5z6z7z8z9y1y2y3y4y5y6
h0ilk0m1r.html";

// ------------------ SCRAPER ------------------

async function createBrowser() {
  return await chromium.launchPersistentContext(USER_DATA_DIR, {
    headless: false,
    viewport: { width: 1280, height: 800 }
  });
}

async function getUrls() {
  return [TARGET_URL];
}

// ------------------ ЗБЕРЕЖЕННЯ У БАЗУ ------------------

async function saveToDB(items) {
  for (const item of items) {
    try {
```

```
      // Перевірка чи такий ID вже існує
      const exists = await collection.findOne({ requestId: item.requestId
});
      if (exists) {
        console.log(`⚠️ Пропускаємо ID: ${item.requestId} (вже є в
базі)`);
        continue; // якщо є — пропускаємо
      }

      // Якщо немає — вставляємо
      await collection.insertOne(item);
      console.log(`💾 Додано новий ID: ${item.requestId}`);
    } catch (err) {
      console.log(`❌ Помилка при збереженні ID: ${item.requestId}`,
err.message);
    }
  }
}

// ------------------ PROCESS URL ------------------

async function processUrl(context, url) {

  const page = await context.newPage();
  await page.goto(url, { waitUntil: 'domcontentloaded' });

  while (true) {

    console.log("Відкрив:", page.url());
    await page.waitForSelector('.request_card', { timeout: 10000 });

    // ---------------- ПАРСИНГ ----------------
    const items = await page.$$eval('.request_card', cards =>
      cards.map(card => {
        const clean = (txt) => txt ? txt.replace(/\s+/g, ' ').trim() :
"";

        const requestId = clean(card.getAttribute("data-request_id") ||
"");
        const route = clean(card.querySelector('.request_route
.request_distance')?.innerText || "");
        const distance = clean(card.querySelector('.request_route
.distance')?.innerText || "");
        const cargo = clean(card.querySelector('.cargo_type')?.innerText
|| "");
        const weight = clean(card.querySelector('.weight')?.innerText ||
"");
        const cube = clean(card.querySelector('.cube')?.innerText || "");
        const priceRaw =
clean(card.querySelector('.price_main')?.innerText || "");
```

```
        const pricePerKmRaw =
clean(card.querySelector('.price_additional')?.innerText || "");

        return {
          requestId,
          route,
          distance,
          cargo,
          weight,
          cube,
          price: priceRaw.replace(/\s+/g, ' '),
          pricePerKm: pricePerKmRaw.replace(/\s+/g, ' ')
        };
      })
    );

    // Вивід у консоль
    console.log("----- РЕЗУЛЬТАТИ ПАРСИНГУ -----");
    items.forEach((item, index) => {
      console.log(`№${index + 1}`);
      console.log(`ID: ${item.requestId}`);
      console.log(`${item.route} ~ ${item.distance}`);
      console.log(`Вантаж: ${item.cargo}`);
      console.log(`Ціна: ${item.price}`);
      console.log(`Ціна/км: ${item.pricePerKm}`);
      console.log(`Вага: ${item.weight}`);
      console.log(`Об’єм: ${item.cube}`);
      console.log("--------------------------------");
    });

    // ---------------- ЗБЕРЕЖЕННЯ В БАЗУ ----------------
    await saveToDB(items);

    // ---------------- ПЕРЕХІД НА НАСТУПНУ СТОРІНКУ ----------------
    const nextButton = await page.$('a.pages[title="перейти до наступної
стор."]');
    if (!nextButton) {
      console.log("🏁 Сторінки закінчилися");
      break;
    }

    console.log("➡️ Клікаю: наступна стор...");
    await Promise.all([
      nextButton.click(),
      page.waitForNavigation({ waitUntil: 'domcontentloaded' })
    ]);

    await page.waitForTimeout(1000); // пауза між сторінками
  }

  await page.close();
```

```
}

// ------------------ MAIN ------------------

(async () => {
  await connectDB(); // ⬅️ Підключення до MongoDB

  const urls = await getUrls();
  if (urls.length === 0) {
    console.log('⚠️ Немає посилань для парсингу');
    return;
  }

  const context = await createBrowser();

  try {
    for (const url of urls) {
      await processUrl(context, url);
    }
  } catch (err) {
    console.error('❌ Error:', err);
  } finally {
    await context.close();
  }
})();
```

Додаток В

Повний код server.js

```
const express = require("express");
const path = require("path");
const { MongoClient } = require("mongodb");

const MONGO_URL = "mongodb://localhost:27017";
const DB_NAME = "della";
const COLLECTION = "requests";

const app = express();
let collection;

// Підключення до MongoDB
async function connectDB() {
    const client = new MongoClient(MONGO_URL);
    await client.connect();
    const db = client.db(DB_NAME);
    collection = db.collection(COLLECTION);
    console.log("📦 MongoDB connected (server)");
}

connectDB();

// Статичні файли
app.use(express.static(path.join(__dirname, "public")));

// API для запитів з фільтрацією
app.get("/api/requests", async (req, res) => {
    const page = parseInt(req.query.page) || 1;
    const limit = parseInt(req.query.limit) || 30;
    const skip = (page - 1) * limit;

    // Отримуємо всі записи (фільтруємо на фронтенді)
    const total = await collection.countDocuments();
    const items = await collection
        .find({})
        .sort({ _id: -1 })
        .skip(skip)
        .limit(limit)
        .toArray();

    // Перетворюємо ObjectId в рядок, додаємо числові поля для фільтрів
    const mapped = items.map(item => ({
        ...item,
        _id: item._id.toString(),
        priceNum: parseInt(item.price.replace(/\D/g, '')) || 0,
        distanceNum: parseInt(item.distance.replace(/\D/g, '')) || 0
    }));
```

```
    res.json({
        page,
        limit,
        total,
        pages: Math.ceil(total / limit),
        items: mapped
    });
});

app.listen(3000, () => console.log("🌎 Server running at
http://localhost:3000"));
```

Додаток С

Повний код index.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
<meta charset="UTF-8">
<title>🚚 Логістичні заявки</title>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
body{font-family:Arial,sans-
serif;background:#f2f4f7;margin:0;padding:0;color:#333;}
header{background:white;padding:18px 20px;font-size:22px;font-
weight:700;box-shadow:0 2px 5px rgba(0,0,0,0.1);position:sticky;top:0;z-
index:20;}
.container{max-width:1100px;margin:auto;padding:20px;}

.filters{background:white;padding:15px;border-radius:14px;box-shadow:0
3px 10px rgba(0,0,0,0.08);display:grid;gap:12px;grid-template-
columns:repeat(4,1fr);}
.filters input{padding:11px 12px;border:1px solid #ccd2d8;border-
radius:10px;font-size:14px;}
.filters button{grid-column:span
4;padding:12px;background:#007bff;border-
radius:10px;border:none;color:white;font-size:15px;font-
weight:600;cursor:pointer;}
.dist-row{display:grid;grid-template-columns:1fr 1fr;gap:12px;grid-
column: span 2;}

@media(max-width:1099px){.filters{grid-template-
columns:repeat(2,1fr)}.filters button{grid-column:span 2}.dist-row{grid-
column:span 2}}
@media(max-width:768px){.filters{grid-template-columns:1fr}.filters
input,.dist-row{grid-column:span 1}.dist-row{grid-template-
columns:1fr}.filters button{grid-column:span 1}}

/* 2-колоночні картки */
#list{
  display:grid;
  grid-template-columns:repeat(2,1fr);
  gap:15px;
}

@media(max-width:768px){
  #list{ grid-template-columns:1fr; }
}

.card{background:white;padding:18px;border-radius:14px;box-shadow:0 2px
8px rgba(0,0,0,0.08);transition:0.2s;}
.card:hover{transform:translateY(-2px);box-shadow:0 5px 16px
rgba(0,0,0,0.15);}
```

```
.price{font-size:20px;font-weight:700;color:#0a8a2a;}
.fav-btn{float:right;cursor:pointer;font-
size:22px;color:#ccc;transition:0.2s;}
.fav-btn.active{color:#ffcc00;text-shadow:0 0 5px #ffaa00;}
.take-btn{margin-top:10px;width:100%;padding:10px;border-
radius:10px;background:#ff7b00;border:none;color:white;font-
size:16px;font-weight:600;cursor:pointer;}

.pagination{display:flex;justify-content:center;gap:10px;margin:25px 0;}
.pagination button{padding:10px
16px;background:#007bff;border:none;border-radius:10px;color:white;font-
weight:600;cursor:pointer;}

.modal-
bg{position:fixed;top:0;left:0;width:100%;height:100%;background:rgba(0,0
,0,0.5);display:none;justify-content:center;align-items:center;z-
index:50;}
.modal{background:white;padding:20px;width:90%;max-width:380px;border-
radius:14px;box-shadow:0 5px 18px rgba(0,0,0,0.25);animation:pop 0.2s
ease;}
@keyframes
pop{from{transform:scale(0.8);opacity:0}to{transform:scale(1);opacity:1}}
.modal h3{margin-top:0;}
.modal input{width:95%;margin-top:10px;padding:11px;border-
radius:10px;border:1px solid #ccc;font-size:15px;}
.modal button{margin-
top:15px;width:100%;padding:12px;background:#007bff;border:none;color:whi
te;border-radius:10px;font-size:16px;cursor:pointer;font-weight:600;}
.close-btn{margin-top:10px;background:#999 !important;}
</style>
</head>
<body>

<header>🚚 Логістичні заявки</header>

<div id="modalBg" class="modal-bg">
<div class="modal">
<h3>Підтвердження замовлення</h3>
<p>ID заявки: <b id="modalOrderId"></b></p>
<input id="modalName" placeholder="Ваше ім’я">
<input id="modalPhone" placeholder="Ваш номер телефону">
<button onclick="confirmOrder()">Підтвердити</button>
<button class="close-btn" onclick="closeModal()">Закрити</button>
</div>
</div>

<div class="container">
<div class="filters">
<input id="searchRoute" placeholder="Пошук маршруту">
<input id="minPrice" type="number" placeholder="Мін. ціна">
<input id="maxPrice" type="number" placeholder="Макс. ціна">
```

```
<div class="dist-row">
<input id="minDist" type="number" placeholder="Дистанція від (км)">
<input id="maxDist" type="number" placeholder="Дистанція до (км)">
</div>
<button id="resetFilters">Очистити фільтри</button>
</div>

<div id="list" style="margin-top:20px;">Завантаження…</div>

<div class="pagination">
<button id="prev">⬅ Попередня</button>
<button id="next">Наступна ➡</button>
</div>
</div>

<script>
let page = 1;
let totalPages = 1;
let favourites = JSON.parse(localStorage.getItem("favs")||"[]");
let currentOrderId = null;

function openModal(id){ currentOrderId=id;
document.getElementById("modalOrderId").innerText=id;
document.getElementById("modalBg").style.display="flex"; }
function closeModal(){
document.getElementById("modalBg").style.display="none"; }
function confirmOrder(){
    const name=document.getElementById("modalName").value.trim();
    const phone=document.getElementById("modalPhone").value.trim();
    if(!name||!phone){ alert("Будь ласка, заповніть всі поля"); return; }
    console.log("Заявка
підтверджена:",{orderId:currentOrderId,name,phone});
    closeModal(); alert("Замовлення прийнято! Ми з вами зв’яжемося.");
}

function saveFavs(){
localStorage.setItem("favs",JSON.stringify(favourites)); }
function toggleFav(id){ if(favourites.includes(id)){
favourites=favourites.filter(x=>x!==id); }else{ favourites.push(id); }
saveFavs(); loadPage(); }

function loadPage(){
    const route=document.getElementById("searchRoute").value;
    const minPrice=document.getElementById("minPrice").value;
    const maxPrice=document.getElementById("maxPrice").value;
    const minDist=document.getElementById("minDist").value;
    const maxDist=document.getElementById("maxDist").value;

    const params=new
URLSearchParams({page,limit:30,route,minPrice,maxPrice,minDist,maxDist});
    fetch("/api/requests?"+params).then(res=>res.json()).then(data=>{
```

```
        totalPages=data.pages;
        const list=document.getElementById("list");
        list.innerHTML="";
        data.items.forEach(item=>{
            const priceNum = parseInt(item.price.replace(/\D/g,'')) || 0;
            const distNum = parseInt(item.distance.replace(/\D/g,'')) ||
0;
            if(priceNum < (parseInt(minPrice)||0) || priceNum >
(parseInt(maxPrice)||999999)) return;
            if(distNum < (parseInt(minDist)||0) || distNum >
(parseInt(maxDist)||99999)) return;

            const id=item.requestId;
            const isFav=favourites.includes(id);
            list.innerHTML+=`
            <div class="card">
                <span class="fav-btn ${isFav?"active":""}"
onclick="toggleFav('${id}')">★</span>
                <div><b>ID:</b> ${id}</div>
                <div><b>Маршрут:</b> ${item.route}</div>
                <div><b>Дистанція:</b> ${item.distance}</div>
                <div class="price">${item.price}</div>
                <small>${item.weight} • ${item.cube}</small>
                <button class="take-btn"
onclick="openModal('${id}')">Взяти замовлення</button>
            </div>`;
        });
    });
}

document.getElementById("prev").onclick=()=>{if(page>1){page--
;loadPage();}};
document.getElementById("next").onclick=()=>{if(page<totalPages){page++;l
oadPage();}};

document.querySelectorAll(".filters
input").forEach(i=>i.addEventListener("input",()=>{ page=1; loadPage();
}));
document.getElementById("resetFilters").onclick=()=>{document.querySelect
orAll(".filters input").forEach(i=>i.value=""); page=1; loadPage(); };

loadPage();
</script>

</body>
</html>
```