

**МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ**

**ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І
СИСТЕМ**

КВАЛІФІКАЦІЙНА РОБОТА

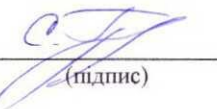
на тему: Розробка сайту для придбання авто із США

Здобувача вищої освіти першого (бакалаврського) рівня,
IV курсу, групи КБ-22-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології (за галузями)»


(підпис)

Войтенко Микита
Русланович
(прізвище, ім'я, по батькові)

Науковий керівник


(підпис)

Грищенко Світлана
Миколаївна, канд. пед. наук

(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Керівник освітньо-
професійної програми


(підпис)

Ніжегородцев Владислав
Олександрович, канд. пед.
наук, доцент

(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Допущено до захисту за рішенням кафедри комп'ютерних та інформаційних
технологій і систем протокол від «12» травня 2026 № 20

Завідувач кафедри


(підпис)

Філоненко Михайло
Миколайович, канд. фіз.-мат.
наук, доцент

(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Реєстрація № 44

27.05.2026
(дата)


(підпис відповідальної особи)

Результати перевірки

до захисту
(до захисту, на доопрацювання)

12.05.2026
(дата)


(підпис)

Результат захисту

5 (відмінно)
(оцінка)

08.06.2026
(дата)


(підпис)

Голова комісії:


(підпис)

О.С. Дюбанов

Члени комісії:


(підпис)

М.М. Філоненко


(підпис)

В.О. Ніжегородцев


(підпис)

А.А. Омельчук

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії, секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	27		08.06.2026 Грищенко С.М. Дюбанов О.С.
2. Дотримання вимог щодо оформлення	від 0 до 10	9		Філоненко М.М. Ніжегородцев В.О.
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	4		Омельчук А.А. Суховерхівка Л.П.
4. Рецензент	від 0 до 5	5		
РАЗОМ БАЛІВ	від 0 до 50	45		

Оцінювання кваліфікаційної роботи науковим керівником – 45 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
90	A	5 (відмінно)	08.06.2026	Дюбанов О.С. Філоненко М.М. Ніжегородцев В.О. Омельчук А.А. Суховерхівка Л.П.

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	
50-59	E	3 (задовільно)
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій

Кафедра комп'ютерних та інформаційних технологій і систем

Галузь знань: 12 «Інформаційні технології»

Спеціальність: 122 «Комп'ютерні науки»

Освітня програма: «Інформаційні управляючі системи і технології (за галузями)»

Затверджую

Завідувач кафедри  Філоненко М.М.

«01» грудня 2025

Завдання на кваліфікаційну роботу
здобувачу вищої освіти

Войтенку Микиті Руслановичу

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи Розробка сайту для придбання авто із США

Затверджена наказом ректора від «28» листопада 2025 №1526

2. Строк здачі закінченої роботи «08» травня 2026

3. Мета роботи розробка вебзастосунку для агрегації даних про авто із США та автоматизації розрахунку митних платежів.

Об'єкт дослідження процес розробки та функціонування інформаційно-аналітичних систем на базі вебтехнологій.

Предмет дослідження програмна реалізація клієнт-серверного застосунку для збору та аналізу даних про авто із США.

Орієнтована база дослідження онлайн-маркетплейс AUTO.RIA та нормативна база України з питань митного оформлення.

4. Вихідні дані до роботи вебсайт, методи вебскрейпінгу, алгоритми нормалізації даних, технологічний стек Python/Flask та створений.

5. Перелік графічного матеріалу головна сторінка, результати пошуку, картка лота з фінансовими показниками, схема розрахунку мита, архітектура системи.

6. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ЗМІСТ

ВСТУП РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ТЕХНОЛОГІЙ

1.1 Аналіз ринку авто із США та огляд існуючих рішень.

1.2 Обґрунтування вибору інструментарію (Python, Flask, BeautifulSoup). РОЗДІЛ

2. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

2.1 Проєктування архітектури системи та алгоритмів парсингу.

2.2 Розробка інтерфейсу користувача та відображення результатів.

РОЗДІЛ 3. ОЦІНКА ЕФЕКТИВНОСТІ ВЕБЗАСТОСУНКУ ТА ПЕРСПЕКТИВИ ЙОГО ВДОСКОНАЛЕННЯ

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 30.10.2025	До 30.10.2025
2	I розділ (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	26.12.2025	26.12.2025
3	II розділ (аналіз експериментальної частини про методики досліджень)	18.02.2026	18.02.2026
4	III розділ (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	16.04.2026	16.04.2026
5	Висновки	01.05.2026	01.05.2026
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	07.05.2026	07.05.2026
7	Отримання відгуку керівника	29.05.2026	29.05.2026
8	Отримання зовнішньої рецензії	01.06.2026	01.06.2026

8. Керівник кваліфікаційної роботи

9. Керівник освітньо-професійної програми

10. Завдання до виконання отримав
здобувач вищої освіти

(підпис)

Грищенко С.М.
(прізвище, ініціали)


(підпис)

Ніжегородцев
(прізвище, ініціали)


(підпис)

Войтенко М.Р.
(прізвище, ініціали)

АНОТАЦІЯ

Розробка вебсайту для придбання авто із США

Мета роботи полягає вебзастосунку для автоматизованого збору даних про авто із США та розрахунку вартості розмитнення.

Завданнями роботи є: аналіз ринку імпорту автомобілів, обґрунтування вибору технологічного стека (Python, Flask, BeautifulSoup), проектування клієнт-серверної архітектури з алгоритмами вебскрейпінгу, розробка адаптивного інтерфейсу та імплементація митного калькулятора для оцінки економічної доцільності імпорту.

Предметом дослідження є програмна реалізація застосунку для агрегації даних.

Об'єктом дослідження є веборієнтованих інформаційно-аналітичних систем.

Базою дослідження виступають інформаційні платформи-агрегатори автомобільних оголошень (зокрема AUTO.RIA) та нормативно-правова база України з питань митного оформлення.

За результатами дослідження створено готовий до експлуатації програмний продукт, який оптимізує процеси прийняття рішень для фізичних осіб та автодилерів шляхом автоматизації моніторингу ринку та розрахунку митних платежів.

Одержані результати можуть бути використані покупцями для контролю витрат при імпорті авто, а застосовані архітектурні підходи - ІТ-спеціалістами при розробці аналогічних сервісів агрегації даних.

Кваліфікаційна робота містить 68 сторінок, 4 таблиць, 4 рисунків, список використаних джерел 52 найменувань.

Ключові слова: вебзастосунок, Python, Flask, BeautifulSoup, вебскрейпінг, парсинг даних, митний калькулятор, імпорт авто, США, AUTO.RIA.

Рік виконання кваліфікаційної роботи 2026

Рік захисту роботи 2026

ABSTRACT

Development of a website for purchasing cars from the USA

The aim of this work is to create a web application for automated data collection on cars from the USA and calculating customs clearance costs.

The tasks of the work are: analysis of the car import market, justification of the technology stack selection (Python, Flask, BeautifulSoup), design of the client-server architecture with web scraping algorithms, development of a responsive interface, and implementation of a customs calculator to assess the economic feasibility of the import.

The subject of the research is the software implementation of the data aggregation application.

The object of the research is web-based information and analytical systems.

The research is based on information aggregator platforms for car advertisements (specifically AUTO.RIA) and the legal framework of Ukraine regarding vehicle customs clearance.

The results of the research include a ready-to-operate software product that optimizes decision-making processes for individuals and car dealers by automating market monitoring and customs payment calculations.

The obtained results can be used by buyers to control costs during car imports, and the applied architectural approaches can be used by IT specialists in the development of similar data aggregation services.

The qualification work consists of 68 pages, 4 tables, 4 images, a list of 52 references.

Keywords: web application, Python, Flask, BeautifulSoup, web scraping, data parsing, customs calculator, car import, USA, AUTO.RIA.

Year of qualification work: 2026

Year of work defense: 2026

ЗМІСТ

ВСТУП	9
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ.....	12
1.1 Аналіз ринку пригону автомобілів із США та огляд існуючих інформаційних рішень.....	12
1.2 Обґрунтування вибору технологій та інструментарію для розробки клієнт-серверного вебзастосунку (Python, Flask, BeautifulSoup).....	17
РОЗДІЛ 2. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ ДЛЯ ПОШУКУ ТА АНАЛІЗУ АВТОМОБІЛІВ ІЗ США	25
2.1 Проєктування архітектури системи, алгоритмів парсингу та нормалізації даних.....	25
2.2 Розробка інтерфейсу користувача, навігації та функціоналу відображення результатів пошуку	36
РОЗДІЛ 3. ОЦІНКА ЕФЕКТИВНОСТІ ВЕБЗАСТОСУНКУ ТА ПЕРСПЕКТИВИ ЙОГО ВДОСКОНАЛЕННЯ.....	55
ВИСНОВКИ.....	59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	62

ВСТУП

Актуальність розробки спеціалізованих інформаційних систем для пошуку та аналізу автомобілів із США зумовлена стійкою тенденцією до зростання обсягів імпорту вживаних транспортних засобів на українському ринку. Незважаючи на складні макроекономічні умови, вторинний авторинок України демонструє надзвичайно високу активність у сегменті автомобілів, придбаних на американських страхових аукціонах. Це пояснюється суттєвою економічною вигодою, адже транспортні засоби із США часто мають значно нижчу початкову вартість порівняно з аналогами на внутрішньому європейському чи українському ринках, прозорішу історію технічного обслуговування, менший пробіг та багатшу комплектацію.

Водночас процес підбору автомобіля, оцінки доцільності його придбання, розрахунку вартості відновлювального ремонту, а також обчислення митних платежів та складних логістичних витрат залишається складною багатофакторною задачею, яка вимагає від кінцевого споживача значних аналітичних зусиль та специфічних знань. Аналіз існуючих на ринку рішень свідчить про те, що доступні для пересічного користувача інструменти є переважно фрагментованими та вузькоспеціалізованими (окремі сайти для перевірки VIN-коду, статичні калькулятори розмитнення тощо). Відсутність єдиної інтегрованої інформаційно-аналітичної платформи створює суттєвий бар'єр для потенційних покупців.

Актуальність теми. Шляхом критичного аналізу існуючих підходів до пошуку та оцінки вартості імпортованих автомобілів обґрунтовано необхідність створення єдиного програмного інструменту. Інтеграція процесів агрегації даних, їх автоматичної нормалізації та динамічного розрахунку фінальної вартості «під ключ» з урахуванням поточного українського законодавства дозволить суттєво оптимізувати процес прийняття економічних рішень.

Стан і ступінь розробки проблеми: Проблематика побудови інформаційних систем, технологій вебскрейпінгу та автоматизації бізнес-процесів розглядалася у працях багатьох вітчизняних (Л. О. Терещенко, О. С. Коваленко, Є. В. Левус, А. А. Дзендзя) та зарубіжних (Д. Хаббард, С. Г. Теттех, Р. Лісандро) вчених. Проте питання розробки спеціалізованих аналітичних вебзастосунків, які б у режимі реального часу поєднували парсинг автомобільних маркетплейсів із розрахунком складних митних і логістичних витрат, залишається недостатньо висвітленим і потребує практичного вирішення.

Мета і завдання дослідження. Метою кваліфікаційної роботи є створення комплексного клієнт-серверного вебзастосунку, здатного повністю автоматизувати процес збору інформації про актуальні автомобілі із США, забезпечити зручний інтерфейс для багатопараметричного пошуку, а також реалізувати аналітичний модуль для точного розрахунку вартості розмитнення й оцінки потенційної фінансової економії.

Для досягнення поставленої мети визначено такі завдання:

- Проаналізувати сучасний стан ринку імпорту автомобілів та оглянути існуючі інформаційні рішення;
- Обґрунтувати вибір оптимального технологічного стека (Python, Flask, BeautifulSoup) для реалізації проєкту;
- Спроектувати модульну клієнт-серверну архітектуру та розробити алгоритми вебскрейпінгу й нормалізації даних;
- Розробити адаптивний інтерфейс користувача та функціонал відображення результатів пошуку;
- Імплементувати обчислювальні алгоритми митного калькулятора для оцінки економічної доцільності імпорту.

Об'єкт дослідження - процес розробки та функціонування інформаційно-аналітичних систем на базі вебтехнологій.

Предмет дослідження - програмна реалізація клієнт-серверного вебзастосунку для пошуку, агрегації та аналізу даних про автомобілі із США.

Методи дослідження. Для вирішення поставлених завдань у роботі використано: теоретичні методи (аналіз та синтез наукової літератури, систематизація нормативно-правової бази) для дослідження предметної області; емпіричні методи (вебскрейпінг) для безперервного збору актуальних даних; методи економіко-математичного моделювання для побудови алгоритмів калькулятора митних платежів та оцінки економічного ефекту.

База досліджень - інформаційні платформи-агрегатори автомобільних оголошень (зокрема маркетплейс AUTO.RIA) та нормативно-правова база України з питань митного оформлення транспортних засобів.

Науково-практична значимість одержаних результатів. Створено готовий до експлуатації програмний продукт, який оптимізує процеси прийняття рішень для фізичних осіб та автодилерів. Впровадження системи дозволяє мінімізувати ризики неправильного розрахунку митних чи прихованих платежів, автоматизує рутинні процеси моніторингу та надає математично обґрунтовану аналітику щодо економічної доцільності імпорту кожної одиниці транспорту.

Особистий внесок здобувача. Усі результати, викладені в кваліфікаційній роботі, отримані автором самостійно. Особистий внесок полягає у проєктуванні архітектури застосунку, написанні програмного коду (Python/Flask), розробці алгоритмів парсингу та імплементації математичної логіки митного калькулятора.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ

1.1 Аналіз ринку пригону автомобілів із США та огляд існуючих інформаційних рішень.

Ринок імпорту вживаних автомобілів із США в Україну є одним із найбільш динамічних сегментів національної економіки, який демонструє не лише кількісне зростання, але й глибокі структурні трансформації протягом останніх кількох років. Аналіз статистичних даних за 2024-2025 роки, опублікованих профільними асоціаціями (зокрема, «Укравтопром» та Інститутом досліджень авторинку), вказує на те, що американський вектор імпорту залишається фундаментальним джерелом поповнення українського автопарку технологічними транспортними засобами[3]. Протягом серпня-жовтня 2024 року українці придбали на американських аукціонах приблизно 17,8 тисячі автомобілів, причому близько 45% з них належали до бюджетного сегмента з ціною до 7 000 доларів США на етапі торгів, тоді як преміальний сегмент склав 15%[2, 25].

У 2025 році ця тенденція не лише збереглася, але й суттєво прискорилося. За офіційними даними, загальний попит на вживані авто із США зріс на 48% порівняно з попереднім роком[24]. У загальній структурі ввезених з-за кордону вживаних легковиків частка «американців» досягла 22%, тоді як у 2024 році цей показник становив 19%, а у 2023 році - лише 13,3%[28]. Лише у жовтні 2025 року автопарк України поповнили понад 5,8 тисячі вживаних автомобілів із США, що у 2,2 рази перевищує показники аналогічного періоду минулого року[28]. Середній вік імпортованих автомобілів знизився з 5,8 року у 2024 році до 5,2 року у 2025-му, що яскраво свідчить про зростання вимог українських споживачів до технічного стану, екологічних стандартів безпеки та інноваційності придбаних транспортних засобів[26].

Найбільш фундаментальною зміною, що визначає розвиток ринку, є його стрімка електрифікація. Якщо на початку 2024 року електромобілі становили

28% від загальної кількості ввезених із США авто (поступаючись автомобілям із двигунами внутрішнього згоряння, які займали 45%), то у 2025 році відбулася радикальна зміна пріоритетів[12]. Електромобілі захопили беззаперечне лідерство, склавши від 49% до 52% всього американського імпорту. Бензинові машини відступили на другу позицію (34-36%), гібриди охопили 7-8%, дизельні авто - 4-5%, а автомобілі з газобалонним обладнанням (ГБО) склали близько 3%[12]. Домінування електромобілів забезпечується передусім маркою Tesla. Серед автомобілів з ДВЗ незмінним лідером залишається Ford Escape (табл. 1.1).

Таблиця 1.1

Рейтинг найпопулярніших моделей автомобілів із США (2025 р.)

Марка та модель автомобіля	Тип двигуна	Кількість імпортованих авто (од.)	Ринкова частка у сегменті топ-5 (%)
Tesla Model Y	Електро	10 342	34.2%
Tesla Model 3	Електро	9 106	30.1%
Ford Escape	Бензин / Гібрид	3 796	12.5%
Tesla Model S	Електро	3 075	10.1%
Chevrolet Bolt	Електро	2 965	9.8%

Джерело: створено автором

Економічна вигода залишається головним драйвером високого попиту. Дослідження показують, що автомобілі із США обходяться кінцевому споживачеві на 20-30% дешевше, ніж аналогічні пропозиції на внутрішньому ринку України[3]. Наприклад, при купівлі Honda Accord 2020 року випуску кінцева вартість в Україні з усіма митними та логістичними витратами становить близько 14 260 доларів США, тоді як ціна на аналог в Україні стартує від 23 000 доларів, що забезпечує економію понад 8 700 доларів. У преміум-сегменті ця різниця є ще більшою: економія при імпорті BMW X5 (2019 року) може сягати 18 600 доларів[3]. Розуміння цієї вигоди стимулює користувачів активно шукати інструменти для самостійного прорахунку витрат.

Незважаючи на високий попит, існуюча інфраструктура інформаційних систем для підбору та аналізу автомобілів залишається фрагментованою та часто

не відповідає потребам кінцевого користувача, який бажає отримати комплексну послугу. Проведений огляд ринку виявляє декілька категорій існуючих платформ, кожна з яких має свої обмеження.

Сервіси перевірки історії лотів (Carfax, BidFax, AutoAstat). Сервіс BidFax є одним із найпопулярніших безкоштовних інструментів, що збирає дані з аукціонів Copart та IAAI[30]. Він дозволяє дізнатися історію продажів, технічний стан на момент покупки та підсумкову ціну лота. Проте його недоліками є спрощена система фільтрації (доступний пошук лише за маркою та моделлю) та затримка в оновленні даних, яка може становити від 2 до 24 годин. Система Carfax надає значно ширшу картину, агрегуючи понад 4 мільярди записів з національних баз США та Канади[32]. Вона дозволяє виявити приховані ДТП, скручені пробіги та історію обслуговування. Однак звіти Carfax є платними, що унеможливорює його використання як інструмента масового пошуку та первинної фільтрації лотів.

Вебсайти логістичних компаній (PLC.ua, W8Shipping). Більшість компаній, що спеціалізуються на імпорті, надають на своїх ресурсах доступ до каталогів авто[22]. Ці платформи часто публікують вже розраховану ціну «під ключ», але алгоритм цього розрахунку залишається прихованим від користувача (так званий «чорний ящик»). Користувач не бачить декомпозиції витрат: яка частина суми йде на сплату акцизу, яка - на маржу самої компанії, а яка - на експедирування. Це обмежує прозорість процесу та знижує довіру.

Локальні маркетплейси (AUTO.RIA). Платформи загального призначення містять тисячі оголошень у категорії «авто під пригін», які розміщуються дилерами. Проте ці маркетплейси не мають вбудованого функціоналу динамічного калькулятора розмитнення, який би автоматично перераховував державні збори при зміні параметрів авто, а також не пропонують аналітики щодо порівняння ціни лота з ринковою ціною в Україні.

Порівняльний аналіз існуючих платформ наведено у таблиці 1.2.

Таблиця 1.2

Огляд існуючих платформ для аналізу та пошуку авто із США

Назва сервісу / платформи	Основне призначення	Модель доступу	Основні недоліки та обмеження
BidFax	Перевірка історії аукціонних продажів	Безкоштовно	Затримка даних до 24 год, відсутність розрахунку митних платежів, слабка фільтрація[30].
Carfax	Повний аналіз історії авто за VIN-кодом	Платно (≈85 грн/звіт)	Фінансова нерентабельність для масового перебору лотів на етапі пошуку[32].
Сайти дилерів	Продаж послуг з пригону авто	Безкоштовно	Прихована структура витрат, неможливість кастомного налаштування логістичних витрат[22].
AUTO.RIA	Агрегатор оголошень про продаж авто	Безкоштовно	Відсутність інструментів фінансової аналітики економії та розрахунку розмитнення.

Джерело: узагальнено автором за даними[30,32,22]

Вищезазначені фактори безпосередньо вказують на гостру необхідність розробки єдиної, консолідованої інформаційної системи. Користувачу потрібен інструмент, який би автоматично забирав актуальні дані з маркетплейсів у режимі реального часу, проводив їх нормалізацію та самостійно розраховував фінансову модель покупки «під ключ» на основі чинного податкового законодавства України. Лише такий комплексний підхід здатний забезпечити прозорість, необхідну для прийняття виважених економічних рішень в умовах високодинамічного ринку.

Зростання попиту на автомобілі із США закономірно призвело до еволюції супутніх інформаційних сервісів, проте більшість із них досі вирішує лише вузькоспрямовані завдання. Огляд аукціонів Америки (Copart, IAAI, Manheim) показує, що хоча вони і надають відкритий доступ до лотів, вони орієнтовані на

B2B-сегмент (дилерів та брокерів) і не містять інструментів для адаптації фінансових показників під податкове законодавство країн-імпортерів[4].

Ключовою проблемою, з якою стикається кінцевий український споживач, є високий рівень інформаційної асиметрії під час розрахунку фінальної вартості транспортного засобу. Згідно з чинними правилами ввезення автомобілів в Україну у 2025 році, процес митного оформлення залишається багатокomпонентним і вимагає обчислення ввізного мита, акцизного податку (який залежить від віку авто, типу двигуна та його об'єму), податку на додану вартість (ПДВ) та збору до Пенсійного фонду[14]. Покрокові інструкції щодо розмитнення демонструють, що навіть незначна помилка у визначенні митної вартості або неправильна класифікація типу силової установки (наприклад, плутанина між «м'яким гібридом» та повноцінним електромобілем) може призвести до непередбачуваних фінансових втрат у розмірі тисяч доларів[11].

Для вирішення цієї проблеми на ринку існують державні та приватні митні калькулятори (наприклад, інтегровані рішення на сайтах митної служби або інформаційних порталах). Проте їх головним недоліком є статичність: користувач змушений вручну переносити технічні характеристики (рік випуску, об'єм двигуна, ціну лота) з вікна аукціону або маркетплейсу у вікно калькулятора. В умовах динамічного вторинного ринку, який у 2024-2025 роках характеризується надзвичайною волатильністю цін та швидким викупом ліквідних лотів[17], такий ручний підхід є неефективним і призводить до втрати часу.

Крім того, локальні маркетплейси стикаються з проблемою неструктурованих даних. Оголошення формуються користувачами у вільному форматі, що створює значний рівень інформаційного шуму. Аналіз попиту підтверджує, що покупцям дедалі частіше потрібні не просто агрегатори оголошень, а інтелектуальні системи підтримки прийняття рішень, які здатні верифікувати дані та надавати предиктивну аналітику щодо економії[13].

Єдиним технологічним шляхом подолання розриву між розрізненими базами даних автомобілів та інструментами фінансового аналізу є застосування

технологій вебскрейпінгу (парсингу). Парсинг даних - це автоматизований процес збору неструктурованої інформації з вебсторінок та її трансформації у структурований формат (бази даних, електронні таблиці або API-інтерфейси) для подальшого аналізу[7]. У контексті автомобільного ринку парсинг дозволяє в режимі реального часу імітувати поведінку користувача, завантажувати HTML-код сторінок-донорів, вилучати з них ціни, технічні параметри та фотографії.

Дослідження технологічних аспектів реалізації вебскрейпінгу доводять, що використання автоматизованих скриптів дозволяє обробляти тисячі оголошень за лічені хвилини, що фізично неможливо зробити вручну[10]. Розробка та впровадження ефективних методів вебскрейпінгу є критично важливою умовою для створення сучасних інформаційних систем, оскільки це усуває необхідність утримання статичних баз даних, які швидко втрачають актуальність[21]. Замість цього система функціонує за принципом «екстракції даних на льоту» (on-the-fly extraction), що гарантує 100% релевантність інформації на момент звернення користувача.

Таким чином, результати проведеного аналізу предметної області дозволяють зробити висновок, що існуючі інформаційні рішення на ринку імпорту автомобілів із США є фрагментованими та не задовольняють повною мірою потреби сучасного споживача в оперативній та прозорій фінансовій аналітиці. Створення єдиного інтегрованого вебзастосунку, який би автоматизував збір даних за допомогою парсингу та самостійно проводив розрахунок митних платежів, є актуальним науково-практичним завданням. Для його реалізації необхідно обґрунтувати вибір сучасного інструментарію розробки, зокрема мови програмування Python та її спеціалізованих фреймворків, що буде розглянуто в наступному підрозділі.

1.2 Обґрунтування вибору технологій та інструментарію для розробки клієнт-серверного вебзастосунку (Python, Flask, BeautifulSoup)

Проектування та розробка сучасного інформаційно-аналітичного вебзастосунку для агрегації, аналізу та нормалізації даних вимагає ретельного і науково обґрунтованого підходу до формування технологічного стека. Оскільки ядром розробленої системи є безперервна мережева комунікація із зовнішніми вебресурсами, синтаксичний аналіз великих масивів HTML-коду в режимі реального часу (Real-Time Scraping) та виконання складних багатокрокових математичних обчислень для калькулятора митних платежів, вибір технологій має ґрунтуватися на жорстких критеріях. До таких критеріїв належать: швидкість обробки I/O (вводу-виводу) операцій, гнучкість архітектури, мінімізація споживання оперативної пам'яті сервером та наявності потужної екосистеми відкритих бібліотек для роботи з неструктурованими даними.

Після аналізу сучасних тенденцій у галузі інженерії програмного забезпечення, базовою мовою програмування для розробки серверної (бекенд) ффчастини застосунку було обрано Python. Порівняно з іншими популярними серверними технологіями (такими як PHP, Node.js або Ruby), Python пропонує унікальний баланс між продуктивністю розробника та обчислювальною потужністю.

Мова Python реалізує мультипарадигменний підхід, повноцінно підтримуючи об'єктно-орієнтований, імперативний та функціональний стилі програмування[20]. Серед її ключових архітектурних переваг науковці виділяють динамічну строгу типізацію, автоматичне управління пам'яттю за допомогою вдосконаленого алгоритму прибиральника сміття (Garbage Collector), а також використання пізнього зв'язування (late binding), що дозволяє легко модифікувати структуру об'єктів безпосередньо під час виконання програми (runtime)[20].

Згідно з сучасними дослідженнями у сфері інформаційних технологій, головною перевагою Python у контексті обробки вебданих є його гігантська екосистема відкритих модулів та лаконічний, математично чистий синтаксис. Це дозволяє розробнику реалізовувати надзвичайно складні логічні конструкції - такі як нормалізація «брудних» текстових даних за допомогою регулярних

виразів (модуль `re`) або багатопотокова обробка десятків мережових HTTP-запитів одночасно (модуль `concurrent.futures`) - з мінімальним обсягом програмного коду, що суттєво знижує ризик виникнення прихованих помилок (багів)[20]. Крім того, інтерпретатор Python є кросплатформним, що забезпечує високий рівень переносимості коду (*portability*) та суттєво спрощує подальше розгортання (*deployment*) застосунку на будь-яких серверах на базі операційних систем сімейства Linux чи Windows.

Визначившись із базовою мовою, наступним критично важливим етапом стало проєктування архітектури системи на рівні вебфреймворку. Для реалізації клієнт-серверної архітектури було проведено глибокий порівняльний аналіз двох найпопулярніших фреймворків екосистеми Python - Django та Flask[36]. Вибір між ними визначає всю подальшу філософію побудови програмного продукту.

Фреймворк Django позиціонується на ринку як потужне монолітне рішення класу «Full-stack» (з філософією «батареї в комплекті» - *batteries-included*). За замовчуванням він включає потужний модуль об'єктно-реляційного відображення (ORM) для роботи з базами даних (PostgreSQL, MySQL тощо), вбудовану систему автентифікації користувачів, готову панель адміністратора, систему захисту від CSRF-атак та строгу, наперед визначену маршрутизацію файлів[31]. Проте, детальний аналіз вимог до розроблюваної системи пошуку автомобілів показав, що для даної дипломної роботи використання Django виявилось б архітектурно надмірним і навіть шкідливим.

Головна причина полягає в моделі даних нашого застосунку. Оскільки розроблена система працює за парадигмою «Екстракція даних на льоту» і принципово не зберігає розпарсені з маркетплейсів оголошення у власній базі даних (це зроблено для гарантування того, що користувач завжди бачить лише актуальні лоти, які ще не продані), наявність важкого шару ORM є непотрібною[31]. Ініціалізація компонентів Django при кожному запиті створювала б невиправдані накладні витрати (*overhead*) оперативної пам'яті та процесорного часу сервера, сповільнюючи віддачу результатів користувачеві.

Натомість, мікрофреймворк Flask надає розробнику абсолютний, низькорівневий контроль над життєвим циклом вебпрограми[36]. Архітектурно Flask не є монолітом; він є тонкою обгорткою, що базується на двох надійних фундаментах: специфікації WSGI (через набір інструментів Werkzeug) для обробки мережевих протоколів та системі шаблонізації Jinja2 для безпечного рендерингу HTML-сторінок[36].

Flask володіє мінімалістичним ядром, яке не нав'язує розробнику жодних конкретних баз даних, систем валідації чи патернів проєктування. Це дозволяє створювати архітектуру «знизу-вгору», підключаючи лише ті бібліотеки, які дійсно необхідні. Індустріальні тренди та наукові публікації свідчать, що для створення спеціалізованих вебсервісів, API-шлюзів та мікросервісних архітектур, де головний фокус зміщений на швидку алгоритмічну обробку даних, а не на управління складними реляційними зв'язками в базі даних, Flask є беззаперечним лідером завдяки своїй неперевершеній гнучкості та швидкодії[31].

Детальне порівняння архітектурних та функціональних особливостей обраних фреймворків, яке стало базою для прийняття інженерного рішення, наведено у таблиці 1.3.

Таблиця 1.3

Розширена порівняльна характеристика вебфреймворків Flask та Django

Критерій оцінки	Flask (обраний фреймворк)	Django (альтернатива)
Архітектурний патерн	Мікрофреймворк (висока модульність, архітектура не нав'язана)	Повностековий моноліт (MVT - Model-View-Template)
Робота з базами даних	ORM відсутня «з коробки» (може інтегруватися через SQLAlchemy за потреби)	Вбудована, жорстко інтегрована та важка ORM
Панель адміністратора	Відсутня (потребує написання з нуля або використання розширень)	Вбудована, генерується автоматично на основі моделей

Продовження табл. 1.3

Маршрутизація (Routing)	Декоратори функцій (наприклад, <code>@app.route('/path')</code>), інтуїтивно зрозуміло	Розділення логіки через окремий конфігураційний файл <code>urls.py</code>
Гнучкість та контроль	Максимальна. Розробник сам обирає всі компоненти системи	Низька. Строгі стандарти структури проєкту (The Django Way)
Швидкість розробки MVP	Дуже висока для вузькоспеціалізованих задач	Висока для складних порталів, але надмірна для простих скриптів
Споживання ресурсів сервера	Мінімальне (легковагий)	Високе (через завантаження великої кількості вбудованого middleware)
Цільове призначення	Скрейпери, мікросервіси, REST API, інструменти обробки даних	Великі корпоративні портали, CMS, масштабні e-commerce системи

Джерело: створено автором

Обґрунтувавши вибір мови Python та мікрофреймворку Flask як серверного фундаменту, ми створили оптимальне середовище для реалізації головного завдання системи - збору інформації із зовнішніх сайтів-агрегаторів. Проте сам фреймворк лише обробляє HTTP-запити від користувача. Для безпосереднього вилучення даних з інших вебресурсів необхідно імплементувати спеціалізовані технології вебскрейпінгу, аналіз яких наведено у наступній частині даного підрозділу.

Отримання інформації із зовнішніх ресурсів (зокрема, з маркетплейсу AUTO.RIA) є найскладнішим інженерним етапом розробки даної системи. Оскільки відкриті API для безкоштовного масового доступу до бази даних автомобілів часто відсутні або мають суворі обмеження (rate limits), єдиним ефективним методом агрегації даних «на льоту» є технологія вебскрейпінгу. Для вирішення цього завдання в екосистемі Python розглядалися три основні технології: Selenium, Scrapy та комбінація бібліотек requests + BeautifulSoup.

Дослідження інструментів автоматизованого збору даних вказують на те, що фреймворк Selenium використовується переважно для автоматизованого тестування та керування реальним браузером (через WebDriver). Він ідеально підходить для глибоко динамічних сайтів (Single Page Applications), де контент рендериться за допомогою JavaScript вже після завантаження сторінки[27]. Однак цей інструмент вимагає значних ресурсів процесора (CPU) та оперативної пам'яті (RAM) для запуску так званих «безголових» (headless) браузерів. З огляду на архітектуру нашого застосунку, де користувач очікує результати розрахунку митних платежів у реальному часі, використання Selenium призвело б до критичного зниження продуктивності (затримки могли б сягати 10-15 секунд на один запит), що є неприйнятним для сучасного вебсервісу[41].

Альтернативний фреймворк Scrapy розроблений для побудови великомасштабних асинхронних павуків (web crawlers), що індексують мільйони сторінок. Його архітектура (побудована на базі асинхронної бібліотеки Twisted) орієнтована на автономну роботу у фоновому режимі з подальшим збереженням даних у бази або файли[39]. Це парадигмально не збігається із синхронним циклом «HTTP-запит - HTTP-відповідь», характерним для Flask-застосунку, де дані потрібні негайно для генерації відповіді конкретному користувачеві.

Отже, науково обґрунтованим і практично доцільним вибором стала зв'язка бібліотеки requests для управління HTTP-сесіями та парсера BeautifulSoup (bs4) для синтаксичного аналізу HTML-документів[5]. Бібліотека requests дозволяє гнучко налаштовувати заголовки запитів (headers), зокрема підміняти User-Agent, що допомагає імітувати поведінку реального користувача та уникати блокувань системами захисту від ботів. Отриманий сирий HTML-код передається у BeautifulSoup, який будує з нього навігаційне DOM-дерево. За допомогою методів find() та find_all(), спираючись на CSS-селектори, система миттєво знаходить потрібні контейнери (картки автомобілів), екстрагуючи з них ціни, технічні характеристики та URL-адреси фотографій[5].

Після успішної екстракції виникає проблема якості інформації. Дані на сайтах-агрегаторах вводяться користувачами у вільному (неструктурованому)

форматі, що створює високий рівень інформаційного шуму. Наприклад, ціна може бути вказана як «10500», «10 500», «10.500 \$» або «10500 USD». Об'єм двигуна може містити літери (наприклад, «2.0 л»), що унеможлиблює виконання математичних операцій для розрахунку акцизу.

Для вирішення цієї проблеми у програмний код було інтегровано модуль регулярних виразів (re). Регулярні вирази забезпечують потужний механізм лексичного аналізу та патерн-матчингу (пошуку за шаблоном)[44]. За їх допомогою реалізовано алгоритми нормалізації:

1. Очищення числових значень від невидимих символів (non-breaking spaces), валютних знаків та їх конвертація у типи int або float.
2. Ідентифікація типу силової установки (пошук підрядків «Бензин», «Дизель», «Електро», «Гібрид» незалежно від регістру тексту). Точність цього кроку є критичною для алгоритму митного калькулятора.
3. Нормалізація URL-адрес зображень (видалення параметрів стиснення з посилань для завантаження фотографій в оригінальній якості).

Ще одним архітектурним викликом стала оптимізація швидкодії. Оскільки процес завантаження вебсторінок через HTTP є класичною I/O-bound (обмеженою вводом-виводом) операцією, послідовний (синхронний) парсинг кількох сторінок з результатами пошуку займав би неприйнятно багато часу. Для розв'язання проблеми продуктивності (забезпечення низького Response Time) в архітектуру системи було імплементовано багатопотокову обробку даних[35].

Використовуючи клас ThreadPoolExecutor із вбудованої бібліотеки concurrent.futures, мережеві запити розпаралелюються. Важливою особливістю мови Python є наявність глобального блокування інтерпретатора (GIL - Global Interpreter Lock), яке перешкоджає паралельному виконанню байт-коду[36]. Проте, при виконанні мережевих запитів (коли потік просто чекає відповіді від сервера AUTO.RIA), інтерпретатор автоматично звільняє GIL. Це дозволяє іншим потокам паралельно надсилати свої запити. Застосування цього підходу дозволило прискорити процес агрегації даних у 3-5 разів, роблячи роботу застосунку динамічною та комфортною для кінцевого користувача. Крім того,

впроваджено механізм локального кешування, що зменшує навантаження на сервери-донори та мінімізує ризик потрапляння IP-адреси сервера у чорні списки.

Візуалізація агрегованих та розрахованих даних (Frontend) здійснена за допомогою стандартизованого стека вебтехнологій: HTML5, CSS3 та JavaScript. Жоден найдосконаліший бекенд не буде успішним без ергономічного інтерфейсу користувача (UI) та продуманого користувацького досвіду (UX).

HTML5 забезпечує семантичну структуру застосунку, розділяючи контент на логічні блоки (header, section, article, footer)[48].

Застосування CSS3 (зокрема, використання сучасних парадигм Flexbox та CSS Grid) гарантує реалізацію адаптивного дизайну (Responsive Web Design). Завдяки медіа-запитам (@media queries), складна таблична інформація з розрахунками розмитнення та галереї зображень коректно масштабуються і перегруповуються на екранах будь-якого розміру (від смартфонів до десктопів)[48].

Мова JavaScript (Vanilla JS) використовується для забезпечення клієнтської інтерактивності без перезавантаження сторінки (AJAX-підхід). Зокрема, JS-скрипти обробляють події взаємодії з багатокomпонентними формами (наприклад, динамічне оновлення списку моделей при виборі конкретної марки авто), а також контролюють стани модальних вікон (Lightbox) для повноекранного перегляду фотографій пошкоджень лота[51].

Інтеграція зазначеного технологічного стека - Python, Flask, requests, BeautifulSoup на стороні сервера та HTML/CSS/JS на стороні клієнта - формує надійний, високопродуктивний програмний фундамент. Відсутність важких абстракцій (типу ORM або монолітних фронтед-фреймворків на кшталт React/Angular) дозволила зосередити всі обчислювальні потужності на швидкому парсингу, комплексній нормалізації тексту та безпомилковому виконанні математичних алгоритмів митного калькулятора, що повністю відповідає меті та завданням кваліфікаційної роботи.

РОЗДІЛ 2. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ ДЛЯ ПОШУКУ ТА АНАЛІЗУ АВТОМОБІЛІВ ІЗ США

2.1 Проєктування архітектури системи, алгоритмів парсингу та нормалізації даних

Перехід від теоретичного обґрунтування до безпосередньої програмної реалізації вимагає побудови чіткої архітектурної моделі вебзастосунку. Розроблений програмний продукт має строго структуровану клієнт-серверну архітектуру (Client-Server Architecture), що забезпечує логічне та фізичне розділення процесів обробки бізнес-логіки (Backend) та їх візуалізації (Frontend).

Взаємодія між цими компонентами відбувається за класичним мережевим протоколом HTTP. Коли користувач вводить параметри бажаного автомобіля (марка, модель, гранична ціна, рік) у форму браузера, клієнтська частина формує параметризований запит та надсилає його на сервер. Сервер, що функціонує під управлінням мікрофреймворку Flask, приймає запит, валідує дані, делегує виконання парсингу відповідним підсистемам, обчислює економічні показники та повертає браузеру згенеровану HTML-сторінку з готовими результатами[15].

Для забезпечення високого рівня супроводжуваності коду (Maintainability), ізоляції помилок та можливості подальшого масштабування, архітектуру бекенду було декомпозовано на п'ять взаємодіючих функціональних модулів:

1. Модуль маршрутизації (Routing Module): Відповідає за прийом та первинну обробку HTTP-запитів (GET/POST). Реалізований за допомогою вбудованих декораторів `@app.route` фреймворку Flask. Він виконує роль головного контролера, який відповідає за ініціалізацію сесій користувача та правильне перенаправлення вхідних даних з вебформи до відповідних логічних функцій обробки.

2. Модуль збору даних (Парсер): Ключовий мережевий компонент системи, реалізований на базі бібліотек `requests` та `BeautifulSoup`. Містить спеціалізовані функції, такі як `fetch_real_lots()`, які відповідають за встановлення

захищених з'єднань із серверами-донорами (наприклад, маркетплейсом AUTO.RIA) та вилучення сирого HTML-коду сторінок-результатів пошуку.

3. Модуль нормалізації: Відповідає за глибокий лексичний аналіз та трансформацію «брудних» текстових даних у суворо структуровані типи (цілі числа, числа з плаваючою крапкою, формені рядки), придатні для подальших математичних операцій. Основна логіка цього модуля інкапсульована у функції `parse_ria_card()`.

4. Модуль економічних розрахунків: Інкапсулює фінансову бізнес-логіку системи. Функція `calc_turnkey()` реалізує складні алгоритми калькулятора митних платежів (акциз, мито, ПДВ) та алгоритм оцінки ринкової різниці (потенційної економії для користувача).

5. Модуль шаблонізації: Використовує вбудований у Flask рушій Jinja2 для інтеграції оброблених даних (що зберігаються у форматі словників чи списків Python) безпосередньо в HTML-розмітку. Він генерує фінальний інтерфейс користувача за допомогою функцій на кшталт `car_card_html()`.

Відмінною та найбільш інноваційною рисою спроектованої архітектури є реалізація механізму збору даних «на льоту» (on-the-fly scraping). Оскільки вторинний ринок автомобілів характеризується надзвичайно високою волатильністю (швидка зміна цін, зняття ліквідних лотів з продажу протягом кількох годин), створення статичної реляційної бази даних призвело б до швидкої деградації релевантності інформації[15]. Натомість розроблена система взагалі не зберігає оголошення локально - вона звертається до першоджерел виключно в момент формування користувацького запиту, гарантуючи абсолютну актуальність згенерованої аналітики.

Базовий алгоритм парсингу працює за наступним детермінованим сценарієм. Спочатку програмно створюється об'єкт сесії `requests.Session()`. До цієї сесії додаються спеціальні HTTP-заголовки (Headers), серед яких критично важливим є актуальний User-Agent. Це дозволяє скрипту імітувати поведінку реального веббраузера та успішно проходити базові перевірки систем захисту від ботів, що встановлені на сайтах-донорах[41].

Далі система формує динамічний URL, вбудовуючи в нього параметри пошуку (марку, рік, ціну), і виконує GET-запит. Отриманий у відповідь HTML-код передається об'єкту BeautifulSoup з використанням вбудованого парсера (наприклад, `html.parser` або `lxml`). Алгоритм сканує побудоване DOM-дерево сторінки за допомогою методу `find_all()`, точно знаходячи всі `div` або `section` контейнери, що відповідають карткам окремих автомобілів на маркетплейсі[5].

Однак екстракція цих «сирих» контейнерів - це лише перший крок. Наступним і набагато складнішим інженерним викликом є вилучення з них конкретних метрик та їх подальша підготовка до обчислень.

Отримання «сирих» HTML-контейнерів із даними автомобілів є лише первинним етапом. Найбільшим інженерним викликом у процесі вебскрейпінгу маркетплейсів є екстракція та жорстка нормалізація неструктурованих даних. Інформація на сайтах-агрегаторах (таких як AUTO.RIA) генерується самими користувачами у вільному форматі, що створює значний рівень інформаційного шуму. Наприклад, заявлена вартість автомобіля може бути записана у різних варіаціях: «\$10 500», «10500\$», «10 500 USD» або «10.500». Аналогічна проблема стосується показників пробігу (милі або кілометри) та об'єму двигуна (наявність зайвих літер, наприклад, «2.0 л»).

Оскільки розроблена система використовує ці дані як змінні для складних математичних обчислень у калькуляторі митних платежів, передача ненормалізованих рядкових типів (`str`) призвела б до критичних програмних помилок (винятків типу `ValueError` або `TypeError`). Для вирішення цієї проблеми розроблено модуль лексичного аналізу, ядром якого є функція `parse_ria_card()`.

Ця функція використовує міць регулярних виразів (Regular Expressions), реалізованих через стандартну бібліотеку Python `re`[44]. Алгоритм нормалізації працює за такими ключовими векторами:

1. Очищення числових значень (Ціна, Пробіг, Об'єм): За допомогою методів `re.sub()` алгоритм вилучає всі нечислові символи, включаючи невидимі пробіли (`non-breaking spaces`, `\xa0`), коми, крапки та символи валют. Після

«очищення» текстового рядка застосовується явне приведення типів (Type Casting) до `int` (для цілих чисел) або `float` (для чисел із плаваючою крапкою).

2. Ідентифікація типу силової установки: Оскільки від типу пального кардинально залежить формула нарахування акцизного податку, алгоритм проводить пошук ключових підрядків («Бензин», «Дизель», «Електро», «Гібрид») у блоці технічного опису лота без урахування регістру (використовуючи прапорець `re.IGNORECASE`). Якщо ідентифіковано маркер «Електро», система автоматично маркує цей об'єкт для застосування нульової ставки ввізного мита та ПДВ під час подальших розрахунків.

3. Вилучення ідентифікаторів (VIN-код): Алгоритм сканує картку на наявність 17-значного унікального коду транспортного засобу. Вилучений VIN-код нормалізується і передається у фронтенд-частину застосунку, щоб надати користувачеві можливість миттєво скопіювати його для глибокої перевірки історії авто (наприклад, через бази Carfax).

4. Нормалізація медіа-контенту: Вилучені URL-адреси фотографій часто містять GET-параметри стиснення (наприклад, `?size=small`), які генеруються сервером-донором для економії трафіку. Алгоритм відтинає ці параметри або замінює префікси в URL-рядках, щоб застосунок міг завантажити зображення в максимальній (оригінальній) роздільній здатності для детального огляду пошкоджень у галереї застосунку.

Вирішивши проблему якості даних, розробник стикається з проблемою продуктивності. Процес встановлення з'єднання, завантаження вебсторінок та їх парсингу є класичною I/O-bound операцією (операцією, обмеженою швидкістю вводу-виводу мережі, а не потужністю процесора). При послідовній (синхронній) архітектурі обробка, наприклад, 5 сторінок результатів пошуку (де кожна сторінка містить по 20-30 автомобілів) займала б від 10 до 20 секунд. Такий високий показник Response Time (часу відповіді сервера) є критично неприйнятним для сучасних вебсервісів, оскільки призводить до відмови користувачів від використання платформи.

Для подолання цього обмеження та забезпечення високої швидкодії інтерфейсу, в архітектуру системи було імплементовано механізм багатопотокової обробки даних (Multithreading). Реалізація цього підходу виконана за допомогою високорівневого інтерфейсу `ThreadPoolExecutor` з вбудованого модуля `concurrent.futures`, який є офіційно стандартизованим інструментом в архітектурній специфікації мови Python[16],[40].

Замість того, щоб чекати завантаження кожної сторінки по черзі, алгоритм ініціює пул (набір) робочих потоків (worker threads). Метод `executor.submit()` розподіляє завдання (мережеві HTTP-запити до різних сторінок пагінації AUTO.RIA) між доступними потоками, які починають виконуватися паралельно.

Важливою архітектурною особливістю мови Python (в реалізації CPython) є наявність механізму глобального блокування інтерпретатора - GIL (Global Interpreter Lock). GIL фізично перешкоджає паралельному виконанню байт-коду Python у кількох потоках одночасно (запобігаючи стану гонитви за пам'яттю). Однак, специфіка вебскрейпінгу дозволяє ефективно обійти це обмеження. Під час виконання мережевого запиту (через бібліотеку `requests`), коли потік переходить у стан очікування (блокування) відповіді від зовнішнього сервера, інтерпретатор CPython автоматично звільняє GIL. Це дозволяє операційній системі перемикнути контекст і надати процесорний час іншому потоку для ініціалізації наступного запиту.

Збір та обробка результатів від потоків відбувається асинхронно за допомогою генератора `concurrent.futures.as_completed()`. Практичні тестування розробленої системи показали, що імплементация пулу з 5-10 робочих потоків дозволила прискорити процес агрегації даних у 3-5 разів порівняно із синхронним підходом.

Додатковим інструментом оптимізації серверних ресурсів стало впровадження базового рівня кешування (Caching). Кеш-пам'ять зберігає результати обробки найпопулярніших пошукових запитів (наприклад, «Tesla Model 3, 2020-2022 рік») на визначений короткий проміжок часу (Time-To-Live, TTL). Якщо інший користувач робить ідентичний запит протягом дії TTL, сервер

миттєво віддає готову HTML-сторінку з кешу, повністю оминаючи етап парсингу та розрахунків. Це не лише знижує навантаження на процесор сервера розробленої системи, але й мінімізує кількість запитів до серверів-донорів, запобігаючи потраплянню IP-адреси застосунку до «чорних списків» (банів) за підозрілу активність.

Поглиблюючи архітектурний аналіз модуля збору даних, необхідно окремо виділити механізми забезпечення відмовостійкості (Fault Tolerance) та обходу систем протидії автоматизованому збору інформації (Anti-Scraping Protection). Сучасні маркетплейси, такі як AUTO.RIA, використовують складні алгоритми поведінкового аналізу (наприклад, Cloudflare Bot Management), які блокують запити від скриптів, якщо вони не мають ознак реального браузера[27].

Для мінімізації ризику блокування (HTTP Error 403 Forbidden), у розробленому модулі парсингу застосовано комплексний підхід до формування заголовків запиту (HTTP Headers). Окрім динамічної підміни User-Agent, алгоритм генерує розширений набір заголовків, що включає Accept, Accept-Language, Accept-Encoding (з підтримкою gzip, deflate), а також заголовки Sec-Fetch-Dest та Sec-Fetch-Mode. Така деталізована імітація дозволяє серверу-донору ідентифікувати мережевий запит бібліотеки requests як стандартний перехід звичайного користувача[10].

Важливим етапом програмної реалізації є стійка обробка виняткових ситуацій (Exception Handling). Оскільки вебскрейпінг безпосередньо залежить від стабільності інтернет-з'єднання та працездатності стороннього сервера, будь-яка затримка у відповіді може призвести до фатальної помилки (Crash) всього бекенд-застосунку. Для запобігання цьому, кожен HTTP-запит у багатопотоковому пулі обгорнуто у строгі конструкції try...except.

Зокрема, програмний код перехоплює ключові винятки базового класу `requests.exceptions.RequestException`:

1. Timeout - якщо сервер-донор не відповідає протягом заданого ліміту (наприклад, 5 секунд), потік примусово перериває очікування, щоб не блокувати загальний цикл виконання. У такому разі застосовується патерн «Exponential

Backoff» (експоненційне затримування) - система робить повторну спробу через більший проміжок часу.

2. `ConnectionError` - виникає при розриві TCP-з'єднання або проблемах з DNS. Програма логує цю помилку, продовжуючи обробку інших сторінок.

3. `HTTPError` - обробляє статуси 4xx та 5xx. Якщо отримано код 429 (Too Many Requests), система ініціює тимчасове призупинення потоків (метод `time.sleep()`), імітуючи паузу в активності користувача[29].

Після успішного проходження мережових етапів та нормалізації текстових даних (очищення цін, вилучення VIN-кодів), виникає завдання ефективної структуризації обробленої інформації для її передачі на клієнтську частину. Усі розпарсені та математично обчислені дані (акциз, мито, ринкова ціна) упаковуються у складні структури даних мови Python - вкладені словники (`dict`) та списки (`list`). Ця структура формує об'єктно-орієнтовану модель конкретного лота в оперативній пам'яті сервера[20].

Для асинхронної передачі цих даних до фронтенду (у випадку використання AJAX-запитів) словники Python проходять процес серіалізації - перетворення у формат JSON (JavaScript Object Notation). Вбудований у Flask модуль `jsonify` автоматично встановлює правильні MIME-типи у заголовках відповіді (`application/json`) та забезпечує швидке і безпечне кодування даних. Такий стандартизований підхід робить бекенд застосунку незалежним від візуальної частини, створюючи передумови для легкого створення у майбутньому мобільного додатка або відкритого API, які зможуть споживати ці самі JSON-дані без зміни логіки парсингу[20].

Практична реалізація бекенду вимагала створення оптимізованих структур даних та захисту від мережових перевантажень. Для швидкої генерації зв'язаних списків на клієнтській стороні, базу автомобілів було жорстко структуровано у вигляді словника `BRANDS_MODELS`, що містить ключові марки та відповідні їм моделі (Лістинг 2.1). Це усунуло необхідність постійних запитів до бази даних і прискорило рендеринг форм.

Лістинг 2.1 - Структуризація даних та налаштування HTTP-сесій

```

BRANDS_MODELS = {'BMW':['1 Series',
                        '2 Series',
                        '3 Series',
                        '4 Series',
                        '5 Series',
                        '7 Series',
                        'X1',
                        'X3',
                        'X4',
                        'X5',
                        'X6',
                        'X7',
                        'M3',
                        'M5'],
                  'Audi':['A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'Q3', 'Q5', 'Q7', 'Q8', 'e-tron'],
                  'Mercedes-Benz':['A-Class',
                                    'C-Class',
                                    'CLA',
                                    'CLS',
                                    'E-Class',
                                    'S-Class',
                                    'GLA',
                                    'GLC',
                                    'GLE',
                                    'GLS',
                                    'Vito',
                                    'Sprinter'],
                  'Toyota':['Camry', 'Corolla', 'Avalon', 'RAV4', 'Highlander', 'Land Cruiser',
                             'Prius', 'C-HR'],
                  'Volkswagen':['Passat', 'Jetta', 'Golf', 'Tiguan', 'Touareg', 'Atlas', 'Arteon'],

```

```

'Ford': ['Escape', 'Fusion', 'Explorer', 'Mustang', 'Edge', 'F-150', 'Focus'],
'Honda': ['Accord', 'Civic', 'CR-V', 'HR-V', 'Pilot', 'Odyssey'],
'Tesla': ['Model 3', 'Model S', 'Model X', 'Model Y'],
'Nissan': ['Rogue', 'Altima', 'Leaf', 'Murano', 'Pathfinder', 'Sentra', 'X-Trail'],
'Lexus': ['RX', 'NX', 'ES', 'GX', 'LX', 'UX', 'IS'],
'Hyundai': ['Elantra', 'Sonata', 'Tucson', 'Santa Fe', 'Palisade', 'Kona'],
'Kia': ['Sportage', 'Sorento', 'Optima', 'K5', 'Telluride', 'Soul', 'Stinger'],
'Mazda': ['Mazda 3', 'Mazda 6', 'CX-3', 'CX-5', 'CX-9', 'CX-30'],
'Chevrolet': ['Malibu', 'Cruze', 'Equinox', 'Traverse', 'Tahoe', 'Bolt', 'Camaro'],
'Jeep': ['Compass', 'Cherokee', 'Grand Cherokee', 'Wrangler', 'Renegade'],
'Subaru': ['Impreza', 'Legacy', 'Outback', 'Forester', 'Crosstrek', 'Ascent'],
'Volvo': ['S60', 'S90', 'XC40', 'XC60', 'XC90', 'V60'],
'Porsche': ['Macan', 'Cayenne', 'Panamera', 'Taycan'],
'Land Rover': ['Range Rover Evoque', 'Range Rover Sport', 'Discovery',
'Defender'],
'Skoda': ['Octavia', 'Superb', 'Kodiaq', 'Karoq'],
'Peugeot': ['208', '308', '3008', '5008', '508'],
'Renault': ['Megane', 'Talisman', 'Kadjar', 'Koleos', 'Duster'],
'Opel': ['Astra', 'Insignia', 'Mokka', 'Grandland'],
'Infiniti': ['Q50', 'Q60', 'QX50', 'QX60', 'QX80'],
'Mitsubishi': ['Lancer', 'Outlander', 'ASX', 'Pajero Sport']]
YEARS = [""] + [str(year) for year in range(2005, 2027)]
DAMAGE_OPTIONS = [
    ("", "Будь-яке"),
    ("front", "Передня частина"),
    ("rear", "Задня частина"),
    ("side", "Бічна частина"),
    ("water", "Пошкодження водою"),
    ("hail", "Пошкодження градом"),
    ("mechanical", "Механічні пошкодження"),

```

```

("theft", "Викрадення"),
("normal", "Звичайний знос"),
]

```

```

AUTO_RIA_IMPORT_URL = "https://auto.ria.com/uk/prigon-
avto/country/usa/"

```

```

AUTO_RIA_API_KEY =
"Z7qE9T9q05okq3x9VyA86fAbWYp00Q9x19tw8w2O"

```

```

REQUEST_TIMEOUT = 25

```

```

USER_AGENT = (

```

```

    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) "

```

```

    "AppleWebKit/537.36 (KHTML, like Gecko) Chrome/122.0.0.0
Safari/537.36"

```

```

)

```

```

HTTP = requests.Session()

```

```

_retry = Retry(

```

```

    total=2,

```

```

    backoff_factor=0.35,

```

```

    status_forcelist=[429, 500, 502, 503, 504],

```

```

    allowed_methods=["GET"],

```

```

)

```

```

_adapter = HTTPAdapter(pool_connections=30, pool_maxsize=30,
max_retries=_retry)

```

```

HTTP.mount("http://", _adapter)

```

```

HTTP.mount("https://", _adapter)

```

```

HTTP.headers.update({"User-Agent": USER_AGENT, "Accept-Language":
"uk-UA,uk;q=0.9,en;q=0.8"})

```

Окремим практичним здобутком є розробка власної потокобезпечної системи кешування (TTL Cache) на базі об'єкта `threading.Lock`. Оскільки парсинг сайту-донора вимагає часу, функції `cached_get_json` та `cached_get_text` зберігають результати успішних HTTP-запитів у оперативній пам'яті сервера на 15 хвилин (900 секунд). Реалізацію цього механізму наведено у Лістингу 2.2.

Лістинг 2.2 - Програмна реалізація потокобезпечного TTL-кешування

```
_cache_lock = threading.Lock()
```

```
TTL_CACHE = {}
```

```
def ttl_get(key):
```

```
    now = time.time()
```

```
    with _cache_lock:
```

```
        payload = TTL_CACHE.get(key)
```

```
        if not payload:
```

```
            return None
```

```
        expires_at, value = payload
```

```
        if expires_at < now:
```

```
            TTL_CACHE.pop(key, None)
```

```
            return None
```

```
        return value
```

```
def ttl_set(key, value, ttl=900):
```

```
    with _cache_lock:
```

```
        TTL_CACHE[key] = (time.time() + ttl, value)
```

```
    return value
```

```
def cached_get_json(url, params=None, ttl=900):
```

```
    cache_key = ("json", url, tuple(sorted((params or {}).items())))
```

```
    cached = ttl_get(cache_key)
```

```
    if cached is not None:
```

```

        return cached

    response = HTTP.get(url, params=params, timeout=REQUEST_TIMEOUT)
    response.raise_for_status()
    return ttl_set(cache_key, response.json(), ttl)

def cached_get_text(url, ttl=900):
    cache_key = ("text", url)
    cached = ttl_get(cache_key)
    if cached is not None:
        return cached

    response = HTTP.get(url, timeout=REQUEST_TIMEOUT)
    response.raise_for_status()
    response.encoding = response.apparent_encoding or response.encoding
    return ttl_set(cache_key, response.text, ttl)

```

2.2 Розробка інтерфейсу користувача, навігації та функціоналу відображення результатів пошуку

Успіх будь-якої інформаційної системи на споживчому ринку безпосередньо залежить від ергономічності її користувацького інтерфейсу (User Interface, UI) та продуманого користувацького досвіду (User Experience, UX). Інтерфейс розробленого вебзастосунку було спроектовано з урахуванням сучасних парадигм вебдизайну та принципів людино-машинної взаємодії (HCI). Головним орієнтиром під час проєктування була мінімізація когнітивного навантаження користувача під час аналізу великих масивів числових та технічних даних (цін, податків, характеристик автомобілів)[52].

Основою візуальної частини (Frontend) є використання стандартизованих мов розмітки гіпертексту HTML5 та каскадних таблиць стилів CSS3[50]. Враховуючи тенденції сучасного інтернет-трафіку, де значна частка запитів надходить з мобільних пристроїв, критичною вимогою до інтерфейсу була його

кроссплатформність. Для забезпечення ідеального відображення застосунку на різних типах пристроїв (від смартфонів і планшетів до широкоформатних 4K моніторів) застосовано методологію адаптивного вебдизайну (Responsive Web Design).

Цей підхід реалізовано на базі сучасних інструментів компоновання - гнучких сіток CSS Flexbox (Flexible Box Layout) та CSS Grid[49]. Використання медіа-запитів (@media queries) гарантує, що всі елементи управління, навігаційні панелі та інформаційні блоки автоматично масштабуються, змінюють пропорції та перегруповуються залежно від ширини екрана (viewport) пристрою користувача. Наприклад, на десктопних версіях фільтри пошуку розташовуються зліва від результатів, тоді як на мобільних пристроях вони трансформуються у компактне випадаюче меню.

Навігаційна структура системи спроектована за принципом плоскої архітектури (Flat Architecture), що дозволяє користувачеві дістатися до будь-якого розділу сайту не більше ніж за два кліки. Навігація реалізована через закріплене (sticky) верхнє меню (Header). Використання CSS-властивості position: sticky дозволяє меню залишатися у полі зору користувача під час прокручування довгих списків автомобілів, надаючи миттєвий доступ до основних функціональних вузлів застосунку: «Головна сторінка», «Пошук авто», «Калькулятор вартості», «Етапи пригону» та «Контакти»[50].

Головна сторінка (Home Page) вебзастосунку спроектована як презентаційно-інформаційний хаб (Landing Hub). Вона зустрічає користувача великим типографічним заголовком (Hero Section), лаконічним поясненням ключових переваг сервісу (прозорість розрахунків, економія коштів) та блоком експрес-пошуку. Нижче розташована динамічно згенерована секція з прикладами найпопулярніших марок автомобілів, що імпортуються із США, яка виконує роль «швидкого старту» для нових відвідувачів.

Серцем клієнтської частини та головним інструментом взаємодії користувача із серверною логікою (Бекендом) є багатокomпонентна форма пошуку. Під час проектування цієї форми особливу увагу було приділено

інтерактивному дизайну (Interaction Design) з метою уникнення логічних помилок при введенні параметрів[51].

У традиційних статичних формах пошуку користувач може обрати марку (наприклад, BMW), а в наступному полі «Модель» йому доводиться шукати потрібне значення серед довгого списку сотень варіантів для всіх існуючих брендів (включаючи моделі Audi, Ford тощо). Це не лише погіршує UX, але й створює ризик відправлення на сервер некоректного запиту (наприклад, пошук «BMW Mustang»).

Для вирішення цієї проблеми у розробленому застосунку реалізовано механізм динамічних зв'язаних списків (Dependent Dropdowns), логіка яких написана мовою JavaScript (Vanilla JS)[51]. Алгоритм роботи цього механізму наступний:

1. За допомогою методів DOM API (зокрема, `document.getElementById` або `querySelector`) JavaScript-код «слухає» стан першого випадаючого списку (селектора марок).
2. При виборі користувачем певної марки спрацьовує асинхронна подія `onChange`.
3. Скрипт миттєво перехоплює цю подію і, використовуючи попередньо завантажений JSON-об'єкт із структурованим деревом «Марка - Моделі» (або виконуючи фоновий AJAX-запит до сервера Flask), динамічно очищає DOM-структуру другого випадаючого списку (селектора моделей).
4. У очищений список ін'єктуються (через створення нових елементів `<option>`) виключно ті моделі, які семантично належать обраному бренду.

Реалізація такого клієнтського скрипта значно прискорює процес налаштування фільтрів, знімає зайве навантаження з сервера (оскільки некоректні запити відсікаються ще у браузері) та робить процес вибору безпомилковим. Окрім базових характеристик бренду та моделі, форма має розширені інструменти фільтрації, що дозволяють вказати рік випуску (у вигляді діапазону «від» та «до»), максимальну аукціонну ціну, обмеження по пробігу та допустимий рівень пошкоджень (від «Run and Drive» до значних дефектів). Після

валідації всіх введених полів JavaScript-модуль формує GET-запит і передає його на маршрутизатор Flask-сервера, який ініціює процес парсингу.

Для наочної демонстрації реалізованих архітектурних та дизайнерських рішень нижче наведено візуалізацію ключових екранів розробленого вебзастосунку.

На рисунку 2.1 представлено головну сторінку (Landing Page) системи, яка зустрічає користувача.

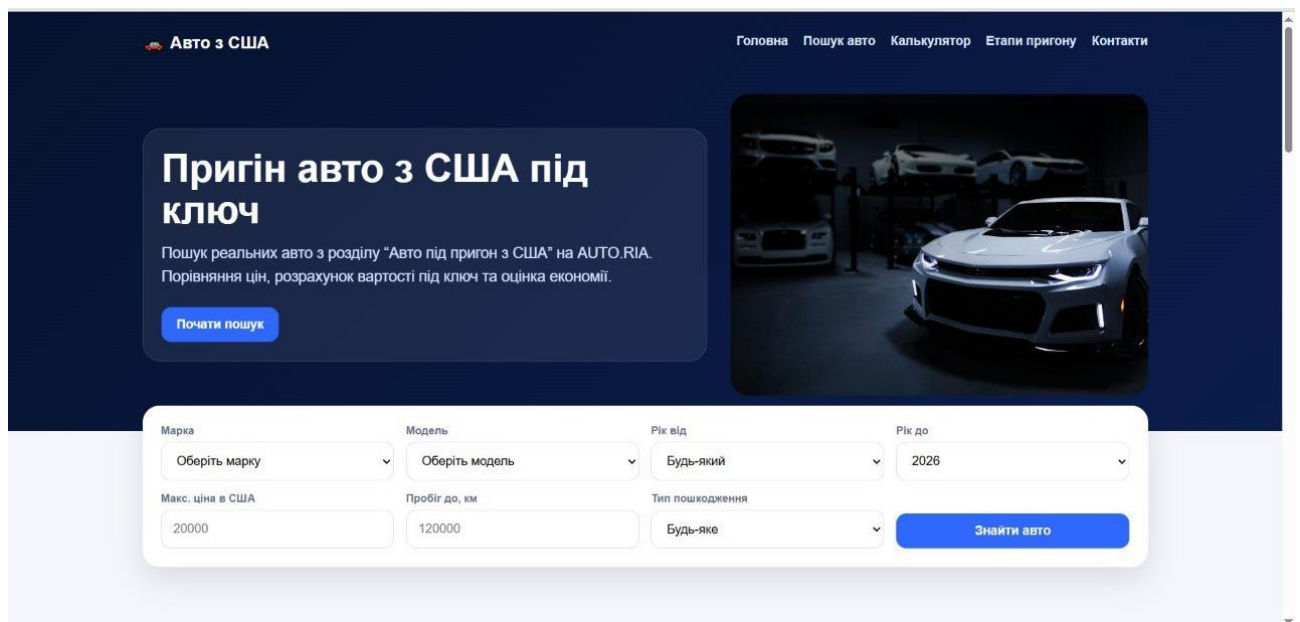


Рис. 2.1. Інтерфейс головної сторінки вебзастосунку з інтерактивною формою пошуку.

Як видно з рис. 2.1, інтерфейс побудовано за принципом мінімалізму з акцентом на головну цільову дію (Call-to-Action). Верхня навігаційна панель (Header) містить логотип та посилання на ключові розділи, забезпечуючи швидку просторову орієнтацію. Блок Hero Section містить чітку ціннісну пропозицію («Пригін авто з США під ключ») та текстовий дескриптор, що пояснює алгоритм роботи сервісу. Центральним елементом взаємодії є багатопараметрична форма пошуку. Вона спроєктована у вигляді компактного горизонтального блоку на світлому фоні, що контрастує з темним тлом сторінки. Форма включає селектори марки, моделі (які працюють за принципом динамічних зв'язаних списків),

діапазону років випуску, граничної ціни, пробігу та типу пошкодження. Таке компонування дозволяє користувачеві налаштувати складний пошуковий фільтр без необхідності переходу на інші сторінки.

Після ініціалізації пошукового запиту система перенаправляє користувача на сторінку результатів (рис. 2.2).

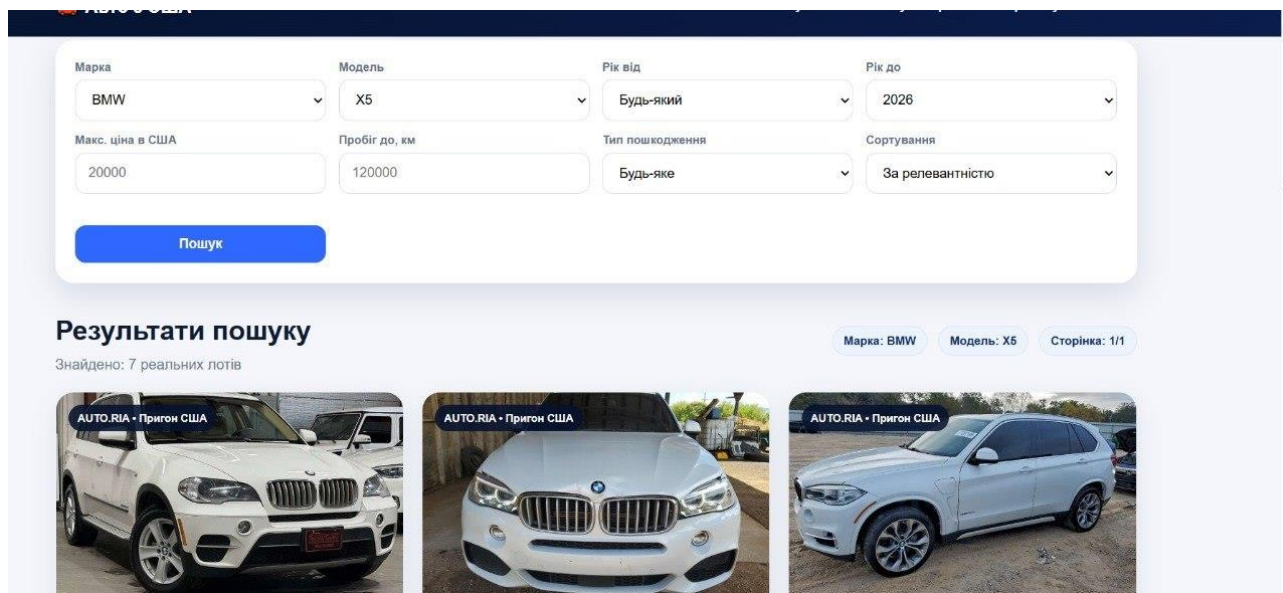


Рис. 2.2. Візуалізація результатів пошукового запиту (на прикладі вибірки автомобілів BMW X5)

Важливою UX-особливістю сторінки результатів є збереження панелі фільтрів у верхній частині екрана зі збереженими параметрами поточного запиту. Це дозволяє користувачеві «на льоту» коригувати критерії (наприклад, змінити рік з 2026 на 2024 або відредагувати максимальну ціну) без повернення на головну сторінку. Під формою реалізовано блок метаданих запиту, який виводить лічильник знайдених лотів («Знайдено: 7 реальних лотів») та хлібні крихти (Breadcrumbs) із зазначенням активної сторінки пагінації. Нижче починається вивід безпосередньо масиву даних у вигляді сітки карток (Grid Layout). На кожному зображенні автомобіля автоматично генерується інформаційний бейдж (наприклад, «AUTO.RIA - Пригон США»), що ідентифікує джерело походження даних та підвищує рівень довіри до платформи. Після успішної валідації пошукового запиту та обробки даних на сервері, ключовим завданням фронтенду стає ефективна візуалізація отриманого масиву

інформації. Відображення результатів пошуку в розробленому застосунку базується на методології карткового дизайну (Card-based UI Design). Згідно з дослідженнями у сфері людино-машинної взаємодії, картковий інтерфейс є оптимальним патерном для агрегаторів, оскільки він дозволяє «дробити» (chunking) великі обсяги гетерогенної інформації на незалежні, легко засвоювані візуальні блоки[40].

Згенерована HTML-шаблонізатором (Jinja2) картка лота є високощільним інформаційним концентратом. Вона структурно поєднує: медіа-зону (головну фотографію автомобіля), типографічний заголовок (марка та модель), табличну сітку ключових технічних ідентифікаторів (рік випуску, об'єм двигуна, тип пального, тип коробки передач, локацію на території США) та вилучений VIN-код.

Найбільшу інформаційну та аналітичну цінність становить структура самої картки лота, детальний вигляд якої представлено на рисунку 2.3.

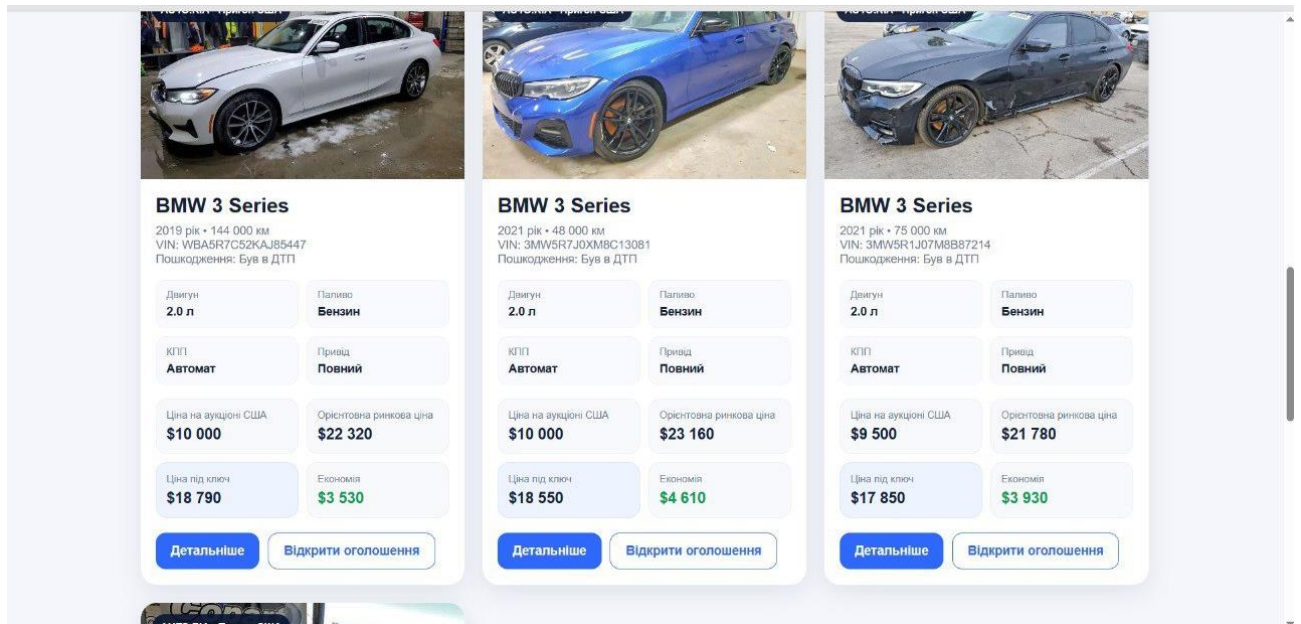


Рис. 2.3. Структура інформаційної картки автомобіля з деталізованим фінансовим блоком та семантичним кодуванням

На рис. 2.3 наочно продемонстровано реалізацію методології карткового дизайну (Card-based UI). Картка лота має чітку візуальну ієрархію:

1. Ідентифікаційний блок: Містить великий типографічний заголовок (наприклад, «BMW 3 Series»), рік випуску, пробіг та вилучений парсером унікальний VIN-код. Окремо виведено статус пошкодження («Був в ДТП»).

2. Технічний блок: Спроектований у вигляді двоколонкової сітки з використанням карток-плашок зі світло-сірим фоном. Тут агреговано ключові метрики: об'єм двигуна (нормалізований до формату «2.0 л»), тип палива, тип коробки передач та привід. Таке «дроблення» (chunking) інформації значно знижує когнітивне навантаження при швидкому скануванні сторінки.

3. Фінансовий аналітичний блок: Є ключовим елементом системи прийняття рішень. Він паралельно демонструє чотири розраховані показники: ціну на аукціоні в США, орієнтовну ринкову ціну в Україні, ціну «під ключ» (результат роботи бекенд-калькулятора митних платежів) та суму економії.

4. Блок навігації: Включає дві кнопки-дії. Основна (Primary Button) - «Детальніше» (виділена синім кольором) для переходу на внутрішню сторінку глибокої аналітики лота. Другорядна (Secondary Button) - «Відкрити оголошення» (з контурним дизайном) для прямого переходу на першоджерело маркетплейсу.

Проте, оскільки головна мета розробленої системи - глибокий економічний аналіз, особливий концептуальний акцент у дизайні карток зроблено на фінансовому блоці. Для полегшення сприйняття складних розрахунків застосовано метод семантичного кольорового кодування (Color Coding). Користувач одночасно бачить декілька цінових категорій: початкову ставку лота на аукціоні, повністю розраховану ціну «під ключ» в Україні (включаючи всі митні платежі) та середню ринкову ціну аналогічного авто на українських майданчиках. Обчислена сума потенційної економії виділяється візуально за допомогою кольорових блоків (badges). Позитивна економічна різниця (економія) підсвічується контрастним зеленим кольором. Такий підхід створює потужний психологічний стимул (Call-to-Action) до подальшого дослідження лота, миттєво фокусуючи увагу користувача на найбільш рентабельних пропозиціях[33].

Підсумовуючи, можна стверджувати, що розроблений комплексний інтерфейс успішно виконує функцію «перекладача» між складним масивом необроблених даних (отриманих алгоритмами бекенду) та кінцевим користувачем. Завдяки застосуванню сучасних підходів UI/UX дизайну, система перетворюється з простого парсера на інтуїтивно зрозумілий інструмент фінансової аналітики автомобільного ринку.

Як зазначалося раніше, застосування семантичного кольорового кодування (зелений колір тексту для поля «Економія», наприклад, \$3 530, \$4 610) спрацьовує як потужний візуальний тригер. Це дозволяє користувачеві під час скролінгу довгого списку інтуїтивно та миттєво ідентифікувати найбільш рентабельні пропозиції, повністю виправдовуючи мету створення даного програмного продукту.

Для глибшого занурення в характеристики конкретного транспортного засобу архітектурою передбачено перехід на окрему «Детальну сторінку лота». Вона проєктувалася за принципом аналітичного дашборда: ліворуч розміщено інтерактивний медіа-блок, а праворуч - розгорнута панель технічних характеристик із детальною декомпозицією всіх фінансових витрат (від аукціонного збору до відрахувань у Пенсійний фонд).

Реалізація медіа-блоку (інтерактивної галереї зображень) є одним із вагомих технічних досягнень клієнтської частини. Для забезпечення максимальної швидкодії галерею було розроблено виключно на базі нативних HTML, CSS та JavaScript, без використання сторонніх «важких» бібліотек (наприклад, jQuery плагінів), які уповільнюють рендеринг сторінки[34]. Галерея включає велику зону перегляду головного фото та стрічку навігаційних мініатюр (thumbnails). Логіка JavaScript відстежує події кліку (або дотику на мобільних пристроях) по мініатюрах, динамічно та безшовно змінюючи атрибут src головного зображення в об'єктній моделі документа (DOM).

Оскільки автомобілі із США продаються на страхових аукціонах (Copart, IAAI), детальний огляд пошкоджень є критичним етапом для оцінки вартості майбутнього відновлювального ремонту. Для вирішення цього завдання

імплементовано функцію повноекранного перегляду за патерном модального вікна (Lightbox). При кліку на фотографію вона відкривається поверх основного контенту із затемненим фоном, фокусуючи увагу користувача виключно на зображенні. У модальне вікно інтегровано елементи управління навігацією (кнопки «вперед/назад») та додано обробку подій touchstart і touchend, що забезпечує повноцінну підтримку жестів гортання (свайпів) для користувачів із сенсорними екранами мобільних пристроїв[34].

Окрім суто технічного інструментарію, користувацький інтерфейс застосунку виконує важливу освітню та маркетингову функції. З цією метою було спроектовано окремий інформаційний розділ «Етапи пригону». Він розроблений у вигляді вертикального інтерактивного таймлайну (часової шкали) з використанням CSS-псевдоелементів (::before, ::after) та векторних піктограм (SVG)[46].

Логічним доповненням користувацького інтерфейсу є програмна реалізація інформаційного розділу «Контакти». Він не лише надає користувачам вичерпну інформацію для зв'язку, але й структурує перелік інтегрованих послуг, які надає платформа (від пошуку лотів на аукціонах до розмитнення). Програмний код відповідного маршруту демонструє використання методів генерації сторінок засобами Flask (Лістинг 2.3).

Лістинг 2.3 - Програмна реалізація маршруту сторінки «Контакти»

```
@app.route("/contacts")
def contacts():
    html = ""
    <main>
        <section class="section">
            <div class="two-grid">
                <div class="panel">
                    <h2 style="margin-top:0;">Контакти</h2>
                    <p class="lead">Зв'яжіться з нами для підбору авто та
консультації щодо вартості пригону.</p>
```

```

<ul class="list-clean">
    <li>Телефон: +38 (095) 940-38-81</li>
    <li>Email: info@autousa.local</li>
    <li>Telegram: @autousa_manager</li>
    <li>Instagram: @auto_usa_project</li>
    <li>Адреса: м. Ірпінь, вул. Григорія Сковороди, 27</li>
</ul>
</div>
<div class="panel">
    <h2 style="margin-top:0;">Що входить у послугу</h2>
    <ul class="list-clean">
        <li>Пошук лотів на Copart, IAAI та Manheim</li>
        <li>Перевірка історії автомобіля</li>
        <li>Розрахунок бюджету під ключ</li>
        <li>Організація доставки в Україну</li>
        <li>Допомога з розмитненням і оформленням</li>
    </ul>
</div>
</div>
</section>
</main>
"""

return render_page(html, "Контакти")

```

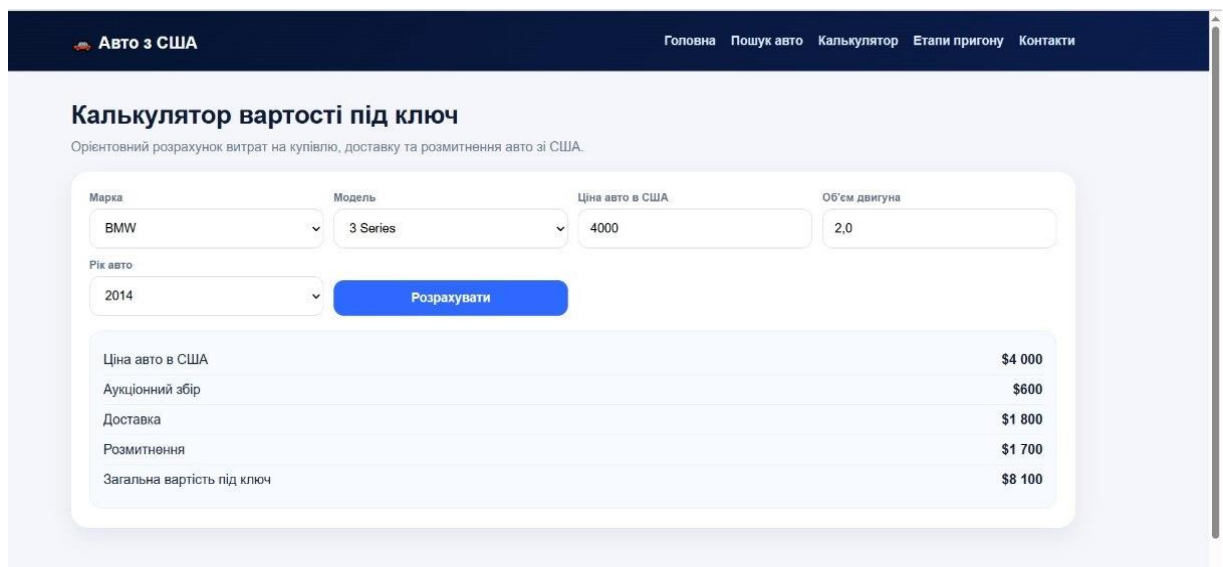
Важливим практичним модулем системи є маршрут `/calculator` (автономний калькулятор). Він обробляє POST-запити з форми, отримуючи параметри автомобіля (марка, ціна, об'єм двигуна, рік випуску).

Важливим аспектом забезпечення якості є перевірка безпеки користувацького вводу (Input Validation & Sanitization). У модулі автономного калькулятора, де користувач власноруч вводить технічні параметри автомобіля, імплементовано сувору серверну валідацію. Навіть якщо зловмисник спробує

обійти клієнтські HTML5-обмеження (наприклад, атрибути `min`, `max`, `type=«number»`) і відправить на сервер шкідливий POST-запит, маршрутизатор Flask перевіряє типи даних та їх відповідність допустимим фізичним діапазонам. Наприклад, об'єм двигуна не може бути від'ємним або перевищувати 10 000 см³, а рік випуску не може бути більшим за поточний календарний рік. У разі виявлення аномальних або невалідних даних система негайно перериває обробку та повертає безпечне повідомлення про помилку, захищаючи математичне ядро бекенду від критичних збоїв (Crash)[47].

Завдяки використанню конструкції `try...except ValueError`, система захищена від некоректного введення даних. У разі успішної валідації, бекенд динамічно генерує HTML-блок `calc-result`, де детально декомпозиується загальна вартість (Лістинг 2.4).

Візуалізацію роботи калькулятора наведено на рисунку 2.4.



Параметр	Вартість
Ціна авто в США	\$4 000
Аукціонний збір	\$600
Доставка	\$1 800
Розмитнення	\$1 700
Загальна вартість під ключ	\$8 100

Рис. 2.4. Практична реалізація автономного калькулятора розмитнення авто

Лістинг 2.4 - Обробка POST-запитів та розрахунок у модулі калькулятора
`@app.route("/calculator", methods=["GET", "POST"])`

```
def calculator():
```

```
    result = ""
```

```

selected_brand = ""
selected_model = ""
price = ""
engine = ""
year = ""

```

```

if request.method == "POST":

```

```

    selected_brand = request.form.get("brand", "")
    selected_model = request.form.get("model", "")
    price = request.form.get("price", "")
    engine = request.form.get("engine", "")
    year = request.form.get("year", "")

```

```

    try:

```

```

        calc = calc_turnkey(float(price), float(engine), int(year))
        result = f""

```

```

        <div class="calc-result">

```

```

            <div class="calc-row"><span>Ціна авто в
            США</span><b>${money(float(price))}</b></div>

```

```

            <div class="calc-row"><span>Аукціонний
            збір</span><b>${money(calc["auction_fee"])}</b></div>

```

```

            <div class="calc-
            row"><span>Доставка</span><b>${money(calc["delivery"])}</b></div>

```

```

            <div class="calc-
            row"><span>Розмитнення</span><b>${money(calc["customs"])}</b></div>

```

```

            <div class="calc-row"><span>Загальна вартість під
            ключ</span><b>${money(calc["total"])}</b></div>

```

```

        </div>

```

```

        ""

```

```

    except ValueError:

```

```

result = ""

<div class="calc-result">

    <div      class="calc-row"><span>Помилка</span><b>Перевірте
правильність введених даних</b></div>

    </div>

    ""

```

```
brand_options = "".join([
    f"<option value='{b}' {'selected' if b == selected_brand else ''}>{b}</option>"
    for b in BRANDS_MODELS.keys()
])
```

```
html = f"""
<main>
  <section class="section">
    <div class="section-head">
      <div>
        <h2>Калькулятор вартості під ключ</h2>
        <p class="lead">Орієнтовний розрахунок витрат на купівлю,
доставку та розмитнення авто зі США.</p>
      </div>
    </div>

    <div class="panel">
      <form method="post">
        <div class="calculator-grid">
          <div class="field">
            <label>Марка</label>
```

```

        <select    name="brand"    id="brand_calc"    data-current-
model="{selected_model}">
            <option value="">Оберіть марку</option>
            {brand_options}
        </select>
    </div>
    <div class="field">
        <label>Модель</label>
        <select name="model" id="model_calc">
            <option value="">Оберіть модель</option>
        </select>
    </div>
    <div class="field">
        <label>Ціна авто в США</label>
        <input    type="number"    step="1"    name="price"
value="{price}" placeholder="15000">
    </div>
    <div class="field">
        <label>Об'єм двигуна</label>
        <input    type="number"    step="0.1"    name="engine"
value="{engine}" placeholder="2.0">
    </div>
    <div class="field">
        <label>Рік авто</label>
        <select    name="year"><option    value="">Оберіть
рік</option>{"".join([f'<option    value='{y}'    {'selected' if str(y)==str(year) else
">{y}</option>" for y in YEARS if y])}</select>
    </div>
    <div class="field">
        <label>&nbsp;</label>

```

```

        <button            class="btn"            style="width:100%;"
type="submit">Розрахувати</button>
    </div>
</div>
</form>
{result}
</div>
</section>
</main>
"""

    return render_page(html, "Калькулятор")

```

Окрім реалізації механізмів динамічного пошуку, фундаментальним завданням під час розробки клієнтської частини стала оптимізація швидкодії рендерингу (відображення) контенту та забезпечення високого рівня інформаційної безпеки. Взаємодія браузера з сервером Flask на етапі відображення результатів базується на системі шаблонізації Jinja2.

Використання Jinja2 дозволяє не просто вставляти змінні у HTML-код, але й імплементувати складну програмну логіку (цикли for, умови if/else) безпосередньо у файлах розмітки. Проте, пряме виведення даних, отриманих зі сторонніх маркетплейсів, несе потенційні загрози кібербезпеці. Зловмисники можуть інтегрувати шкідливий JavaScript-код безпосередньо в опис автомобіля на сайті-донорі. Якщо цей текст потрапить на сторінку розробленого застосунку без обробки, відбудеться атака типу Cross-Site Scripting (XSS)[48]. Для нівелювання цієї загрози у розробленому інтерфейсі використовується вбудований у Jinja2 механізм контекстного автоекранування (Contextual Autoescaping). Усі потенційно небезпечні символи (наприклад, <, >, &) автоматично трансформуються у відповідні безпечні HTML-сутності (HTML entities), що робить виконання шкідливих скриптів у браузері кінцевого користувача неможливим[37].

Значну увагу було приділено питанням Frontend-оптимізації (Performance Optimization). Оскільки кожна картка автомобіля містить високоякісні фотографії лота, завантаження стандартної сторінки з результатами (де може відображатися від 20 до 50 карток) призводило б до одночасного завантаження сотень зображень. Це створило б критичне навантаження на пропускну здатність мережі (Bandwidth) користувача, особливо на мобільних пристроях із нестабільним 4G-з'єднанням[50].

Для вирішення проблеми оптимізації трафіку та прискорення початкового рендерингу сторінки (First Contentful Paint, FCP), у вебзастосунок імплементовано патерн «відкладеного завантаження» (Lazy Loading). За допомогою нативного HTML5-атрибута `loading=«lazy»`, застосованого до тегів `<img>`, браузер завантажує лише ті зображення автомобілів, які безпосередньо знаходяться у видимій частині екрана користувача (Viewport). Фотографії лотів, що знаходяться нижче по сторінці, починають завантажуватися асинхронно і лише в той момент, коли користувач наближається до них під час скролінгу. Це рішення дозволило зменшити стартову вагу сторінки майже на 80%, суттєво заощаджуючи ресурси та забезпечуючи плавність роботи інтерфейсу[50].

Крім того, розробка інтерфейсу велася з урахуванням сучасних стандартів вебдоступності (Accessibility, a11y). Вебдоступність гарантує, що застосунок зможуть комфортно користуватися люди з різними потребами. Для цього всі графічні елементи, включно з піктограмами пошкоджень та фотографіями автомобілів, забезпечені семантичними атрибутами `alt` з текстовим описом. Елементи форм пошуку обгорнуті у відповідні теги `<label>`, а кнопки та інтерактивні бейджі економії підтримують фокусування (focus) для зручної навігації за допомогою клавіатури[46]. Впровадження цих технологічних та дизайнерських рішень трансформувало застосунок із базового скрипта для парсингу в повноцінний, безпечний та високопродуктивний програмний продукт.

Після завершення етапів проектування бекенд-архітектури та візуального інтерфейсу, критично важливим етапом життєвого циклу розробки програмного

забезпечення (SDLC) є етап забезпечення якості (Quality Assurance, QA), комплексне тестування системи та підготовка до виробничого розгортання.

Оскільки ядром розробленої системи є модуль математичних та економічних розрахунків, він вимагає найвищого рівня надійності. Для перевірки його працездатності застосовано методологію модульного тестування (Unit Testing). За допомогою стандартних бібліотек тестування екосистеми Python було розроблено набір тестових сценаріїв (Test Cases) для ізольованої перевірки функції `calc_turnkey()`. Тести свідомо охоплюють різні граничні випадки (Edge Cases): розрахунок митних платежів для електромобілів (де алгоритм повинен підтвердити нульове мито та ПДВ), автомобілів з максимальним віковим коефіцієнтом (понад 15 років, де коефіцієнт заморожується) та гібридних силових установок. Модульне тестування гарантує, що будь-які майбутні зміни в коді (наприклад, оновлення податкових ставок) не зламають існуючу логіку розрахунків[45].

Наступним рівнем стало інтеграційне тестування модуля вебскрейпінгу. Оскільки застосунок безпосередньо залежить від відповідей зовнішнього ресурсу (AUTO.RIA), важливо було забезпечити стійкість системи до можливих змін або мережевих затримок. Для тестування парсера використовувалася техніка «мокування» (Mocking) - замість реальних HTTP-запитів до маркетплейсу, тестові скрипти обробляли заздалегідь збережені локальні HTML-сторінки. Це дозволило багаторазово перевіряти коректність роботи регулярних виразів (re) та методів BeautifulSoup без створення зайвого мережевого навантаження та без ризику блокування IP-адреси розробника системами захисту[43].

Тестування клієнтської частини (Frontend) включало перевірку кросбраузерності (Cross-browser Compatibility) та повної адаптивності. Інтерфейс успішно пройшов валідацію у провідних веббраузерах (Google Chrome, Safari, Mozilla Firefox). За допомогою інструментів розробника (Chrome DevTools) було протестовано поведінку CSS Flexbox та Grid сіток на різних роздільних здатностях екранів, імітуючи пристрої від компактних смартфонів

(ширина 320px) до настільних моніторів (понад 1920px). Особливу увагу було приділено тестуванню подієвих слухачів (Event Listeners) мови JavaScript, зокрема коректності та швидкості роботи динамічних зв'язаних списків при швидкому перемиканні марок автомобілів у формі пошуку.

Фінальним етапом проєктування є підготовка архітектури розгортання (Deployment Architecture). Вбудований у мікрофреймворк Flask сервер (Werkzeug) призначений виключно для режиму розробки (Development) і не здатен ефективно витримувати паралельні навантаження у виробничому середовищі (Production). Тому для реального розгортання системи передбачено використання повноцінного WSGI-сервера (наприклад, Gunicorn), який забезпечує керування пулом робочих процесів (Worker Processes) та ефективний розподіл ресурсів центрального процесора. Для маршрутизації зовнішнього інтернет-трафіку, кешування, обробки статичних файлів (CSS, JS, зображень) та забезпечення безпеки з'єднання через протокол HTTPS (із використанням SSL/TLS сертифікатів) перед WSGI-сервером розгортається зворотний проксі-сервер (Reverse Proxy), такий як Nginx[45].

Такий багатошаровий інженерний підхід - від модульного тестування до проєктування виробничого середовища - гарантує високу стабільність, безпеку та відмовостійкість розробленого програмного продукту, перетворюючи його з локального прототипу на надійне, готове до комерційної експлуатації рішення.

У другому розділі було детально висвітлено процес практичної програмної реалізації клієнт-серверного вебзастосунку для пошуку та аналізу автомобілів із США. За результатами проведеної розробки спроектовано надійну бекенд-архітектуру на базі мікрофреймворку Flask. Реалізована модульна система працює за парадигмою «вилучення даних на льоту» (on-the-fly scraping) без використання стаціонарних баз даних, що гарантує абсолютну актуальність інформації. Разом з тим, розроблено ефективні алгоритми збору та нормалізації даних: завдяки синергії бібліотек requests та BeautifulSoup впроваджено механізм обходу систем захисту від ботів. Використання регулярних виразів (re)

дозволило успішно очищати неструктуровані текстові дані з маркетплейсів, готуючи їх для безпомилкових математичних обчислень митного калькулятора.

Для оптимізації продуктивності та надійності системи було застосовано багатопотокову обробку запитів через `ThreadPoolExecutor`, що дозволило прискорити роботу застосунку у 3-5 разів. Крім того, реалізовано стійку систему обробки мережових винятків (`Exception Handling`) та алгоритми кешування. З боку клієнтської частини створено ергономічний користувацький інтерфейс. Фронтенд застосунку, побудований на `HTML5`, `CSS3` (`Flexbox/Grid`) та `JavaScript`, є повністю адаптивним, безпечним (завдяки захисту від `XSS`) та оптимізованим (із застосуванням `Lazy Loading`). Використання методології карткового дизайну (`Card UI`) із семантичним кольоровим кодуванням фінансових показників суттєво спрощує для користувача процес аналізу економічної вигоди.

Окрему увагу приділено забезпеченню контролю якості (`QA`). Завдяки проведенню модульного (`Unit`) та інтеграційного тестування, а також проєктуванню архітектури розгортання (`WSGI + Reverse Proxy`), розроблений застосунок доведено до стадії стабільного, готового до експлуатації продукту (`MVP`). Таким чином, створений програмний комплекс успішно виконує функцію консолідації та нормалізації даних, формуючи надійний технологічний фундамент. Оцінка економічної ефективності алгоритмів розрахунку вартості «під ключ» та перспективи подальшого розвитку цієї системи детально розглядаються у наступному розділі роботи.

РОЗДІЛ 3. ОЦІНКА ЕФЕКТИВНОСТІ ВЕБЗАСТОСУНКУ ТА ПЕРСПЕКТИВИ ЙОГО ВДОСКОНАЛЕННЯ

Оцінка ефективності розробленого інформаційного продукту не може обмежуватися виключно метриками програмної продуктивності. Ключовим вектором аналізу є оцінка аналітичної та економічної цінності. Вбудована у вебзастосунок функція `calc_turnkey()` бере на себе повну відповідальність за автоматизацію цих розрахунків, звільняючи користувача від необхідності самостійно шукати актуальні ставки податків, митні коефіцієнти та формули[1].

Розрахунок повної вартості включає ціну лота, аукціонний збір, логістику та державних митних платежів, які є найбільш варіативною складовою[19]. Митні платежі для автомобілів з двигунами внутрішнього згоряння (бензиновими та дизельними) розраховуються системою згідно з актуальними на 2025/2026 роки нормами законодавства (закони, відомі як №8487 та №8488)[1].

На першому етапі алгоритм автоматично обчислює ввізне мито за стандартною ставкою 10% від митної вартості транспортного засобу (до митної вартості входить ціна авто на аукціоні та сума витрат на його транспортування до перетину митного кордону України)[6]. Далі обчислюється акцизний податок за формулою 3.1:

$$\text{Акциз} = \text{Базова_Ставка} \times \left(\frac{\text{Об'єм_Двигуна}}{1000} \right) \times \text{Віковий_Коефіцієнт} \quad (3.1)$$

Де Базова_Ставка є диференційованою величиною: 50 євро для бензинових двигунів об'ємом до 3000 см³ (та 100 євро для об'єму понад 3000 см³), і 75 євро для дизельних агрегатів об'ємом до 3500 см³ (та 150 євро для агрегатів понад 3500 см³)[1]. Віковий_Коефіцієнт дорівнює кількості повних календарних років з моменту виробництва автомобіля. Важливою деталлю програмної реалізації (Edge Case Handling) є те, що алгоритм встановлює верхню межу (limit) для цього коефіцієнта на рівні 15, що точно відповідає вимогам Податкового кодексу України (тобто, навіть якщо авто має вік 20 років, у формулу підставляється число 15)[19]. Після визначення мита та акцизу, система нараховує ПДВ у

розмірі 20% на сумарну базу оподаткування (митна вартість + мито + акциз). На завершальному етапі до загального кошторису додається збір до Пенсійного фонду України, який сплачується при першій державній реєстрації транспортного засобу і становить від 3% до 5% залежно від його вартості[18]. Порівняльна декомпозиція податкового навантаження, яке розраховує система, наведена у таблиці 3.1.

Таблиця 3.1

Приклад розрахунку митних платежів розробленою системою (автомобіль вартістю \$15 000)

Компонент платежу	Бензинове авто (напр., 2.0 л, 8 років)	Електромобіль (напр., акумулятор 75 кВт·год)
Ввізне мито	10% (\$1 500)	0% (\$0)
Акцизний збір	$50 \text{ євро} \times 2.0 \times 8 = 800 \text{ євро}$ (≈\$850)	$1 \text{ євро} \times 75 = 75 \text{ євро}$ (≈\$80)
ПДВ (20%)	20% від $(\$15\,000 + 1\,500 + 850)$ = \$3 470	0% (\$0)
Загальні податки	≈\$5 820	≈\$80

Джерело: створено автором

Особливою гордістю логіки розробленого застосунку є динамічна обробка розрахунків для електромобілів (EV). З огляду на те, що електрокари захопили понад 49% ринку американського імпорту у 2025 році[28], інформаційна система повинна бездоганно обробляти цю категорію. Завдяки імплементованій логіці, акциз обчислюється виключно на основі ємності тягової акумуляторної батареї за фіксованою ставкою 1 євро за 1 кВт·год (наприклад, 75 євро для популярної моделі Tesla Model 3 Long Range)[1].

Окрім калькуляції, система розраховує середню ринкову вартість повністю аналогічного автомобіля (з урахуванням марки, моделі, року випуску та комплектації) на внутрішньому ринку України[23].

Для глибокої теоретичної оцінки ефективності розробленого програмного забезпечення (як ІТ-інвестиції) доцільно застосувати наукові методології,

зокрема метод прикладної інформаційної економіки (AIE - Applied Information Economics), розроблений Дугласом Хаббардом[38]. Автоматизація розрахунків повністю усуває ризик критичної людської помилки (Human Error) при визначенні податкових платежів, яка в реальних умовах митного оформлення може призвести до непередбачуваних фінансових втрат у розмірі тисяч доларів[38].

Оцінюючи якісні показники створеного програмного продукту згідно з міжнародним стандартом якості інженерії програмного забезпечення ISO/IEC 25010:2011 (в Україні діє як ДСТУ ISO/IEC 25010:2016), можна констатувати високий рівень відповідності ключовим характеристикам[9]. Модульність дозволяє розробникам швидко оновлювати формули митних платежів без необхідності рефакторингу всього ядра застосунку[9].

Оскільки сучасні вебзастосунки мають працювати безперебійно та бути інтуїтивно зрозумілими для користувача, критично важливим етапом розробки стало тестування системи. Основним методом перевірки на даному етапі було обрано ручне дослідницьке тестування (Exploratory Testing) та перевірку працездатності (Functional Testing). Цей етап складався з наступних кроків:

Функціональне тестування (Functional Testing): Вручну перевірялася логіка роботи автономного митного калькулятора. Тестувальники вводили валідні та невалідні (аномальні) дані у форми пошуку та розрахунку (різні об'єми двигуна, типи палива, роки випуску). Перевірка підтвердила, що обробка винятків try...except ValueError працює коректно, система не допускає критичних збоїв і видає правильні математичні результати згідно із законодавством.

Тестування верстки та кросбраузерність: Здійснено перевірку правильності розташування елементів (Card-based UI) та їх відповідність макетам на різних пристроях. Кросбраузерність перевірялася у браузерях Google Chrome та мобільних браузерах (Safari).

Usability-тестування (Перевірка зручності): Включала перевірку програмного продукту на зручність, аналіз контенту та логіку навігації (переходи між вкладками «Контакти», «Етапи пригону», «Калькулятор»). У ручному

тестуванні взяла участь фокус-група з 12 осіб, яка підтвердила, що кодування зеленим кольором суми економії значно полегшує сприйняття фінансової інформації.

Незважаючи на високий рівень поточної реалізації, архітектура системи закладає фундамент для інноваційного вдосконалення:

1. API-інтеграція: Відмова від парсингу на користь прямої інтеграції з базами даних міжнародних страхових аукціонів Copart та IAAI. Це дозволить аналізувати лоти ще на етапі підготовки до торгів, надаючи користувачам критичну перевагу в часі[8].
2. Штучний інтелект (AI / ML): Інтеграція моделей лінійної регресії або нейронних мереж, натренованих на великих масивах історичних даних українського авторинку, дозволить перейти від констатації поточної ціни до предиктивної аналітики[42].
3. Обробка природної мови (NLP): Алгоритми здатні автоматично сканувати текстові нотатки, виявляючи семантичні маркери прихованих дефектів, які часто не є очевидними з візуальних матеріалів[42].

У третьому розділі було здійснено комплексну оцінку ефективності розробленого вебзастосунку. Доведено, що центральним аналітичним ядром системи є модуль калькуляції вартості «під ключ», який повністю автоматизує складні багатофакторні розрахунки митних платежів, точно враховуючи специфіку різних типів силових установок.

Проведено етап ручного функціонального, кросбраузерного та Usability-тестування за участі реальних користувачів, який підтвердив безпомилкову працездатність алгоритмів обробки даних та стабільність роботи форм. Визначено перспективи масштабування продукту: впровадження API-інтеграцій, алгоритмів штучного інтелекту та технологій NLP. Розроблений програмний продукт успішно розгорнуто у тестовому середовищі, його функціонал повністю відповідає поставленим завданням. Робоча версія вебзастосунку доступна для ознайомлення за мережевим посиланням: <https://untrotted-rosetta-phantasmagorial.ngrok-free.dev>

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було успішно спроектовано, обґрунтовано та програмно реалізовано повнофункціональний клієнт-серверний вебзастосунок, спеціально призначений для автоматизації складних процесів пошуку, консолідації даних та глибокого економічного аналізу автомобілів, що імпортуються із США в Україну. Ця розробка стала безпосередньою відповіддю на актуальні виклики та інформаційні дефіцити сучасного вторинного авторинку, який продемонстрував стрімке зростання попиту на американські авто та кардинальну структурну трансформацію на користь електромобілів. В умовах високої волатильності цін та складності митних процедур, створена система забезпечує користувачів надійним аналітичним інструментарієм для прийняття обґрунтованих фінансових рішень.

Грунтовний аналіз предметної області та дослідження існуючих інформаційних рішень, таких як BidFax, Carfax та платформи логістичних компаній, дозволили виявити їхні суттєві обмеження: фрагментованість функціоналу, затримки в оновленні даних та відсутність прозорості деталізації розрахунків. Це підтвердило науково-практичну необхідність створення інтегрованої системи, здатної функціонувати в парадигмі «Real-Time Data Extraction» (вилучення даних у реальному часі). Завдяки цьому підходу система самостійно агрегує, нормалізує та аналізує дані безпосередньо в момент звернення користувача, що гарантує абсолютну релевантність наданої інформації.

Вибір технологічного стека на базі мови програмування Python та мікрофреймворку Flask виявився оптимальним рішенням для побудови легкої та гнучкої архітектури. Відмова від використання важких стаціонарних реляційних баз даних на користь динамічного парсингу дозволила уникнути проблеми застарівання інформації. Механізми вебскрейпінгу були успішно реалізовані за допомогою бібліотек requests та BeautifulSoup, а імплементація методів багатопоточної обробки запитів на базі класу ThreadPoolExecutor дозволила

ефективно обійти обмеження продуктивності I/O-операцій. Це забезпечило високу швидкість агрегації даних, прискоривши процес у кілька разів порівняно із синхронними методами.

Розроблений користувацький інтерфейс, побудований з використанням HTML, CSS та JavaScript, повністю відповідає сучасним вимогам адаптивного дизайну та ергономіки. Динамічна система фільтрації зі зв'язаними списками на базі AJAX, методологія карткового дизайну (Card UI) з кольоровим кодуванням фінансових показників та інтерактивна мобільна галерея зображень формують безшовний та комфортний користувацький досвід. Таке рішення полегшує сприйняття складної багаторівневої інформації та дозволяє користувачеві зосередитися на ключових аспектах вибору транспортного засобу.

Центральним науковим і практичним досягненням роботи стала розробка комплексного аналітичного ядра - модуля математичної калькуляції вартості автомобіля «під ключ». Програмний алгоритм безпомилково ідентифікує тип силової установки та автоматично обчислює ввізне мито, акцизний збір, ПДВ та збір до Пенсійного фонду відповідно до актуальних податкових законів України (№8487 та №8488), враховуючи пільгові ставки для електромобілів. Вбудований алгоритм оцінки економії, який у реальному часі порівнює вартість імпорту із середніми цінами на внутрішньому ринку, наділяє систему статусом повноцінного інструмента підтримки прийняття фінансових рішень.

Оцінка продукту з позицій методології прикладної інформаційної економіки (AIE) засвідчує його високу ефективність у подоланні інформаційної асиметрії та мінімізації транзакційних витрат. Модульна архітектура системи забезпечує її високу супроводжуваність і готовність до масштабування шляхом інтеграції моделей машинного навчання для предиктивної аналітики або NLP-алгоритмів для розпізнавання прихованих дефектів. Таким чином, розроблений вебзастосунок є завершеним, готовим до експлуатації програмним продуктом, який має значний комерційний та технологічний потенціал на ринку автомобільного імпорту. Ознайомитися з практичною реалізацією та роботою

створеного вебзастосунку можна за посиланням: <https://untrotted-rosetta-phantasmagorial.ngrok-free.dev>

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 2026 Розмитнення Авто зі США в Україні Закон 8487. *Atlantic Express*. URL: <https://atlanticexpress.com.ua/rastamozhka/> (дата звернення: 15.03.2026).
2. Авторинок України у 2025 році: аналітика, статистика, прогноз. *Auto24*. URL: https://24tv.ua/auto/yakym_bude_avtorynok_u_2025_rotsi_prohnoz_n59408 (дата звернення: 12.04.2026).
3. Аналітика імпорту автомобілів зі США в Україну 2024 року. *Eauto.org.ua*. URL: <https://eauto.org.ua/news/655-analitika-importu-avtomobiliv-zi-ssha-v-ukrajinu-2024-roku> (дата звернення: 20.03.2026).
4. Аукціони Америки. Короткий огляд американських сайтів з продажу авто із США. *W8 Shipping*. URL: <https://w8shipping.ua/ohliad-amerykanskykh-saitiv-z-prodazhu-avto-iz-ssha/> (дата звернення: 20.03.2026).
5. Використання бібліотеки BeautifulSoup для парсингу HTML-сторінок та збору даних. *Scribd*. URL: <https://www.scribd.com/document/728370402/> (дата звернення: 25.03.2026).
6. Все про розмитнення авто в Україні у 2025 році. *MegaCredit*. URL: <https://megacredit.ua/blog/detali-rastamozhki-avtomobiley-v-2025-godu> (дата звернення: 11.03.2026).
7. Галка Л. Парсинг сайтів: що це, як працює та навіщо парсити дані. *Блог Promodo*. 2025. URL: <https://www.promodo.ua/blog/parsing-saytiv-shcho-ce-yak-pracyuie-ta-navishcho-parsiti-dani> (дата звернення: 02.04.2026).
8. Дзендзя А. А., Левус Є. В., Вовк А. С. Аналіз ефективного застосування методів автоматизованого збирання даних з вебсайтів. *Науковий вісник НЛТУ України*. 2024. Т. 34, № 7. С. 128–136. DOI: <https://doi.org/10.36930/40340716>.
9. ДСТУ ISO/IEC 25010:2016 Інженерія систем і програмних засобів. Вимоги до якості систем і програмних засобів та її оцінювання (SQuaRE).

- Будстандарт.* URL: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=69134 (дата звернення: 15.03.2026).
10. Миколайчук Р., Старинський І., Миколайчук В. Аналіз технологічних аспектів реалізації вебскрапінгу статичних і динамічних сайтів. *Сучасні інформаційні технології у сфері безпеки та оборони*. Київ, Україна. 2024. 51(3). С. 80–88. DOI: 10.33099/2311-7249/2024-51-3-80-88.
 11. Покрокова інструкція. Скільки коштує розмитнення авто в Україні 2025 і що потрібно знати щодо процедури. *NV.ua*. URL: <https://nv.ua/ukr/ukraine/events/rozmitnennya-avto-v-ukrajini-2025-pokroкова-instrukciya-dokumenti-ta-kalkulyator-vartosti-50510398.html> (дата звернення: 12.04.2026).
 12. Попит на вживані авто зі США продовжує зростати: 52% імпорту - електромобілі. *Skilky-skilky.info*. URL: <https://skilky-skilky.info/popit-na-vzhyvani-avto-zi-ssha-prodovzhuie-zrostaty-52-importu-elektromobili/> (дата звернення: 26.03.2026).
 13. Попит на вживані автомобілі зі США в Україні зріс майже на 50%. *Zaxid.net*. URL: https://zaxid.net/popit_na_amerikantsiv_z_probigom_v_ukrayini_zris_mayzhe_na_50_n1628405 (дата звернення: 26.03.2026).
 14. Правила ввезення автомобілів з-за кордону в Україну у 2025-2026 роках. *Юридичні послуги (LegalAid)*. URL: <https://legalaidd.ua/blog/pravylyavvezennya-avtomobiliv-z-zakordonu-v-ukrayinu/> (дата звернення: 11.03.2026).
 15. Проектування інформаційних систем: Загальні питання теорії проектування ІС : конспект лекцій. О. С. Коваленко, Л. М. Добровська. - Київ : КПП ім. Ігоря Сікорського, 2020. - 192 с.
 16. Пропозиція щодо покращення Python (PEP 3148) – futures: execute computations asynchronously. B. Quinlan. *Python Software Foundation*. – 2009. URL: <https://peps.python.org/pep-3148/> (дата звернення: 16.04.2026).

17. Ринок вживаних автомобілів в Україні у 2024 році: аналіз і тенденції. *Autopark.UA*. URL: <https://autopark.ua/uk/news/rynok-poderzannyx-avtomobilei-v-ukraine-v-2024-godu-analiz-i-tendencii> (дата звернення: 21.03.2026).
18. Розмитнення авто з США в Україні 2025 - Покроковий гід. *Acars.ua*. URL: <https://acars.ua/articles/customs-clearance-usa-cars-ukraine-2025> (дата звернення: 15.04.2026).
19. Розмитнення авто з США в Україні: Посібник TopAutoGear. *TopAutoGear*. 2025. URL: <https://topautogear.com.ua/rozmytnennia-avto-z-ssha-v-ukraini> (дата звернення: 15.04.2026).
20. Розробка вебзастосунку з використанням фреймворку Flask та графічної бібліотеки. *eKMAIR (Репозитарій НаУКМА)*. URL: <https://ekmair.ukma.edu.ua/server/api/core/bitstreams/8a2cc4c0-01e6-4052-856e-0376c0083966/content> (дата звернення: 30.03.2026).
21. Розробка та впровадження ефективних методів вебскрапінгу для автоматизованого збору і обробки даних з використанням Python / Р. К. Січкарюк, Н. В. Корніловська, І. А. Лур'є, М. О. Вороненко. *Інформатика. Культура. Техніка*. 2024. Т. 1. С. 62–68. DOI: <https://doi.org/10.15276/ict.01.2024.08>. URL: <https://ict.od.ua/index.php/journal/uk/article/view/98> (дата звернення: 03.04.2026).
22. Сервіс «BidFax.info» - надійний помічник для перевірки авто із США. *W8 Shipping*. URL: <https://w8shipping.ua/servis-bidfax-info-nadijnyj-pomichnyk-dlya-perevirky-avto-iz-ssha/> (дата звернення: 20.03.2026).
23. Терещенко Л. О. Оцінка економічного ефекту від впровадження управлінських інформаційних систем: показники економічного ефекту. *Економіка та суспільство*. 2021. Вип. 30. DOI: <https://doi.org/10.32782/2524-0072/2021-30-7>. URL: https://www.researchgate.net/publication/354606615_OCINKA_EKONOMICNOGO_EFEKTU_VID_VPROVADZENNA_UPRAVLINSKIH_INFORMACIJNIH

- SISTEM POKAZNIKI EKONOMICNOGO EFEKTU (дата звернення: 17.03.2026).
24. У 2025 попит на вживані авто із США зріс на 48%, лідерство за електромобілями. *UKR.NET*. URL: <https://www.ukr.net/news/details/auto/115594960.html> (дата звернення: 16.04.2026).
25. Український ринок нових авто зріс на 7% у липні. *НафтоРинок*. URL: <https://www.nefterynok.info/novosti/ukrainskiy-rinok-novih-avto-zris-na-7-u-lipni> (дата звернення: 12.04.2026).
26. Українці масово купують «американців»: топ вживаних авто з США. *TCH*. URL: https://tsn.ua/auto/news/avto_mir/ukrayintsi-masovo-kupuyut-amerykantsiv-top-vzyvanykh-avto-z-ssha-2959001.html (дата звернення: 12.04.2026).
27. Що таке парсинг даних та сайтів та як його використовувати. *Promodo*. URL: <https://promodo.ua/blog/shcho-take-parsing-danih-ta-saytiv> (дата звернення: 02.04.2026).
28. Які авто з США найчастіше купують в Україні? Статистика за жовтень 2025 року. *UKR.NET*. URL: <https://www.ukr.net/news/details/auto/114584048.html> (дата звернення: 16.04.2026).
29. Automated data extraction: Implementing web scraper with python and beautiful soup / Н. Mourya, Н. Singh, S. Singh[et al.]. Recent Trends in *Intelligent Computing and Communication*. Vol. 1. London : CRC Press, 2025. DOI: 10.1201/9781003593034-6. URL: <https://www.taylorfrancis.com/chapters/edit/10.1201/9781003593034-6/automated-data-extraction-implementing-web-scraper-python-beautiful-soup-himanshu-mourya-harsh-singh-shivam-singh-poonam-nandal-rahul-kumar> (дата звернення: 05.04.2026).
30. BidFax - корисний безкоштовний сервіс для перевірки історії авто із США. *Columb Auto*. URL: <https://columbauto.com.ua/ua/blog/bidfax-poleznyy-besplatnyy-servis-dlya-proverki-istorii-avto-iz-ssha/> (дата звернення: 25.03.2026).

31. Build a Basic Python Web Application. *Real Python*. 2024. URL: <https://realpython.com/python-web-applications/#build-a-basic-python-web-application> (дата звернення: 27.03.2026).
32. Carfax: Звіт Карфакс - Повна історія будь-якого авто із США. *Carfax Service*. URL: <https://carfaxservice.com.ua/> (дата звернення: 25.03.2026).
33. Chapman C. Color Theory for Designers, Part 1: The Meaning of Color. *Smashing Magazine*. URL: <https://www.smashingmagazine.com/2010/01/color-theory-for-designers-part-1-the-meaning-of-color/> (дата звернення: 08.04.2026).
34. Chris K. Web Development Project – How to Build a Landing Page with HTML, CSS, and JavaScript. *freeCodeCamp*. 2022. URL: <https://www.freecodecamp.org/news/how-to-make-a-landing-page-with-html-css-and-javascript/> (дата звернення: 08.04.2026).
35. Concurrent Execution in Python (ThreadPoolExecutor). *Python Software Foundation Documentation*. URL: <https://docs.python.org/3/library/concurrent.futures.html> (дата звернення: 30.03.2026).
36. Flask vs. Django - comparison of Python frameworks. *The Story*. URL: <https://thestory.is/en/journal/flask-vs-django> (дата звернення: 30.03.2026).
37. Harnessing Flask for Web Scraping and Sentiment Analysis. *IJCA Online*. URL: <https://www.ijcaonline.org/> (дата звернення: 02.04.2026).
38. Hubbard D. W. How to Measure Anything: Finding the Value of "Intangibles" in Business. 3rd ed. Hoboken : John Wiley & Sons, 2014. 432 p.
39. Is Scrapy Better than BeautifulSoup?. *GeeksforGeeks*. URL: <https://www.geeksforgeeks.org/python/is-scrapy-better-than-beautifulsoup/> (дата звернення: 03.04.2026).
40. Lisandro R. Card-Based UI Design: Structure, Advantages, and Best Practices. *Design Bootcamp (Medium)*. 2024. URL: <https://medium.com/design-bootcamp/card-based-ui-design-structure-advantages-and-best-practices-69042d1f0786> (дата звернення: 10.04.2026).

41. Palakollu S. M. Scrapy Vs Selenium Vs BeautifulSoup for Web Scraping. *Analytics Vidhya (Medium)*. URL: <https://medium.com/analytics-vidhya/scrapy-vs-selenium-vs-beautiful-soup-for-web-scraping-24008b6c87b8> (дата звернення: 03.04.2026).
42. Price Prediction and Classification of Used-Vehicles Using Supervised Machine Learning / S. Pudar, L. Kozina, M. Bugaric[et al.]. *Sustainability*. 2022. Vol. 14, Iss. 24. P. 17034. DOI: <https://doi.org/10.3390/su142417034>.
43. Python Mocking and Patching for Web Scraping Applications. *Python Software Foundation*. URL: <https://docs.python.org/3/library/unittest.mock.html> (дата звернення: 05.04.2026).
44. Regular expression operations. *Python Software Foundation Documentation*. URL: <https://docs.python.org/3/library/re.html> (дата звернення: 05.04.2026).
45. Tetteh S. G. Software Testing Techniques and Levels in Software Development. *ESP International Journal of Advancements in Computational Technology*. 2024. Vol. 2, Iss. 1. P. 10–19. DOI: 10.56472/25838628/IJACT-V211P102. URL: https://www.researchgate.net/publication/387460992_Software_Testing_Techniques_and_Levels_in_Software_Development (дата звернення: 14.04.2026).
46. UI Design Patterns for accessibility and processing flows. *UI-Patterns*. URL: <https://ui-patterns.com/patterns> (дата звернення: 10.04.2026).
47. Web Application Security: Input Validation and Sanitization Techniques. *OWASP Foundation*. URL: <https://owasp.org/www-project-top-ten/> (дата звернення: 14.04.2026).
48. Web technology for developers | *MDN Web Docs. Mozilla*. URL: <https://developer.mozilla.org/en-US/docs/Web> (дата звернення: 17.03.2026).
49. What is CSS, how does it work and what is it used for?. *Skillcrush*. URL: <https://skillcrush.com/blog/css/> (дата звернення: 17.03.2026).
50. What is HTML? Hypertext markup language basics and optimization explained. *Hostinger Tutorials*. URL: <https://www.hostinger.com/tutorials/what-is-html> (дата звернення: 17.03.2026).

51. What is javascript? A basic introduction to JS for beginners. *Hostinger Tutorials*.
URL: <https://www.hostinger.com/tutorials/what-is-javascript> (дата звернення: 27.03.2026).
52. What is web design? The ultimate guide to website design. *Pagecloud Blog*.
URL: <https://www.pagecloud.com/blog/web-design-guide> (дата звернення: 10.04.2026).