

**ІНАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”**

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**До захисту допущено:
В.о. завідувача кафедри
Артем ВОЛОКИТА**

_____ (підпис)

“ __ ” _____ 2026 р.

**Дипломний проєкт
на здобуття ступеня бакалавра
за освітньо-професійною програмою
«Комп'ютерні системи та мережі»
спеціальності 123 «Комп'ютерна інженерія»**

на тему: Система пошуку товарів з використанням розподілених обчислень та штучного інтелекту для обробки товарів

Виконав : студент 4 курсу, групи ІО-24
(шифр групи)

Бережанський Данііл Вадимович

(прізвище, ім'я, по батькові)

(підпис)

Керівник ст.викл., док.філос. Гончаренко О.О.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Консультант (нормоконтроль) ст.викл., док.філос. Гончаренко О.О.

(назва розділу)

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

(підпис)

Рецензент _____

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Засвідчую, що у цьому дипломному
проєкті немає запозичень з праць інших
авторів без відповідних посилань.

Студент _____

(підпис)

Київ – 2026 р.

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”**

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма «Комп'ютерні системи та мережі»
спеціальності 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

В. о. завідувача кафедри

Артем ВОЛОКИТА

(підпис)

“ _ ” _____ 2026 р.

ЗАВДАННЯ

на бакалаврський дипломний проєкт студента

Бережанського Данііла Вадимовича

1. Тема проєкту Система пошуку товарів з використанням розподілених обчислень та штучного інтелекту для обробки товарів
керівник проєкту Гончаренко Олександр Олексійович. ст.викл., док.філос.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджений наказом по університету від 23 травня 2025 р. №1705-с.
2. Термін здачі студентом закінченого проєкту 01 червня 2026 р.
3. Вихідні дані до проєкту сервіс GroceryScout розроблений на Python, з використанням FastAPI, curl_cffi та великої мовної моделі
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
огляд наявних рішень у сфері порівняння цін, огляд технологій розробки системи, опис розробки застосунку, методика роботи розробленого застосунку.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема (діаграма класів), алгоритм дій програмного забезпечення.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Гончаренко О.О.		

7. Дата видачі завдання: 05 січня 2026 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	<i>Затвердження теми проєкту</i>	15.03.2026	
2.	<i>Вивчення та аналіз завдання</i>	16.03.2026-10.04.2026	
3.	<i>Розробка архітектури та загальної структури системи</i>	10.04.2026-25.04.2026	
4.	<i>Розробка структур окремих підсистем</i>	25.04.2026-10.05.205	
5.	<i>Програмна реалізація системи</i>	10.05.2026-20.05.2026	
6.	<i>Оформлення пояснювальної записки</i>	20.05.2026-28.05.2026	
7.	<i>Захист програмного продукту</i>	06.06.2025	
8.	<i>Захист</i>	15.06.2025	

Студент-дипломник _____ Данііл БЕРЕЖАНСЬКИЙ
(підпис)

Керівник проєкту _____ Олександр ГОНЧАРЕНКО
(підпис)

АНОТАЦІЯ

У роботі розроблено сервіс GroceryScout для порівняння цін на продовольчі товари в роздрібних мережах. Рішення усуває проблему несумісності найменувань товарів між магазинами завдяки класифікації за допомогою великих мовних моделей.

Функціональні можливості: пошук товарів у декількох магазинах з обходом антибот-захисту, класифікація (смак, маса/об'єм, бренд) через LLM API, порівняння цін та перевірка наявності у реальному часі, REST API, інтерактивна мапа на основі Leaflet.js. Продукт реалізовано мовою Python із використанням FastAPI та curl_cffi.

Ключові слова: порівняння цін, веб-скрапінг, нормалізація товарів, штучний інтелект, Python, FastAPI, LLM API, Leaflet.

ABSTRACT

This thesis presents GroceryScout – a service for grocery price comparison across retail chains. The solution addresses product name incompatibility between retailers through automatic classification using large language models.

System features include: multi-store product search with anti-bot bypass, LLM-based product classification (flavor, weight/volume, brand), real-time price comparison and availability checking, REST API, and an interactive store map built with Leaflet.js. The product is implemented in Python using FastAPI and curl_cffi.

Keywords: price comparison, web scraping, product normalization, artificial intelligence, Python, FastAPI, LLM API, Leaflet.

[illegible]

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Система пошуку товарів з використанням розподілених обчислень
та штучного інтелекту для обробки товарів»

Київ – 2026

ЗМІСТ

НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ	2
ПІДСТАВИ ДЛЯ РОЗРОБКИ	2
МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ.....	2
ДЖЕРЕЛА РОЗРОБКИ.....	2
ТЕХНІЧНІ ВИМОГИ.....	3
Вимоги до розробленого продукту	3
Вимоги до програмного забезпечення	3
Вимоги до апаратної частини	3
ЕТАПИ РОЗРОБКИ	4

					ІАЛЦ.467200.002 ТЗ			
		№ докум.	Підпис	Дата				
Розробив	Бережанський Д.В.				Система пошуку товарів з використанням розподілених обчислень та штучного інтелекту для обробки товарів Технічне завдання	Літ.	Аркуш	Аркушів
Перевірив	Гончаренко О.О.						1	4
						НТУУ “КПІ” ФІОТ ІО-24		
Н. Контр.	Гончаренко О.О.							
Затвердив	Волокита А.М.							

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку системи пошуку товарів з використанням розподілених обчислень та штучного інтелекту для обробки товарів, а також подальшої підтримки та вдосконалення проєкту

Область застосування – споживчий сектор, дати змогу користувачам порівнювати ціни та перевіряти наявність на товари у зоні доступу, автоматизована класифікація та нормалізація товарних найменувань, пошук за допомогою геолокації найближчих торгових точок із потрібним товаром

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського проєкту за освітньо-професійною програмою Комп'ютерні системи та мережі» спеціальності 123 «Комп'ютерна інженерія», затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Мета та призначення даної роботи – розробка системи автоматизованого пошуку та порівняння цін, перевірка наявності на товари в роздрібних мережах на території Києву, із використанням розподілених обчислень та технологій штучного інтелекту, що дасть змогу користувачам ефективно знаходити найвигідніші пропозиції у вибраних користувачем торгових точках

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проєкту є офіційні документації бібліотек та фреймворків (FastAPI, LiteLLM, Leaflet.js, curl_cffi), публікації та статті в мережі Інтернет на тему веб-скрапінгу, обробки природної мови та розподілених обчислень, а також науково-технічна література з відповідної тематики.

					ІАЛЦ.467200.002 ТЗ	Арк.
						2
Зм.	Арк.	№ докум.	Підпис	Дата		

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблений веб сервіс має виконувати такі вимоги:

- Автоматичний пошук товарів за запитом одночасно у декількох магазинах.
- Обхід антибот систем.
- Класифікацію та нормалізацію товарів (бренд, смак, маса/об'єм) з використанням великої мовної моделі.
- Порівняння цін та перевірку наявності у магазинах у реальному часі.
- Відображення підтримуваних магазинів на інтерактивній мапі з можливістю вибору торгових точок для порівняння
- Можливість масштабування системи
- Централізоване керування налаштуваннями, API-ключами та промттами через адмін-панель
- Можливість вмикати/вимикати торгові мережі без перезапуску системи

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac чи Linux.
- Доступ до Groq API або АПІ іншої великої мовної моделі
- Середовище Python 3.10 та вище
- Docker та Docker Compose для контейнеризації та оркестрації сервісів
- Redis 7+ для черги задач та міжпроцесної комунікації

5.3. Вимоги до апаратної частини

- ЦП AMD Ryzen 7 6800H
- RAM 16 ГБ.
- GPU NVIDIA RTX 3050 з обсягом VRAM 4 ГБ

					ІАЛЦ.467200.002 ТЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		3

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	15.03.2026
Вивчення та аналіз завдання	16.03.2026-10.04.2026
Розробка архітектури та загальної структури системи	10.04.2026-25.04.2026
Розробка структур окремих частин системи	25.04.2026-10.05.205
Програмна реалізація системи	10.05.2026-20.05.2026
Оформлення пояснювальної записки	20.05.2026-28.05.2026

					ІАЛЦ.467200.002 ТЗ	Арк.
						4
Зм.	Арк.	№ докум.	Підпис	Дата		

**ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ**

на тему: «Система пошуку товарів з використанням розподілених обчислень та
штучного інтелекту для обробки товарів»

Київ – 2026

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ.....	4
ВСТУП.....	5
РОЗДІЛ 1. АНАЛІЗ ЗАДАЧІ АГРЕГАЦІЇ ТА ПОРІВНЯННЯ ЦІН У РОЗДРІБНІЙ ТОРГІВЛІ	7
1.1 Аналіз предметної області.....	7
1.1.1 Роздрібна торгівля та проблема порівняння цін	7
1.1.2 Веб-скрапінг як метод збору даних.....	8
1.1.3 Антибот-захист та методи його обходу	8
1.2 Огляд існуючих рішень	9
1.2.1 Supermarkets.com.ua	10
1.2.2 Goods.....	11
1.2.3 Де Дешевше	15
1.3 Порівняльний аналіз існуючих рішень	19
ВИСНОВОК ДО РОЗДІЛУ	21
РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ОГЛЯД ТЕХНІЧНИХ РІШЕНЬ.....	23
2.1 Формулювання задачі	23
2.2 Функціональні можливості системи	24
2.2.1 Геолокаційний вибір торгових точок.....	24
2.2.2 Розподілений пошук товарів у реальному часі	24
2.2.3 Нормалізація та групування товарів на основі LLM	25
2.2.4 Програмний інтерфейс доступу до даних	25
2.2.5 Інтерфейс користувача	25
2.2.6 Адміністративна панель	26
2.3 Технічні вимоги системи.....	26
2.4 Мікросервісна архітектура	27
2.5 Аналіз методів автоматизованого збору даних.....	28

					ІАЛЦ.467200.003 ПЗ							
		№ докум.	Підпис	Дата								
Розробив		Бережанський Д.В.			Система пошуку товарів з використанням розподілених обчислень та штучного інтелекту для обробки товарів Технічне завдання			Літ.	Аркуш	Аркушів		
Перевірив		Гончаренко О.О.								1	78	
Н. Контр.		Гончаренко О.О.						НТУУ “КПІ” ФІОТ				
Затвердив		Волокита А.М.						ІО-24				

2.5.1 Порівняльний аналіз HTTP-клієнтів	29
2.5.2 Таблиця порівняння HTTP-клієнтів	31
2.6 Методи виявлення однакових товарів у різних торгових мережах ...	32
2.7 Розподілена обробка запитів	33
2.8 Обґрунтування вибору технологічного стеку	35
2.8.1 Python.....	35
2.8.2 FastAPI.....	36
2.8.3 curl_cffi	36
2.8.4 Redis.....	37
2.8.5 Taskiq	37
2.8.6 LiteLLM та Groq API.....	38
2.8.7 Leaflet.js	38
2.8.8 Docker та Docker Compose.....	38
2.8.9 PostgreSQL	39
2.8.10 PocketBase	40

ВИСНОВОК ДО РОЗДІЛУ	41
---------------------------	----

РОЗДІЛ 3. ДЕТАЛІ РОЗРОБКИ СИСТЕМИ.....	43
--	----

3.1 Загальна архітектура системи	43
3.1.1 Взаємодія компонентів	44
3.1.2 Спільна бібліотека та динамічна конфігурація	44
3.1.3 ShopRegistry	46
3.1.4 Ініціалізація системи.....	48
3.2 Модуль збору даних.....	49
3.2.1 Клієнти торгових мереж	49
3.2.2 Обхід антибот-захисту з curl_cffi	51
3.2.3 Розподілена обробка через Taskiq	53
3.3 LLM сервіс нормалізації.....	55
3.4 Система аутентифікації	56
3.5 Email сервіс	58

3.6 Веб-інтерфейс	59
ВИСНОВОК ДО РОЗДІЛУ	61
РОЗДІЛ 4. ТЕСТУВАННЯ ТА ДЕМОНСТРАЦІЯ РОБОТИ СИСТЕМИ	62
4.1 Демонстрація роботи системи	62
4.1.1 Реєстрація та авторизація	62
4.1.2 Вибір торгових точок на мапі	63
4.1.3 Пошук та нормалізація товарів	64
4.1.4 Вибір аналогів.....	67
4.1.5 Відправка чеку на email	68
4.2 Тестування адміністративних можливостей	70
4.2.1 Вимкнення торгової мережі	70
4.2.2 Зміна налаштувань LLM.....	71
4.3 Регресійне тестування промптів	72
ВИСНОВОК ДО РОЗДІЛУ	74
ВИСНОВКИ.....	75
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	77

ПЕРЕЛІК СКОРОЧЕНЬ

API – (Application Programming Interface) Інтерфейс програмування застосунків

REST – (Representational State Transfer) Архітектурний стиль взаємодії компонентів

LLM – (Large Language Model) Велика мовна модель

TLS – (Transport Layer Security) Протокол захисту транспортного рівня

JSON – (JavaScript Object Notation) Формат обміну даними

HTML – (HyperText Markup Language) Мова розмітки гіпертексту

HTTP – (HyperText Transfer Protocol) Протокол передачі гіпертексту

SPA – (Single Page Application) Односторінковий застосунок

SSE – (Server-Sent Events) Технологія однонаправленої передачі подій від сервера до клієнта

					ІАЛЦ.467200.003 ПЗ	Арк.
						4
Зм.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Більшість сучасних технологій та цифрових сервісів спрямовані на спрощення повсякденного життя людини, автоматизацію та прискорення рутинних процесів та економію часу користувача. Завдяки розвитку вебтехнологій та e-commerce люди отримали швидкий доступ до великої кількості інформації, які раніше вимагали багато часу. Однією з найпоширеніших потреб кожної людини є придбання продовольчих товарів, і кожен користувач зацікавлений у найзручнішій та найвигіднішій можливості придбати бажані товари.

У сучасних умовах одна й та сама продукція може мати різну ціну залежно від торгової мережі, спеціальних акцій або регіональних особливостей ціноутворення. При цьому більшість великих українських супермаркетів надає доступ до інформації про асортимент, наявність через власні вебсайти або мобільні застосунки. Однак за наявності декількох магазинів поруч користувач змушений самотійно перевіряти декілька різних сервісів, що є незручним та потребує багато часу, крім того однакові товари можуть називатися трохи по-різному, що додає ще більше незручностей робити це самотійно.

Ці проблеми та виклики підкреслюють необхідність у створенні системи, що здатна автоматизувати процес пошуку, нормалізувати та співставляти ціни на товари між різними магазинами та перевіряти наявність у торгових точках. Для вирішення задачі нормалізації та класифікації товарних найменувань у даній роботі застосовуються великі мовні моделі (LLM), які здатні семантично зіставляти назви товарів незалежно від відмінностей у формулюванні, порядку слів чи одиницях виміру.

Актуальність теми дипломної роботи зумовлена відсутністю на українському ринку рішення, яке б комплексно вирішувало задачу пошуку та порівняння цін на продовольчі товари з урахуванням географічної близькості

					ІАЛЦ.467200.003 ПЗ	Арк.
						5
Зм.	Арк.	№ докум.	Підпис	Дата		

магазинів до користувача. Існуючі сервіси порівняння цін працюють з фіксованим каталогом товарів та здійснюють порівняння лише на рівні окремого товару, не надаючи можливості визначити, в якому магазині повний кошик покупок обійдеться дешевше. Крім того, більшість таких рішень орієнтована на онлайн-магазини або не враховує геолокацію користувача, тоді як для щоденних покупок принципово важливою є наявність товару саме в найближчих торгових точках - придбання продуктів у віддаленій частині міста може нівелювати будь-яку економію на ціні.

Важливою відмінністю розробленої системи є те, що дані про ціни та наявність товарів отримуються безпосередньо з API торгових мереж у момент запиту користувача, а не з попередньо сформованого статичного каталогу. Це гарантує актуальність інформації та враховує поточні акції й знижки. Система реалізована у вигляді набору незалежних мікросервісів, що забезпечує її масштабованість та можливість легкого підключення нових торгових мереж без зупинки роботи системи в цілому.

					ІАЛЦ.467200.003 ПЗ	Арк.
						6
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ЗАДАЧІ АГРЕГАЦІЇ ТА ПОРІВНЯННЯ ЦІН У РОЗДРІБНІЙ ТОРГІВЛІ

1.1 Аналіз предметної області

1.1.1 Роздрібна торгівля та проблема порівняння цін

Роздрібна торгівля є однією з найбільших галузей споживчого ринку України. Лише у одному Києві налічується 169 магазинів АТБ, серед найближчих конкурентів також варто згадати Varus, Silpo, Novus та інші. Кожна мережа магазинів формує власну цінову політику, проводить унікальні акції та знижки, що безпосередньо створює різницю у вартості на однаковий або схожий товар між мережами.

У сучасного споживача є можливість вибирати серед декількох магазинів у пішій доступності, а вибір місця покупки робиться на основі власного досвіду або потребує самостійного порівняння асортименту серед магазинів, що є досить клопіткою працею. Більшість українських роздрібних мереж надають доступ до інформації про ціни та асортимент своїх магазинів через власні вебсайти, проте єдиного інструменту для агрегації та порівняння цін у реальному часі немає або доступ до функціоналу обмежений, наприклад немає можливості порівняти повний кошик, вибирати серед аналогів або у межах фізичного доступу.

Також додаткових складнощів додає неузгодженість найменувань товарів між різними магазинами, де пряме текстове порівняння стає неможливим. Для вирішення цієї задачі потрібно використати методи обробки природної мови.

					ІАЛЦ.467200.003 ПЗ	Арк.
						7
Зм.	Арк.	№ докум.	Підпис	Дата		

1.1.2 Веб-скрапінг як метод збору даних

Веб-скрапінг – це спосіб автоматичного збору структурованих даних з вебсайтів, АПІ мобільних застосунків та інших вебресурсів за допомогою спеціалізованих скриптів або сервісів, які зазвичай називають скрейперами. Це спосіб отримати різноманітні дані з сайтів у бажаному форматі та структурі, без ручного збору.

Отримання даних здійснюється через пряме звернення до даних, які зазвичай розміщені на HTTP-сторінці або у JSON запиту. Наступним кроком йде парсинг даних – перетворення сирих даних у бажаний формат, для HTML існують готові парсери, які будують DOM-дерево документа, по якому можна ітеруватися та отримувати необхідні дані через CSS-селектори або XPath-вирази. Парсинг JSON набагато легше і швидше, проте не всі дані можуть бути присутні у певному запиті.

1.1.3 Антибот-захист та методи його обходу

Велика кількість вебсервісів впроваджують захист від автоматизованого збору даних, оскільки дані можуть використовуватись для отримання конкуретних переваг, а постійні запити створюють додаткове навантаження на інфраструктуру власника вебсервісу.

Серед найпоширеніх способів захисту можна визначити такі:

- CAPTCHA та JavaScript-челенджі – інтерактивні завдання, виконання яких є недоступним для звичайних HTTP-запитів, оскільки потребує дій характерних для людини: натискання кнопок, вибору зображень тощо.

- Аналіз заголовків та кукісів перевірка значень User-Agent, Accept-Language, Accept-Encoding та інших значень, що можуть видавати не справжнього користувача
- Фінгепринтинг (IP та TLS) - ідентифікація клієнта за IP-адресою та параметрами SSL/TLS-з'єднання. Під час встановлення захищеного з'єднання клієнт передає набір параметрів: версію протоколу, підтримувані шифри та розширення, що разом формують унікальний відбиток. Стандартні HTTP-бібліотеки зазвичай не підтримують TLS або мають характерні відбитки, які відрізняють їх від браузера, і через це вони легко виявляються антибот-системами

Основна ціль фінгепринтингу – ідентифікувати того самого користувача навіть, якщо він змінить IP-адресу, оскільки кожна версія браузера, пристрою користувача та інша інформація зберігаються навіть при зміні IP-адреси і формують унікальний відбиток користувача. Таким чином, якщо відбиток не змінюється, а запити приходять з різних IP-адрес, можна зрозуміти хто це робить і заблокувати йому доступ.

Кожен зі способів захисту не є винятковим, і більшість систем захисту охоплюють комбінують усі способи, що в сукупності може сильно ускладнити скрапінг певних вебресурсів.

1.2 Огляд існуючих рішень

На основі проведеного аналізу предметної області далі будуть розглянуті існуючі рішення у сфері порівняння цін на продовольчі товари, їх ключові підходи, переваги та недоліки.

					ІАЛЦ.467200.003 ПЗ	Арк.
						9
Зм.	Арк.	№ докум.	Підпис	Дата		

1.2.1 Supermarkets.com.ua

Supermarkets.com.ua – вебсервіс агрегації цін на товари в супермаркетах України, з основного функціоналу – пошук товарів та порівняння цін серед торгових мереж, з можливістю перевірити наявність у фізичних точках. Під час аналізу сайту було помічено ряд суттєвих обмежень.

По-перше як видно з Рис 1.1 за запитом “чипси люкс сир” було знайдено лише 1 товар, хоча таких фактично більше, через різні грамовки

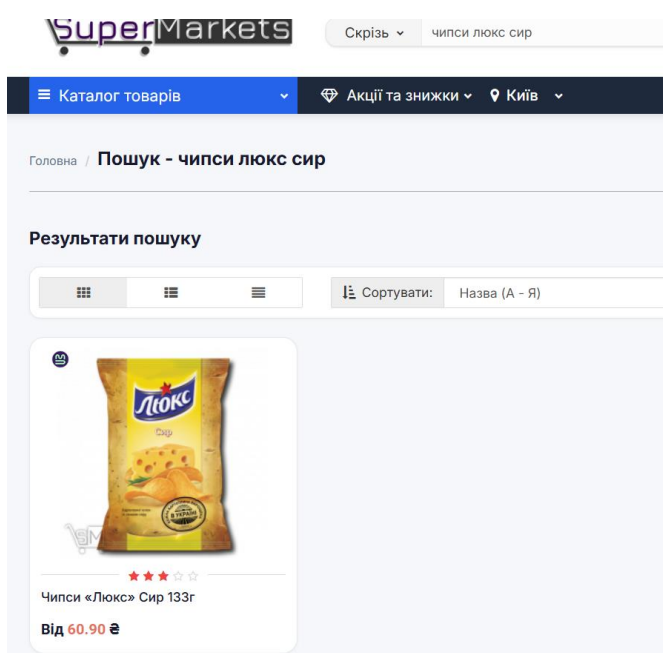


Рисунок 1.1 – Пошук Supermarkets.com.ua

По-друге – ціни, які вказав вебсервіс не збігаються з реальними. Як видно з Рисунок 1.2 та Рисунок 1.3 за адресою Героїв полку “Азов”, 5 (Малиновського, 5) ціна та наявність не збігаються. Цікаво що навіть базова ціна не збігається, тобто без урахування знижки.

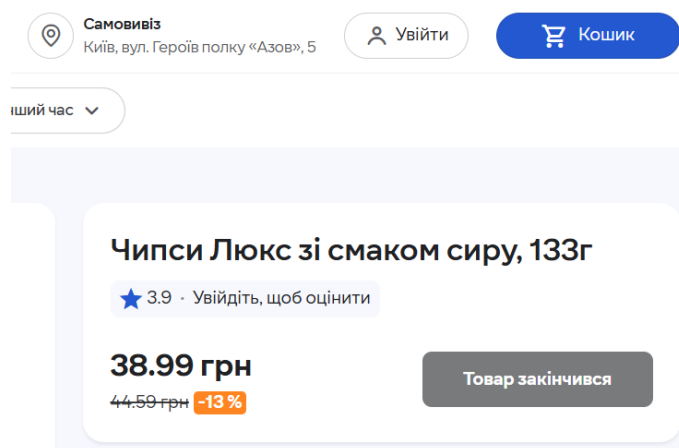


Рисунок 1.2 – Ціна та наявність з офіційного сайту Сільпо

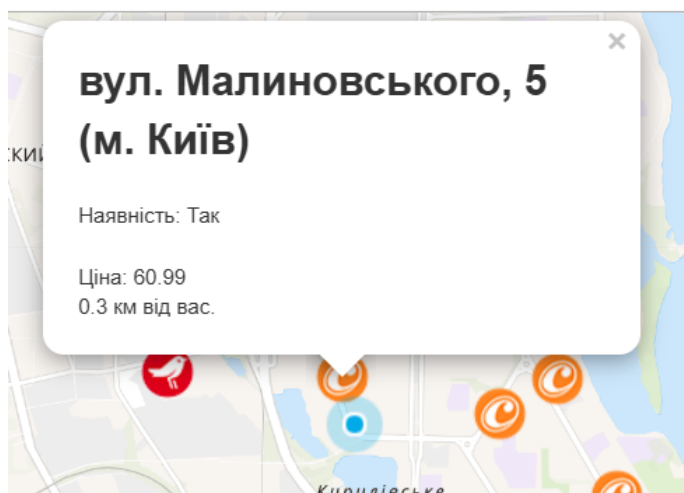


Рисунок 1.3 – Інформація з Supermarkets.com.ua

Також, під час аналізу я не знайшов можливості складати кошик товарів, та способу порівнювати ціну на рівні кошика.

1.2.2 Goods

Це мобільний застосунок, що дає змогу шукати товари серед різних торгових мереж, складати списки покупок, слідувати за акціями певних

					ІАЛЦ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		11

товарів та сканувати товари за штрихкодом. Застосунок підтримує такі мережі як Novus, Сільпо, АТБ, Варус та інші, що робить його одним з найбільш широко охоплених рішень на ринку.

Почнімо з пошуку. Як видно з Рисунку 1.4, пошукова видача є достатньо повною та містить різноманітні варіанти товарів. Проте насамперед варто зазначити, що реклама закриває до 50% екрану, що суттєво погіршує зручність використання та змушує користувача витратити додатковий час для перегляду результатів.

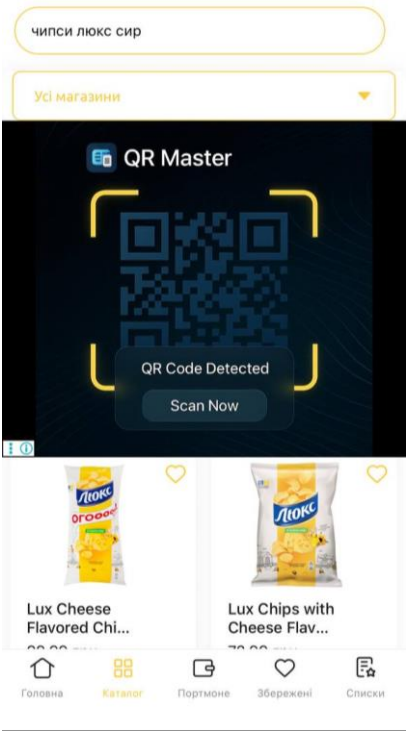


Рисунок 1.4 – Пошук у Goods

Далі можна перевірити "Списки" – це окрема сторінка застосунку Goods де можна формувати кошики товарів. Функціонал списків є корисним інструментом для планування покупок або відстеження акцій на певний асортимент.

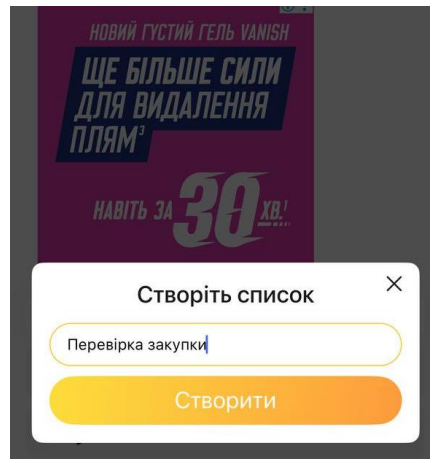


Рисунок 1.5 – Створення списку у Goods

Під час аналізу списків було помічено суттєве обмеження – створення списків можливе лише на рівні однієї торгової мережі, тобто порівнювати ціни на кошик товарів серед різних магазинів у такому режимі неможливо. Це є принциповим недоліком для користувача, який хоче знайти найвигідніший варіант покупок загалом, а не лише в межах однієї мережі.

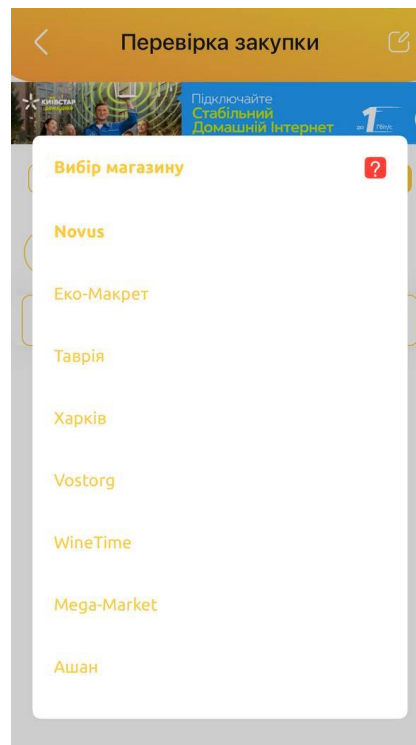


Рисунок 1.6 – Вибір магазину у Goods

Тепер перевіримо ціну. Варто зазначити, що перевірити наявність товару у конкретній фізичній торговій точці, на жаль, неможливо – застосунок працює на рівні асортименту конкретної мережі загалом, без можливості вибрати окремий магазин. Це є суттєвим обмеженням для користувачів, які планують покупки з урахуванням географічної доступності торгових точок.



Рисунок 1.7 – Кошик Novus у Goods

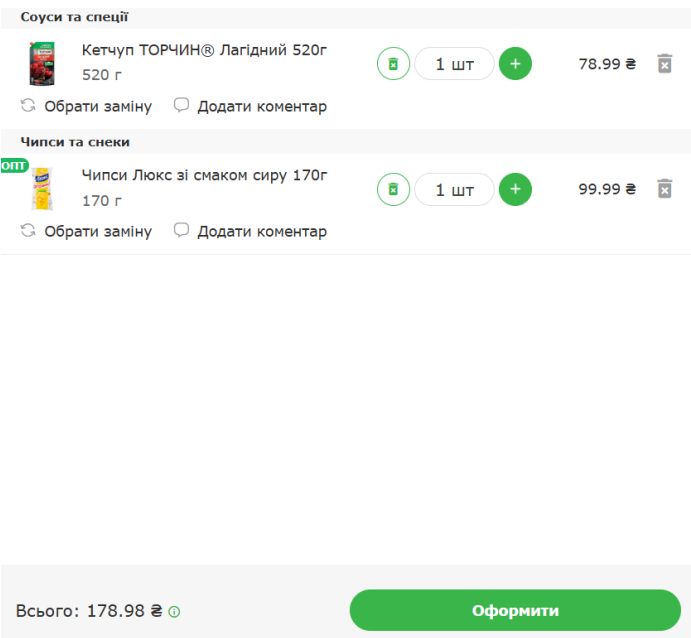


Рисунок 1.8 Перевірка цін Goods

Ціна збігається з реальною, що є безумовним плюсом та свідчить про те, що застосунок отримує актуальні дані від торгових мереж. Також серед переваг варто відзначити можливість сканувати штрихкоди товарів для їх швидкого пошуку – це зручна функція, яка прискорює процес додавання товарів до списку та є корисною при повторних покупках.

Окремо варто зупинитися на проблемі реклами. Інтерфейс застосунку перевантажений рекламними матеріалами – деякі банери перекривають до 50% пошукової видачі, що суттєво ускладнює перегляд результатів. Крім того, майже кожен третій клік супроводжується рекламним відео тривалістю 20–30 секунд, яке неможливо пропустити. Така кількість реклами негативно впливає як на загальний досвід користування, так і на швидкодію застосунку, що може відштовхнути частину аудиторії попри наявний корисний функціонал.

1.2.3 Де Дешевше

Мобільний сервіс, що є найбільш функціонально схожим до розроблюваної системи. Застосунок надає можливість шукати товари серед декількох магазинів, створювати списки товарів та визначати магазин з найнижчою ціною. Підтримувані мережі включають АТБ, Сільпо, Фора, Новус, Варус, ЕкоМаркет та інші великі роздрібні мережі України.

Основний плюс – можливість працювати з мапою, як показано на Рисунку 1.9. Користувач може вибрати радіус відносно своєї геолокації і здійснювати пошук лише серед магазинів у зоні доступності. Проте варто зазначити суттєве обмеження – неможливо самотійно обрати конкретні торгові точки, система автоматично визначає магазини в заданому радіусі без можливості ручного коригування вибору.

					ІАЛЦ.467200.003 ПЗ	Арк.
						15
Зм.	Арк.	№ докум.	Підпис	Дата		

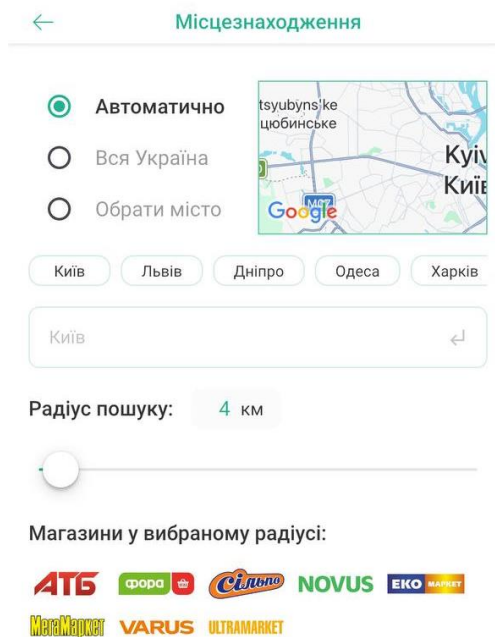


Рисунок 1.9 Робота з геолокації у “Де Дешевше”

З Рисунку 1.10 видно, що пошук працює коректно – відображається декілька пропозицій з різних мереж, а також наявні акційні ціни.

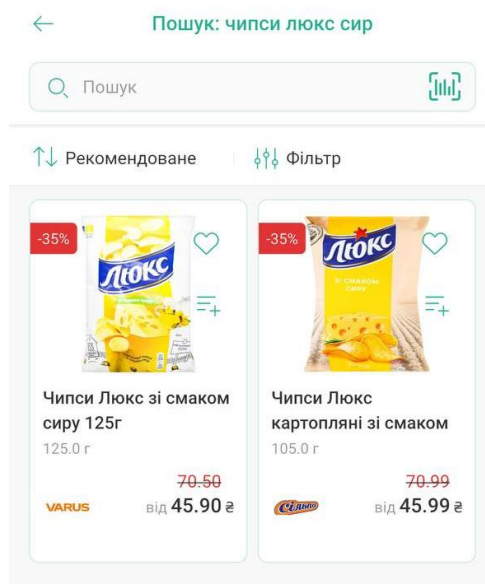


Рисунок 1.10 Пошук “Де Дешевше”

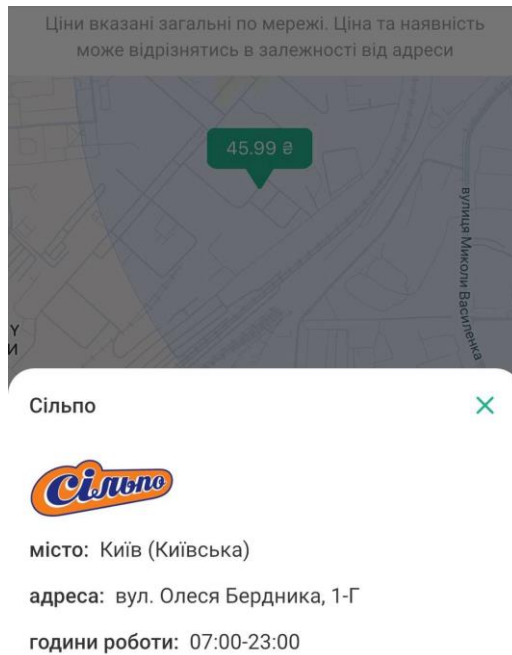


Рисунок 1.11 Ціна на “Чипси Люкс зі смаком сиру, 125г” у “Де Дешевше”

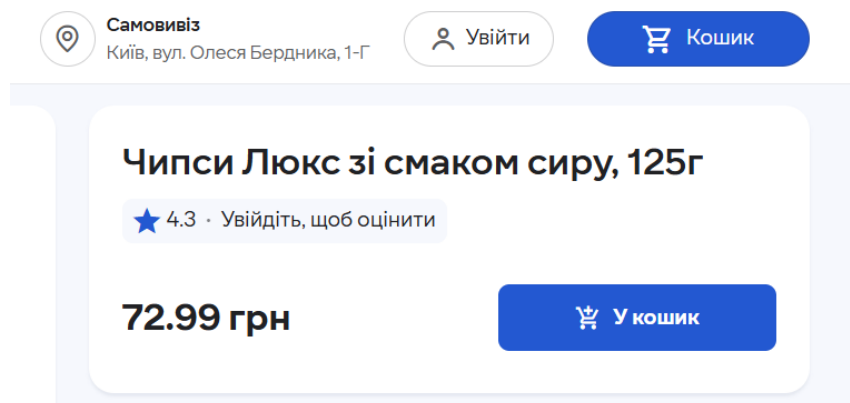


Рисунок 1.12 Ціна на “Чипси Люкс зі смаком сиру, 125г” у “Сільпо”

Як видно з Рисунків 1.11 та 1.12, ціна на однаковий товар "Чипси Люкс зі смаком сиру, 125г" за адресою Олесь Бердника, 1-Г відрізняється від реальної на офіційному вебсайті Сільпо, що свідчить про неактуальність даних та підкреслює обмеження системи, що не працює у реальному часі

					ІАЛЦ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		17

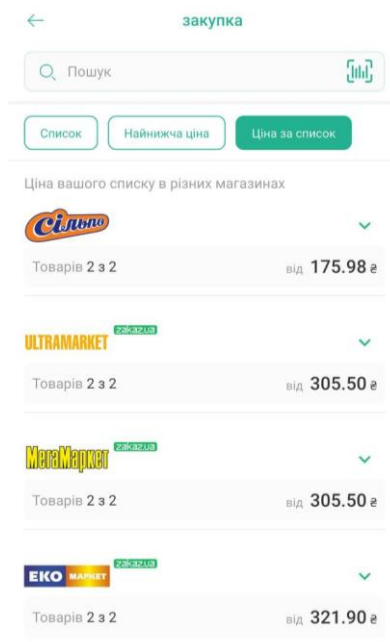


Рисунок 1.13 Порівняння кошика у різних торгових точках “Де Дешевше”

З Рисунок 1.13 можемо помітити основну перевагу серед інших існуючих рішень – порівняння цін у розрізі кошика одразу в декількох торгових точках. Цей функціонал є найбільш наближеним до системи, що розробляється у межах дипломної роботи, та демонструє правильний підхід до вирішення задачі порівняння цін.

Незважаючи на велику кількість переваг, у "Де Дешевше" також були помічені суттєві проблеми:

- Не можливо отримати фізичну адресу магазину, пошук на мапі працює на рівні одного товару, проте отримати адресу магазину в контексті кошика неможливо.
- Ціни не консистентні – оскільки пошук здійснюється в межах власної бази даних, а не через актуальні API торгових мереж, деякі товари мають неправильну ціну або відображають хибну наявність, що є критичним недоліком для користувачів, які покладаються на актуальність інформації.

1.3 Порівняльний аналіз існуючих рішень

Таблиця 1.1. – Порівняння існуючих рішень

Характеристика	Supermarkets.com.ua	Goods	Де Дешевше	GroceryScout
Пошук товарів у реальному часі	Ні	Ні	Ні	Так
Порівняння вартості кошика	Ні	Ні	Так	Так
Відображення конкретної адреси магазину	Частково	Ні	Ні	Так
Інтерактивна мапа магазинів	Так	Ні	Частково	Так
Перевірка наявності у магазині	Частково	Ні	Ні	Так
Підтримка кількох мереж	Так	Так	Так	Так
Фіксований каталог товарів	Так	Так	Так	Ні

Як видно із таблиці 1.1 існуючі рішення не можуть надати повного вирішення для задачі пошуку та порівняння цін з урахуванням усіх ключових вимог. Supermarkets.com.ua та Goods добре працюють на рівні одного продукта,

проте не підтримують порівняння на рівні кошика, що є однією з ключових потреб. Де Дешевше є найближчим аналогом, проте немає можливості отримати конкретну адресу магазину з найвигіднішою ціною.

Також одним з недоліків можна назвати жорстке порівняння товарів. Якщо користувачу не є критичним конкретний бренд, його можуть зацікавити аналоги з інших магазинів, таку гнучність у розроблюваному сервісі додає використання великих мовних моделей для нормалізації та класифікації товарів.

Водночас підхід пошуку у реальному часі, реалізований у GroceryScout, має власне обмеження – відсутність кешування результатів може негативно впливати на продуктивність системи при високому навантаженні, що є напрямком для подальшого вдосконалення.

					ІАЛЦ.467200.003 ПЗ	Арк.
						20
Зм.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВОК ДО РОЗДІЛУ

У першому розділі дипломної роботи було проведено комплексний аналіз предметної області, а саме – розглянуто агрегації та порівняння цін у роздрібній торгівлі, досліджено методи збору даних, методи захисту вебсервісів від збору даних, розглянуто існуючі програмні рішення.

По-перше, було встановлено, що українські роздрібні мережі формують різну цінову політику для однакових або аналогічних товарів, що створює потребу в автоматизованому порівняння цін. Також додаткову складність додає неузгодженість найменувань товарів між різними торговими мережами, що робить неможливим пряме текстове порівняння і потребує додатково застосування методів обробки природної мови.

По-друге, було розглянуто веб-скрапінг, як основний метод збору даних, а також проведено аналітику основних механізмів антибот-захисту, такі як: CAPTCHA, JavaScript-челенджі, аналіз HTTP-заголовків, TLS-фінгерпринтинг [17]. Встановлено, що більшість сучасних систем захисту поєднують усі перелічені підходи, що може суттєво ускладнювати автоматизований збір на сайтах, що мають захист.

Крім того було проведено порівняльний аналіз існуючих рішень: Supermarkets.com.ua, Goods та «Де Дешевше» - розроблені рішення мають схожий функціонал, справляються зі своєю задачею, але не задовольняють основних вимог. Зокрема жодне рішення не поєднує одночасно пошук товарів у реальному часі, порівняння вартості на рівні повного кошика покупок, відображення конкретної адреси торгової точки та перевірку наявності у магазині. Також існуючі рішення не підтримують пошук аналогів, що є цікавим для багатьох користувачів.

					ІАЛЦ.467200.003 ПЗ	Арк.
						21
Зм.	Арк.	№ докум.	Підпис	Дата		

Отже, результати проведеного аналізу свідчать про актуальність, доцільність та унікальність розробки системи GroceryScout, яка усуває виявлені недоліки завдяки пошуку в реальному часі, врахуванню геолокації та застосуванню великих мовних моделей для нормалізації та класифікації товарів.

РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ОГЛЯД ТЕХНІЧНИХ РІШЕНЬ

2.1 Формулювання задачі

Після аналізу предметної області та існуючих рішень і підходів до вирішення задачі агрегації та порівняння цін на товари, можна сформулювати основні вимоги для розроблюваної системи. Проведений аналіз показав, що більшість наявних рішень мають певні обмеження, такі як:

- Порівняння цін лише на рівні онлайн-магазину, без урахування фізичних торгових точок
- Відсутність гнучного пошуку та агрегації товарів
- Відсутність порівняння цін на рівні всього кошика, з подальшим визначенням найвигідної торгової точки

З урахуванням згаданих обмежень виникає потреба у створенні програмного сервісу, що здатен забезпечити гнучний пошук та актуальний підхід до агрегації товарів. У межах проєкту розробляється сервіс GroceryScout, який фокусується на агрегації даних з фізичних торгових точок, класифікації та подальшої обробки в реальному часі з урахуванням географічного розташування магазинів

Основна мета розробки – створення сервісу, що дозволяє користувачам виконувати пошук товарів одночасно у декількох торгових точках, отримувати актуальні ціни та інформацію про наявність, а також порівнювати загальну вартість кошика покупок між різними магазинами, з вибором найвигіднішого варіанту

Також, однією з ключових задач є проблема неузгодженості найменувань товарів, яка виникає через те що різні супермаркети мають власні формати назв,

					ІАЛЦ.467200.003 ПЗ	Арк.
						23
Зм.	Арк.	№ докум.	Підпис	Дата		

відрізняється порядок слів та зазначення якостей товарів. Для вирішення цієї проблеми передбачається використання методів обробки природньої мови та великих мовних моделей для групування однакових або аналогічних товарів.

2.2 Функціональні можливості системи

На основі сформульованих вимог та виявлених обмежень існуючих рішень визначено перелік функціональних можливостей сервісу GroceryScout.

2.2.1 Геолокаційний вибір торгових точок

Центральна точку входу користувача – інтерактивна мапа, на якій будуть зображені маркери фізичних торгових точок підтримуваних супермаркетів. Далі користувач самостійно обирає торгові точки у яких хотів би провести пошук та порівняння цін, що дає змогу врахувати критерії користувача до вибору магазину. Такий підхід істотно відрізняється від існуючих рішень, де пошук здійснюється в рамках торгової мережі без фактичної прив'язки до конкретного магазину.

2.2.2 Розподілений пошук товарів у реальному часі

Оскільки система працює у реальному часі усі пошукові запити мають відбуватись одночасно. Система звертається до пошукового АПІ кожного з торгової мережі з обраною адресою, таким чином дані будуть отримані безпосередньо з джерела, а отже актуальність ціни та інформація про наявність гарантована. Далі результати усіх джерел агрегуються та передаються на подальшу обробку.

					ІАЛЦ.467200.003 ПЗ	Арк.
						24
Зм.	Арк.	№ докум.	Підпис	Дата		

2.2.3 Нормалізація та групування товарів на основі LLM

Наступний крок у роботі системи – групування даних. Як було згадано раніше – кожна мережа має власний формат у найменуванні. Для вирішення цієї задачі буде використано мікросервіс з LLM, задачою якого є групування товарів. Налаштована модель аналізує характеристики з назв, такі як смак, бренд, жирність, спосіб обробки, маса та об’єм, і інші. Результат групування – структурований список нормалізованих позицій із відповідними пропозиціями кожного магазину та актуальними цінами.

2.2.4 Програмний інтерфейс доступу до даних

Доступ до даних та внутрішніх сервісів здійснюється за допомогою REST API, що дає змогу гнучко налаштовувати та користуватись системою, обмінюватися даними або підключати API до інших сервісів чи застосунків. Кожен ендпоінт відповідає за окрему функціональність – пошук товарів, отримання списку магазинів, формування кошика, що забезпечує чітке розмежування відповідальностей та спрощує подальше розширення функціоналу

2.2.5 Інтерфейс користувача

Для зручного користування на базі API побудовано веб-застосунок, що надає змогу взаємодіяти з усіма можливостями системи. Інтерфейс реалізує односторінокивий підхід до навігації, тобто взаємодія відбувається без перезавантаження сторінки, що забезпечує плавний та швидкий користувацький досвід.

Основні можливості веб-застосунку:

					ІАЛЦ.467200.003 ПЗ	Арк.
						25
Зм.	Арк.	№ докум.	Підпис	Дата		

- Перегляд інтерактивної мапи торгових точок та вибір магазинів для пошуку
- Здійснення пошуку товарів
- Додавання товарів у кошик з подальшим порівнянням загальної вартості
- Отримання даних про торгові точки та їх розташування

2.2.6 Адміністративна панель

Система містить адміністративну панель на базі PocketBase, що надає можливість керувати налаштуваннями без перезапуску сервісів. Через адмін-панель адміністратор може вмикати та вимикати підтримувані торгові мережі, змінювати API-ключі зовнішніх сервісів, налаштовувати параметри LLM-моделі та редагувати промпти для нормалізації товарів. Зміни застосовуються миттєво завдяки механізму оновлення конфігурації в реальному часі – сервіси підписані на зміни через SSE і отримують оновлені налаштування без перезапуску.

2.3 Технічні вимоги системи

Технічні вимоги, на відміну від функціональних вимог, описують роботу системи з точки зору продуктивності, надійності, масштабованості.

Швидкодія – одна з основних вимог розроблюваної системи. Через те що пошук здійснюється у реальному часі з паралельними зверненнями до зовнішніх АПІ торгових мереж, архітектура повинна забезпечувати асинхронне виконання запитів з мінімальними затримками. Модуль нормалізації на основі

					ІАЛЦ.467200.003 ПЗ	Арк.
						26
Зм.	Арк.	№ докум.	Підпис	Дата		

LLM також є частиною пошукового ланцюжка. Разом пошук та нормалізація в нормальних умовах повинні займати до 2-3 секунд.

Надійність системи забезпечується стійкістю до відмов зовнішніх АПІ. Досягти надійності можна за допомоги повторення запитів, перестворення сесій, ротації проксії та обходу антибот-систем. Система повинна пропускати джерела даних, що відповідають занадто довго або не працюють за інших обставин. Більшість з цих вимог на стороні досягаються на стороні HTTP-клієнта.

Не менш важливою вимогою є масштабованість, яка досягається завдяки модульній архітектурі, мікросервісам та розподіленій обробці задач, що включають у себе парсинг даних. Додавання нових торгових мереж має не потребувати змін у загальній системі, а доповнювати існуючу, так само як і вимкнення торгових мереж не повинно впливати на роботу інших частин системи. Масштабування з точки зору пропускної здатності досягається нарощенням воркерів та збільшенням пулу проксі.

Зручність розгортання системи досягається з використанням системи контейнеризації Docker, що дозволяє легко розгортатись на нових пристроях та спрощує запуск системи.

Безпека конфігурації забезпечується зберіганням чутливих даних, таких як API-ключі та токени, в адміністративній панелі з обмеженим доступом за паролем, що виключає необхідність зберігати секрети безпосередньо у конфігураційних файлах середовища.

2.4 Мікросервісна архітектура

Мікросервісна архітектура [20] – це підхід розробки програмного забезпечення, де застосунок розбивається на набір невеликих незалежних сервісів, кожен з яких відповідає за окремий функціонал та взаємодіє з іншими

					ІАЛЦ.467200.003 ПЗ	Арк.
						27
Зм.	Арк.	№ докум.	Підпис	Дата		

через API, брокери або інші мережеві механізми. На відміну від монолітного підходу, де всі компоненти існують в єдиному процесі, мікросервіси можна розробляти, розгортати та масштабувати незалежно один від одного.

Система GroceryScout побудована за цим принципом та складається з чотирьох незалежних сервісів. Основний API сервіс відповідає за обробку запитів користувача, взаємодію з інтерфейсом та координацію роботи системи. Воркери Taskiq виконують паралельні запити до API торгових мереж та є окремими процесами, кількість яких можна нарощувати незалежно. LLM мікросервіс відповідає виключно за нормалізацію та групування товарів – він отримує результати пошуку та повертає структуровані дані. Redis виступає спільним брокером між сервісами – зберігає черги задач та результати їх виконання.

Такий підхід дає ряд практичних переваг: вимикати окремі торгові мережі можна без зупинки всієї системи, кількість воркерів можна нарощувати при збільшенні навантаження, а LLM мікросервіс можна замінити або оновити без впливу на основний API сервіс. Додатково система містить адміністративну панель на базі PocketBase, через яку можна керувати конфігурацією всіх сервісів у реальному часі без перезапуску.

Водночас мікросервісна архітектура має і свої недоліки – складніше налагодження, накладні витрати на мережеву взаємодію між сервісами та необхідність оркестрації контейнерів, що вирішується використанням Docker Compose.

2.5 Аналіз методів автоматизованого збору даних

Автоматизований збір даних – ключова вимога у розроблюваній системі, отримання коректних цін та інформації про наявність є найбільш вагомим критерієм для оцінювання роботи системи. Більшість торгових мереж не

					ІАЛЦ.467200.003 ПЗ	Арк.
						28
Зм.	Арк.	№ докум.	Підпис	Дата		

надають публічного АПІ для отримання даних, тому всі дані отримуються через взаємодію з внутрішніми АПІ або HTML-сторінками, які далі парситимуться через відповідні бібліотеки.

Основні способи отримання даних:

- Робота з внутрішніми АПІ сервісів та використання їх через HTTP-клієнти
- Використання браузерної автоматизації, наприклад Playwright або Selenium. Використовується у сильно захищених системах, або системах що потребують велику кількість інтерактивних дій. У таких випадках використання HTTP-клієнта є клопіткою роботою і має багато критичних точок, наприклад зміна одного запита ламає весь процес збору.
- Комбінований – отримання сесій, авторизація, складні дії та проходження CAPTCHA або JavaScript-челенджів відбувається на стороні браузерної автоматизації. Далі використовується HTTP-клієнт з налаштованою сесією, що буде в рази швидше за браузер

У розроблюваній системі браузерна автоматизація – останній порог, який ми хочемо уникати, оскільки він сильно сповільнює та ускладнює систему, тому нам потрібен потужний HTTP-клієнт який зможе закрити 90% кейсів.

2.5.1 Порівняльний аналіз HTTP-клієнтів

Як було описано раніше – хороший HTTP-клієнт спростить розробку системи у рази.

requests – найпопулярніша бібліотека для виконання HTTP-запитів у Python. Проста й інтуїтивно зрозуміла бібліотека, широка документація та

					ІАЛЦ.467200.003 ПЗ	Арк.
						29
Зм.	Арк.	№ докум.	Підпис	Дата		

мінімальна кількість багів. Проте для задач веб-скрапінгу це одне з найгірших рішень.

По-перше, через те що requests стара бібліотека, вона має лише синхронне АПІ, що змушує розробника використовувати потоки замість асинхронних задач – важче масштабувати та контролювати потік запитів.

По-друге, requests використовує під капотом urllib3, що в свою чергу використовує OpenSSL. У OpenSSL не можливо або дуже складно налаштувати TLS, через що усі запити мають одноманітний відбиток і будуть блокуватись системами анти-бот захисту. Приклад такого захисту буде продемонстровано у Розділі 3.

httpx – сучасна альтернатива, головна перевага над requests – асинхронне АПІ, що робить її природним вибором для асинхронного Python. Бібліотека підтримує HTTP/2, має зручний інтерфейс. Проте httpx також використовує стандартний TLS-стек, що робить її вразливою до TLS-фінгерпринтингу.

curl_cffi – принципово інше рішення, це Python-обгорта над бібліотекою libcurl за допомогою CFFI. CFFI – це інша бібліотека яка дає змогу викликати С-код без зайвих складнощey. Ця бібліотека використовує BoringSSL, яким також користується кожен Chrome браузер, тобто на найнижчому рівні ми уже маємо максимальну схожість з справжнім браузером. Також curl_cffi має 20-30 готових профілей “імперсонеїв” – готових налаштувань TLS (наприклад Chrome 135, Chrome 121, Firefox 145 і тд.), порядку заголовків, автоматична генерація Client Hints заголовків, що дає широкий спектр профілів для різноманітності та підготовлює запити виглядати найбільш схоже до справжнього браузера.

Для останньої частини порівняння технологій проведемо тестування швидкодії при різних рівнях конкурентності.

					ІАЛЦ.467200.003 ПЗ	Арк.
						30
Зм.	Арк.	№ докум.	Підпис	Дата		

Порівняння HTTP-клієнтів Python: requests vs httpx vs curl_cffi
Тестовий payload ~200 KB, середнє з 3 запусків

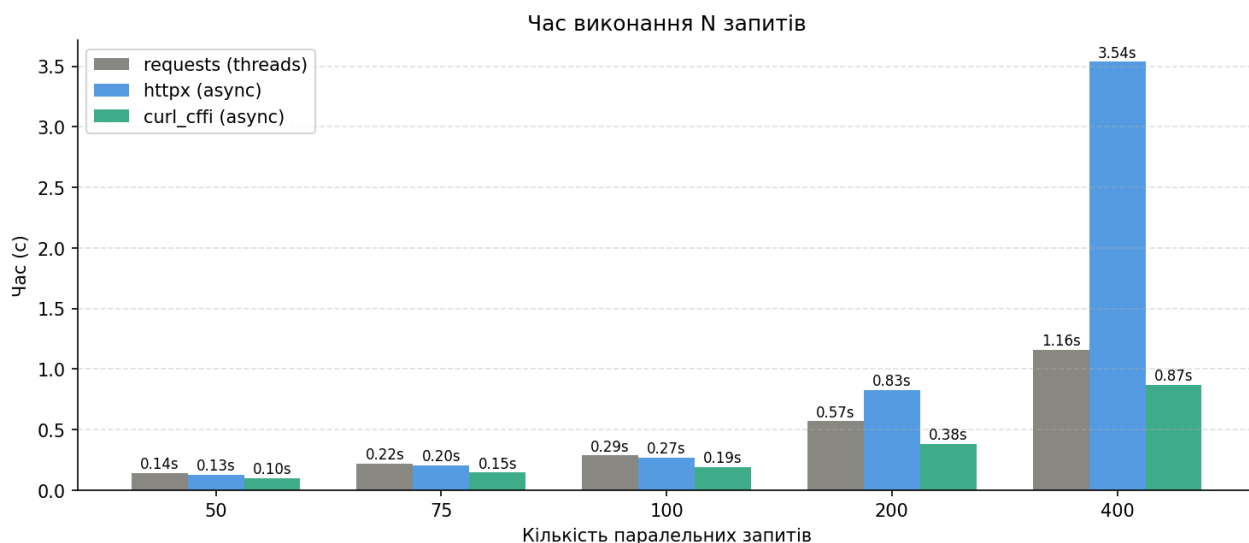


Рисунок 2.1 – Порівняння часу виконання HTTP-запитів бібліотеками requests, httpx та curl_cffi при різних рівнях паралельності

І як видно з Рисунок 2.1. curl_cffi лідирує у кожному тестуванні, при чому продуктивність httpx деградує при 200 одночасних запитах, а requests та curl_cffi добре себе показують при високій конкурентності. curl_cffi стабільно на 30% швидше за requests.

2.5.2 Таблиця порівняння HTTP-клієнтів

Таблиця 2.1 – Порівняння HTTP-клієнтів для Python

Характеристика	requests	httpx	curl_cffi
Асинхронне API	Ні	Так	Так
HTTP/2	Ні	Так	Так
TLS-стек	OpenSSL	OpenSSL	BoringSSL
Імітація браузерного TLS	Ні	Ні	Так

Кінець таблиці 2.1

Готові профілі браузерів	Ні	Ні	Так (20+)
Client Hints заголовки	Ні	Ні	Так
Продуктивність при високій паралельності	Середня	Низька	Висока

Після порівняння та проведених тестувань можна впевнено сказати, що `curl_cffi` є єдиним рішенням [3], що найкраще підійде до розроблюваної системи. Серед основних переваг – асинхронне АПІ, використання BoringSSL замість OpenSSL, що робить запити схожими на браузерні, 20-30 готових профілей з TLS налаштуванням, стабільна продуктивність при високому навантаженні – на 30% швидше за `requests`. Отже, `curl_cffi` обрано як основний HTTP-клієнт для реалізації модуля збору даних у системі GroceryScout.

2.6 Методи виявлення однакових товарів у різних торгових мережах

Виявлення однакових товарів у різних торгових мережах є однією з ключових вимог у сервісі що розробляється. Один й той самий продукт може мати відмінну назву через певні формати у торгових мережах: скорочення, зміна порядку слів. Це робить пряме текстове порівняння не ефективним.

Перший спосіб – регулярні вирази, нормалізація рядків, видалення зайвих символів. Хоч цей спосіб є доволі швидким та може закривати певні кейси, для семантичного аналізу він підійде погано. Таким чином буде важко відрізнити товари за вагою, об'ємом і іншими властивостями, наприклад - жирність для молока.

Більш продвинутий варіант – використання векторних представлень та семантичний пошук. Найменування товарів перетворюються у числові вектори

за допомогою LLM, далі зіставлення відбувається на основі подібності між векторами. Цей підхід добре справляється з семантичними варіаціями, проте потребує конфігурації, оскільки однакові товари за змістом, але відмінні за характеристиками будуть розпізнаватись як однакові. Також цей спосіб ускладнює обробку даних – потрібно також зберігати властивості як вага, жирність, об’єм, смак для кращого аналізу, що може ускладнити розробку.

Найбільш гнучкий підхід – використання LLM для класифікації та зіставлення товарів. Якщо задати LLM якісні пояснення як класифікувати товари та розуміти відмінності можна з високою точністю розрізнити товари. Мовна модель з легкістю зрозуміє “Молоко Яготинське 2.6% 900г” та “Яготинське молоко 2,6% 0.9кг” як один товар, а “Молоко Яготинське 3.2% 900г” класифікує як інший продукт

Для реалізації GroceryScout було використано саме останній підхід, а LLM винесено у окремий мікросервіс, для більшої гнучкості. Також для більшої гнучкості та незалежності використано бібліотеку LiteLLM, що дає змогу користуватись різними LLM без змін у кодовій базі.

Для забезпечення якості класифікації розроблено систему регресійного тестування промптів, що дозволяє оцінювати точність групування на наборі тестових випадків та порівнювати результати при зміні моделі або промπτу.

2.7 Розподілена обробка запитів

Як було описано раніше – паралельна робота з АПІ декількох торгових мереж є ключовою вимогою, що гарантує швидкодію системи. Також велике значення має підхід до реалізації паралельної обробки, оскільки це впливає на масштабованість, надійність та роботу системи в цілому.

Найпростіший спосіб, який йде разом з асинхронним Python – `asyncio.gather`. Це спосіб конкурентно виконувати корутини та дочекатись усіх

					ІАЛЦ.467200.003 ПЗ	Арк.
						33
Зм.	Арк.	№ докум.	Підпис	Дата		

результатів у межах одного процесу. Такий варіант достатній для прототипу та базових сценаріїв, проте він має суттєві обмеження – всі задачі виконуються в одному процесі, що робить горизонтальне масштабування неможливим та не відповідає вимогам розподілених обчислень, тому що робота у межах одного процесу.

Більш масштабований спосіб – використовувати системи черги задач. Кожен запит є окремою задачею, яка існує в черзі задач, незалежні воркери беруть задачу з черги та виконують її, це дає змогу запускати воркерів на різних пристроях і робить обробку справді розподіленою. Брокером виступає Redis, як стабільне рішення, проте без гарантії послідовності, але у нашій задачі це не є критичним.

Серед існуючих бібліотек для Python є Celery, ARQ, Taskiq. Celery – одне з найкращих рішень, перевірене роками і з широким функціоналом, проте він має лише синхронне АПІ, що робить роботу з асинхронними бібліотеками складною та заважаною. ARQ – відносно нове рішення з підтримкою асинхронного `asyncio`, проте бібліотека не популярна, рідко оновлюється. Taskiq [2] – сучасна бібліотека з підтримкою `asyncio`, наближена до Celery та легко інтегрується у FastAPI.

У системі GroceryScout кожне звернення до API окремої торгової точки є окремою задачею Taskiq, що дозволяє паралельно обробляти запити до різних мереж та агрегувати результати після завершення всіх задач. Отже, для реалізації розподіленої обробки запитів обрано Taskiq та Redis – нативна підтримка асинхронного API робить взаємодію з `curl_cffi` та іншими асинхронними компонентами природною.

2.8 Обґрунтування вибору технологічного стеку

Для реалізації проєкту GroceryScout було обрано технологічний стек, що забезпечує швидкодію, розподілену обробку запитів, стійкість до антибот-захисту та зручне розгортання. Вибір кожного компонента обґрунтовано функціональними та технічними вимогами системи.

2.8.1 Python

Обрана мова програмування – Python [10], версія 3.10 і вище. Це інтерпретована мова програмування, що була створена у 1991 році. Починаючи з версії 3.4 Python отримав вбудовану підтримку асинхронного програмування через модуль `asyncio`. Загалом Python зараз одна з найпопулярніших мов програмування, має свої обмеження як-от GIL (Global Interpreter Lock), що обмежує паралельне виконання потоків, проте для I/O bound задач, як мережеві запити це не є критичним через модуль `asyncio`. Python закріпив своє місце у багатьох нішах: веб-розробка, аналіз даних, роботі зі штучним інтелектом – завдяки простоті синтаксису, швидкості розробки та можливості написання модулів на C чи Rust, що дає Python змогу обходити свої обмеження - наприклад `curl_cffi` [3], що використовується у даному проєкті, є саме таким прикладом: Python-інтерфейс над високопродуктивною C-бібліотекою `libcurl`.

Python має величезну екосистему бібліотек - для веб-скрапінгу, асинхронної обробки, роботи з базами даних та інтеграції з LLM API. Для задач що поєднують збір даних та штучний інтелект Python є де-факто стандартом у галузі.

					ІАЛЦ.467200.003 ПЗ	Арк.
						35
Зм.	Арк.	№ докум.	Підпис	Дата		

2.8.2 FastAPI

Основний фреймворк для HTTP-сервера – FastAPI [1]. Це сучасний веб-фреймворк для Python, який побудовано на основі Starlette як ASGI-сервера та Pydantic для валідації даних. ASGI (Asynchronous Server Gateway Interface) - це стандарт взаємодії між веб-сервером та Python застосунком, що на відміну від старішого WSGI стандарту підтримує асинхронну обробку запитів.

FastAPI вирізняється своїми можливостями, насамперед він автоматично генерує OpenAPI документацію на основі анотації типів, що прискорює розробку та тестування API. Вбудована підтримка Dependency Injection дозволяє гнучко керувати залежностями між компонентами. Такі можливості стали новим стандартом і нові веб-фреймворки додають їх у першу чергу, оскільки це дійсно зручно. Крім того FastAPI показує себе як один з найшвидших веб-фреймворків на Python.

2.8.3 curl_cffi

Для виконання HTTP-запитів до API торгових мереж обрано бібліотеку curl_cffi [3]. curl_cffi є Python-обгорткою над бібліотекою libcurl - однією з найпоширеніших бібліотек для передачі даних через мережу, що використовується у мільярдах пристроїв по всьому світу. Взаємодія з С-кодом libcurl реалізована через CFFI (C Foreign Function Interface) - механізм виклику С-функцій безпосередньо з Python без написання С-розширень.

На відміну від requests та httpx, curl_cffi використовує BoringSSL - форк OpenSSL розроблений Google, що є основою TLS-стеку браузера Chrome. Це дозволяє відтворювати точні параметри TLS-з'єднання реальних браузерів, включаючи набір шифрів, розширення та їх порядок. Бібліотека надає 20-30 готових профілів імітації браузерів - Chrome 135, Firefox 128 та інших - що дає

					ІАЛЦ.467200.003 ПЗ	Арк.
						36
Зм.	Арк.	№ докум.	Підпис	Дата		

широкий спектр варіантів для ротації відбитків між запитами. Як було підтверджено власним бенчмарком у розділі 2.4, `curl_cffi` демонструє найвищу продуктивність серед розглянутих альтернатив та обрана як основний HTTP-клієнт системи.

2.8.4 Redis

Redis [6] – надшвидке сховище даних за принципом “ключ-значення”, що живе у оперативній пам’яті. Завдяки роботі у оперативній пам’яті Redis забезпечує неймовірно низьку затримку операцій читання / запису, що робить його ідеальним для задач де швидкість є критичною. Redis підтримує різноманітні структури даних: рядки, списки, множини, хеш-таблиці, черги повідомлень – це дає змогу використовувати його у різних сценаріях. У системі GroceryScout Redis виконує роль спільного брокера між АПІ сервісом та воркерами Taskiq – зберігає черги задач та результати їх виконання.

2.8.5 Taskiq

Для виконання розподіленої обробки запитів використано Taskiq [2]. Це сучасна бібліотека для розподіленої обробки задач у Python, що нативно підтримує асинхронний код, чого наприклад Celery не вистачає. Принцип роботи базується на патерні “producer-consumer” – АПІ сервіс додає задачі у чергу, незалежні воркери слухають чергу та виконують задачі паралельно. Taskiq надає зручну інтеграцію з FastAPI через lifespan контекст, що дозволяє коректно ініціалізувати та завершувати з’єднання з брокером разом із запуском сервера.

					ІАЛЦ.467200.003 ПЗ	Арк.
						37
Зм.	Арк.	№ докум.	Підпис	Дата		

2.8.6 LiteLLM та Groq API

Для інтеграції з LLM API обрано LiteLLM [4] – зручну бібліотеку з уніфікованим інтерфейсом, який дає змогу використовувати різні провайдери LLM, що дозволяє швидко міняти моделі без переписування коду та дає змогу додати fallback механізми, якщо якась модель перестане працювати з яких-небудь причин. Groq API [19] обрано через низьку затримку та існування безкоштовного плану, що достатньо для демонстрації роботи сервісу, проте за потреби провайдера можна буде легко змінити.

2.8.7 Leaflet.js

Leaflet.js [8] – це відкрита JavaScript бібліотека для роботи з інтерактивними картами, що з'явилась у 2011 році та стала одним з найпопулярніших рішень у своїй категорії. Бібліотека відзначається малим розміром – близько 42KB у стиснутому вигляді – що забезпечує швидке завантаження сторінки. Leaflet.js підтримує будь-якого провайдера картографічних тайлів, включаючи OpenStreetMap [11] як безкоштовну альтернативу комерційним рішенням на кшталт Google Maps. Бібліотека надає простий API для роботи з маркерами, подіями кліку та шарами карти, що дозволяє реалізувати інтерактивний вибір торгових точок з мінімальною кількістю коду.

2.8.8 Docker та Docker Compose

Для того щоб зробити процес розгортання зручним та гнучким обрано Docker та Docker Compose [9]. Docker – це платформа для контейнеризації застосунків, що дозволяє упакувати програму разом з усіма її залежностями у

					ІАЛЦ.467200.003 ПЗ	Арк.
						38
Зм.	Арк.	№ докум.	Підпис	Дата		

ізолюваний контейнер. До речі неправильно вважати Docker аналогом віртуальної машини, оскільки він використовує ядро операційної системи хосту, але робить це все прихованим від справжньої ОС – має власну файлову систему та простір процесів. Це гарантує, що застосунок працюватиме однаково на будь-якому хості незалежно від встановленого програмного забезпечення.

Docker Compose – це інструмент опису та запуску декількох контейнерів через єдиний файл конфігурації, що дозволяє розгорнути систему однією командою.

Також контейнеризація захищає всю операційну систему від взлому – ураження одного сервісу не впливає на інші. Проте це не знімає відповідальності за безпеку – оскільки за певних обставин вийти з контейнера у справжню операційну систему можливо.

2.8.9 PostgreSQL

Для зберігання даних користувачів обрано PostgreSQL [16] – одну з найпопулярніших реляційних баз даних з відкритим вихідним кодом. PostgreSQL відзначається надійністю, підтримкою складних запитів та розширеними типами даних. У системі GroceryScout PostgreSQL використовується для зберігання облікових записів користувачів – електронної пошти та хешованого пароля. Взаємодія з базою даних реалізована через asynpg [7] – найшвидший асинхронний драйвер PostgreSQL для Python, що нативно підтримує asyncio та забезпечує мінімальні затримки при роботі з базою даних. Схема бази даних ініціалізується автоматично при першому запуску сервісу, що спрощує розгортання системи.

2.8.10 PocketBase

PocketBase [5] – це бекенд-платформа з відкритим вихідним кодом, написана на Go, що поставляється як єдиний виконуваний файл. Основна перевага – готова платформа для побудови колекцій з вбудованим адміністративним інтерфейсом та REST API, що не потребує додаткового налаштування.

У системі GroceryScout PocketBase виконує роль адміністративної панелі для зберігання конфігурації та змінних середовища – API-ключі зовнішніх та внутрішніх сервісів, налаштування LLM-моделей, список активних торгових мереж та їх метадані. Хоча PocketBase має вбудований функціонал для роботи з користувачами, від нього було вирішено відмовитись – під капотом PocketBase працює з SQLite та вимагає використання власного API без прямого контролю над запитами, що є небажаним для задачі де потрібна повна гнучкість. Для зберігання користувачів обрано PostgreSQL з прямим доступом через asynsrg.

Отримання конфігурації сервісами відбувається через SSE-підписку на зміни колекцій PocketBase, що дає змогу завжди мати актуальний стан конфігурації та змінювати налаштування системи без перезапуску будь-якого сервісу.

					ІАЛЦ.467200.003 ПЗ	Арк.
						40
Зм.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВОК ДО РОЗДІЛУ

У другому розділі було сформульовано постановку задачі для розробки сервісу GroceryScout, визначено функціональні та технічні вимоги до системи, було проведено огляд та порівняльний аналіз технічних рішень для кожного з ключових компонентів.

Після аналізу існуючих обмежень сформульовано основну мету розробки – створення сервісу, що здатен виконувати пошук товарів у реальному часі одночасно у декількох фізичних торгових точках, що також включає класифікацію товарів між різними торговими точками, порівняння вартості товару та кошика. Визначено функціональні можливості системи: інтерактивна мапа з вибором торгових точок, розподілений пошук у реальному часі, групування товарів на основі LLM, REST API та веб-інтерфейс користувача. Сформульовано технічні вимоги щодо швидкодії, надійності, обходу антибот-систем, масштабованості та зручності розгортання.

Під час огляду методів збору даних визначено найефективніший підхід – пряме звернення до внутрішніх API торгових мереж через HTTP-клієнтів. Для того щоб уникати браузерної автоматизації, яка в рази повільніша за HTTP-клієнт було проведено аналіз та власне тестування бібліотек requests, httpx, та curl_cffi. У результаті – curl_cffi показав себе як найкраще рішення, що поєднує асинхронне API, використання BoringSSL для імітації браузерного TLS та найвищу продуктивність при масштабуванні.

Було проаналізовано підходи до виявлення однакових товарів між торговими мережами – на основі regex, векторних представлень та LLM. Було обрано підхід з прямим використанням LLM, як найбільш точний та гнучкий, що не потребує додаткових дій для додавання нових торгових мереж.

Для реалізації розподіленої обробки запитів було обрано Redis як брокер, та Taskiq як бібліотеку для виконання розподілених задач. Taskiq дає змогу

					ІАЛЦ.467200.003 ПЗ	Арк.
						41
Зм.	Арк.	№ докум.	Підпис	Дата		

нативно користуватись асинхронним АПІ та легко інтегрується з FastAPI, curl_cffi.

Архітектурним рішенням системи обрано мікросервісний підхід, що забезпечує незалежне масштабування кожного компонента, динамічне вмикання та вимикання торгових мереж без перезапуску системи, а також можливість оновлення окремих сервісів без впливу на роботу інших. Керування конфігурацією всіх сервісів у реальному часі реалізовано через адміністративну панель PocketBase з SSE-підпискою на зміни.

Також у ході аналізу технологічного стеку було обрано:

- Python 3.10 і вище – як основну мову розробки
- FastAPI – як швидкий веб-фреймворк
- curl_cffi – як найефективніший HTTP-клієнт з готовими TLS-профілями
- Redis – брокер та тимчасове сховище даних для мікросервісів
- Taskiq – менеджер задач з асинхронним АПІ
- LiteLLM – зручний та уніфікований інтерфейс для роботи з LLM
- Leaflet.js – швидка та зручна бібліотека для реалізації інтерактивної мапи
- Docker / Docker Compose – для гнучкості та зручності розгортання мікросервісів
- PocketBase – адміністративна панель для керування конфігурацією системи в реальному часі
- Docker / Docker Compose – для гнучкості та зручності розгортання мікросервісів

Отже, сформовані вимоги та обраний технологічний стек створюють надійне технічне підґрунтя для подальшого проектування архітектури системи та реалізації сервісу GroceryScout.

					ІАЛЦ.467200.003 ПЗ	Арк.
						42
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

3.1 Загальна архітектура системи

Система GroceryScout побудована за мікросервісною архітектурою та розгортається за допомогою Docker Compose як набір ізольованих контейнерів. Система складається з трьох власних мікросервісів та трьох інфраструктурних компонентів, кожен з яких має чітко визначену зону відповідальності.

у та актуальними цінами.

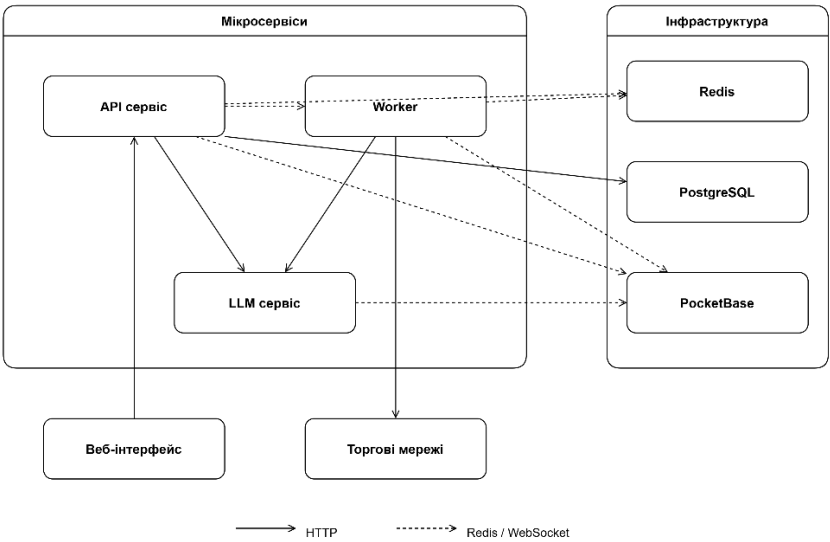


Рисунок 3.1 Візуалізація архітектури

Серед власних мікросервісів – API сервіс, воркери Taskiq та LLM match сервіс. API сервіс – це основний вузол системи, що обробляє HTTP-запити від веб-інтерфейсу, створює задачі для воркерів та агрегує результати пошуку та порівняння. Воркери – окремі процеси, що слухають чергу на наявність задач та використовуються конкретні клієнти торгових мереж для пошуку, масштабувати воркерів можна незалежно від усієї системи, що здійснюється

нарощенням кількості процесів. LLM match сервіс відповідає лише за класифікацію та групування товарів.

Серед інфраструктурних компонентів – Redis як брокер задач та тимчасове сховище результатів, PostgreSQL – основна база даних для збереження даних користувачів та PocketBase як адміністративна панель та джерело конфігурації для мікросервісів.

3.1.1 Взаємодія компонентів

Типовий сценарій пошуку товарів відбувається наступним чином. Користувач обирає магазини на інтерактивній мапі та надсилає пошуковий запит. API сервіс отримує запит, перевіряє які торгові мережі увімкнені через ShopRegistry та для кожного обраного магазину додає окрему задачу до черги Redis. Воркери паралельно виконують задачі – кожен звертається до API своєї торгової мережі через `curl_cffi`. Результати зберігаються в Redis та агрегуються API сервісом після завершення всіх задач. Агреговані дані передаються до LLM сервісу для нормалізації та групування. Фінальний результат повертається користувачу.

Конфігурація всіх сервісів керується через PocketBase – при старті кожен мікросервіс підключається через SSE та отримує актуальні налаштування. При зміні конфігурації в адмін-панелі всі сервіси отримують оновлення миттєво без перезапуску.

3.1.2 Спільна бібліотека та динамічна конфігурація

Оскільки система складається з кількох незалежних мікросервісів, виникає потреба у спільному коді який доступний кожному з них. Для цього виділено директорію *libs* що монтується як Docker volume у кожен контейнер.

					ІАЛЦ.467200.003 ПЗ	Арк.
						44
Зм.	Арк.	№ докум.	Підпис	Дата		

Це дозволяє уникнути дублювання коду та забезпечує єдину точку оновлення спільної логіки.

Центральним компонентом бібліотеки є `pb_client.py` – модуль що реалізує два класи: `PocketBaseClient` для взаємодії з PocketBase API та `RuntimeSettings` для зберігання поточної конфігурації в пам'яті.

`RuntimeSettings` – це простий контейнер що зберігає налаштування як словник та надає метод `get` для їх отримання. Кожен мікросервіс має власний екземпляр `runtime` що заповнюється при старті.

```
class RuntimeSettings:
    def __init__(self) -> None:
        self._data: dict = {}

    def load(self, settings: dict) -> None:
        self._data = settings

    def get(self, key: str, default: str = "") -> str:
        return self._data.get(key, default)

    def __getitem__(self, key: str) -> str:
        return self._data[key]

runtime = RuntimeSettings()
```

Рисунок 3.2 клас `RuntimeSettings`

При старті кожного мікросервісу викликається функція `start_pb` – вона авторизується в PocketBase, завантажує всі налаштування з колекції `settings` у `runtime` та запускає фонову задачу підписки на зміни.

```

async def start_pb() -> RuntimeSettings:
    pb = PocketBaseClient()
    await pb.auth()
    settings = await pb.get_settings()
    runtime.load(settings)

    async def _realtime_loop() -> NoReturn:
        while True:
            try:
                await pb.subscribe()
            except Exception as e:
                logger.warning("Realtime reconnecting: %s", e)
                await asyncio.sleep(5)

    asyncio.create_task(_realtime_loop())
    return runtime

```

Рисунок 3.3 функція start_pb

Підписка реалізована через SSE – мікросервіс підключається до ендпоінту /api/realtime [18] PocketBase та отримує події при будь-якій зміні в колекціях settings або brands. При отриманні події відповідний ключ в runtime оновлюється миттєво. Якщо з'єднання обривається – фонові задача автоматично перепідключається через 5 секунд. Таким чином зміна будь-якого налаштування в адмін-панелі PocketBase – API ключа, промпту, списку активних брендів – миттєво відображається у всіх мікросервісах без перезапуску.

3.1.3 ShopRegistry

Реєстр торгових мереж – центральний компонент пошукової системи, він агрегує та визначає робочі торгові мережі. Основна ідея – додавання торгової мережі має відбуватись при мінімальних змінах у коді, а можливість вмикати та вимикати мережі має відбуватись без перезапуску системи. Такі можливості досягаються у поєднанні PocketBase SSE із зручною реалізацією реєстру.

Edit **settings** record

Ow id

hmvqva6u4cniiya

T key

enabled_brands

T value

atb,varus,fora

Рисунок 3.2 Адміністративна панель, ключ enabled_brands

Реєстр є класом ShopRegistry, що автоматично зчитує з PocketBase увімкнені мережі. Далі для кожної увімкненої мережі, він зчитує файл *{brand}_shops.json*, що містить усі фізичні точки з такими параметрами як short_name, lat, lon, id. Далі ShopRegistry надає інтерфейс з методами, за якими можна дістати усі увімкнені торгові мережі, отримати Taskiq задачу за назвою мережі і тд.

```
def get_task(self, brand: str) -> Callable:
    """
    Returns the Taskiq task for `brand` by convention:
    | search_{brand}_task must exist in src/tasks.py
    """
    from src import tasks # late import - avoids circular deps
    task_name = f"search_{brand}_task"
    task = getattr(tasks, task_name, None)
    if task is None:
        raise KeyError(
            f"No task '{task_name}' in src.tasks. "
            f"Add it to support brand '{brand}'."
        )
    return task
```

Рисунок 3.3 Функція get_task

Увімкнені мережі зберігаються як посилання на змінну, яка оновлюється через SSE, оскільки *pb_client* запускає роботу з SSE у асинхронній задачі основна робота системи не переривається, що гарантує швидкодію реєстру.

Кожен бренд також має метадані що зберігаються в RocketBase – назва для відображення, колір та фоновий колір для використання у веб-інтерфейсі та email шаблонах. Це дає змогу налаштовувати відображення торгових мереж без змін у коді.

3.1.4 Ініціалізація системи

Кожен мікросервіс системи GroceryScout має чітко визначений порядок ініціалізації що реалізований через механізм lifespan з FastAPI. Lifespan – це контекстний менеджер який виконується при старті та зупинці застосунку, що дозволяє коректно ініціалізувати та звільняти ресурси.

```
@asynccontextmanager
async def lifespan(app: FastAPI) -> Generator[None, Any, None]:
    await start_pb()
    registry.load()
    await registry.sync_all_shops()
    await db.setup_pool()
    await broker.startup()
    yield
    await broker.shutdown()
    await db.teardown_pool()
```

Рисунок 3.4 lifespan функція

При старті API сервісу спочатку викликається start_pb – встановлюється з'єднання з RocketBase, завантажується конфігурація у runtime та запускається фонові задача підписки на зміни. Далі завантажується ShopRegistry – зчитуються файли магазинів для кожного увімкненого бренду, оновлюється список магазинів для кожної робочої торгової мережі. Після цього ініціалізується брокер Taskiq – встановлюється з'єднання з Redis. Лише після успішного завершення всіх кроків сервіс починає приймати запити.

Docker Compose забезпечує правильний порядок запуску контейнерів через depends_on з перевіркою стану здоров'я – API сервіс стартує лише після

					ІАЛЦ.467200.003 ПЗ	Арк.
						48
Зм.	Арк.	№ докум.	Підпис	Дата		

того як PostgreSQL і PocketBase успішно пройшли healthcheck. Це гарантує що при зверненні до бази даних або адмін-панелі вони вже готові до роботи.

При зупинці сервісу lifespan коректно завершує з'єднання з брокером Taskiq, що забезпечує обробку всіх задач що вже знаходяться в черзі перед вимкненням.

3.2 Модуль збору даних

Модуль збору даних відповідає за отримання актуальних цін та інформації про наявність товарів безпосередньо з API торгових мереж у реальному часі. Кожна торгова мережа має власний клієнт що інкапсулює всю логіку взаємодії з її API – отримання списку магазинів, пошук товарів та отримання детальної інформації про них.

3.2.1 Клієнти торгових мереж

На момент розробки система підтримує три торгові мережі – АТВ, Varus та Fora. Незважаючи на те що кожна мережа має власний формат API та внутрішню логіку пошуку всі клієнти реалізують однаковий інтерфейс – метод *search_in_shop* що приймає об'єкт магазину та пошуковий запит і повертає список товарів у єдиному форматі.

Отримання списку магазинів відбувається при першому запуску клієнта – він звертається до відповідного API торгової мережі та передає результат у ShopRegistry через метод *refresh_shops*. Список зберігається у JSON файл та використовується при наступних запусках без зайвих запитів до зовнішніх API.

					ІАЛЦ.467200.003 ПЗ	Арк.
						49
Зм.	Арк.	№ докум.	Підпис	Дата		

```

async def get_stores(self) -> list[ATBShop]:
    if not self.stores:
        self.stores = await self._fetch_stores()
    return self.stores

async def search_in_shop(self, shop: ATBShop, query: str) -> list[ATBProduct]:
    params = {
        "id": self.ATB_MULTISEARCH_ID,
        "key": self.ATB_API_KEY,
        "lang": "uk",
        "location": str(shop.id),
        "m": str(int(time.time() * 1000)),
        "q": _gen_q(),
        "query": query,
        "s": "small",
        "uid": self.uid,
    }
    response: Response = await self.search_session.get(
        "https://api.multisearch.io/",
        params=params,
        headers=MULTISEARCH_HEADERS,
    )
    data = response.json()
    if not data.get("results", {}).get("item_groups"):
        return []
    return _parse_products(data)

```

Рисунок 3.5 методи `get_stores` та `search_in_shop` класу `AtbClient`

Для більшості мереж пошук відбувається у два етапи – спочатку отримуються ідентифікатори товарів через пошуковий API, далі за цими ідентифікаторами запитується детальна інформація з цінами та наявністю. Такий підхід дозволяє отримати актуальні дані саме для обраного магазину, а не загальні дані по мережі. Деякі торгові мережі одразу повертають достатньо інформації при першому запиті, що зменшує загальну кількість дій для отримання важливої інформації.

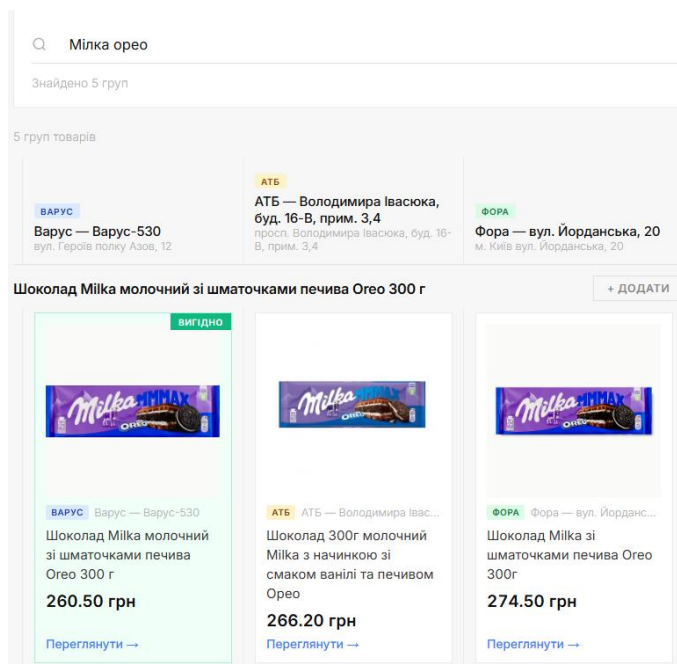


Рисунок 3.6 Пошукова видача GroceryScout для трьох фізичних точок

Як видно з Рисунку 3.6 система успішно агрегує результати пошуку з декількох торгових мереж одночасно, відображаючи актуальні ціни та інформацію про наявність товарів у кожній з обраних фізичних торгових точок. Такий підхід дозволяє користувачу миттєво порівняти пропозиції різних магазинів в єдиному інтерфейсі без необхідності перевіряти кожен сайт окремо.

3.2.2 Обхід антибот-захисту з curl_cffi

Більшість сайтів торгових мереж використовують системи захисту від автоматизованого збору даних. Як було описано у Розділі 2 – стандартні Python бібліотеки легко виявляються такими системами і не підходять для нашої задачі, як приклад можна подивитись на один з запитів до сайту АТБ

```

1 import requests
2 from src.settings import BROWSER_TYPES_CYCLE
3
4 cookies = {
5     'lang': 'uk',
6 }
7
8 data = {
9     'city': '395',
10 }
11
12 response = requests.post(
13     'https://www.atbmarket.com/site/getstore',
14     headers=headers,
15     data=data,
16 )
17 response.status_code, response.text
✓ 2.0s
(403,
'<!DOCTYPE html><html lang="en-US"><head><title>Just a moment...</title>

```

```

1 from curl_cffi import requests
2 from src.settings import BROWSER_TYPES_CYCLE
3
4 cookies = {
5     'lang': 'uk',
6 }
7
8 data = {
9     'city': '395',
10 }
11
12 response = requests.post(
13     'https://www.atbmarket.com/site/getstore',
14     headers=headers,
15     data=data,
16     impersonate=next(BROWSER_TYPES_CYCLE),
17 )
18 response.status_code, response.text
✓ 0.6s
(200,
'{"res":1,"coordinates":[{"lat":"50.4459719382","lng":"30.4167577657","id":1262},

```

Рисунок 3.7 Порівняння запитів requests та curl_cffi

Як видно з Рисунок 3.7 – один й той самий запит з однаковими заголовками для requests повернув статус 403 з типовою відповіддю “Just a moment...” – що означає що відповідь була захищена від скрапінгу і виникла капча. На той самий запит curl_cffi повернув відповідь з 200 статус кодом та корисними даними, що свідчить про те що curl_cffi у контексті скрапінгу працює набагато краще.

Щоб зрозуміти чому в таких кейсах curl_cffi може обходити певні захисти потрібно звернутись на рівень нижче. TLS (Transport Layer Security) – це протокол шифрування що використовується при встановленні HTTPS з'єднання. При підключенні клієнт надсилає серверу повідомлення ClientHello що містить список підтримуваних шифрів, розширень та їх порядок. Сукупність цих параметрів формує унікальний відбиток – JA3 хеш [17], який є характерним для кожного клієнта. Стандартні Python бібліотеки requests та httpx використовують OpenSSL з типовим набором параметрів що легко ідентифікується як не-браузерний клієнт. curl_cffi натомість використовує BoringSSL [3] – той самий TLS стек що використовує Chrome браузер, що робить відбиток неможливим для відрізнєння від реального браузера на рівні мережевого з'єднання. Проте імітація TLS фінгерпринтів не є універсальним

рішенням – існують захисти які мають інші критичні точки, по яким можна зрозуміти що запити “не справжні” – тобто автоматизовані.

Для забезпечення різноманітності відбитків між запитами реалізовано ротацію профілів браузерів через *BROWSER_TYPES_CYCLE* – нескінченний ітератор що по черзі повертає профілі різних версій Chrome та Android браузерів. Кожен новий клієнт отримує наступний профіль з циклу, що мінімізує ймовірність виявлення повторюваного патерну запитів антибот системою.

```
BROWSER_TYPES = [
    "chrome99", "chrome100", "chrome101", "chrome104",
    "chrome107", "chrome110", "chrome116", "chrome119",
    "chrome120", "chrome123", "chrome124", "chrome131",
    "chrome133a", "chrome136", "chrome142", "chrome145",
    "chrome146", "chrome99_android", "chrome131_android"
]
BROWSER_TYPES_CYCLE = cycle(BROWSER_TYPES)
```

Рисунок 3.8 Список підтримуваних TLS профілей та ітератор

Кожен профіль містить не лише TLS налаштування але й відповідний набір HTTP заголовків характерних для конкретної версії браузера – User-Agent, Accept-Language, Accept-Encoding та інші. Це дозволяє імітувати реальний браузер одночасно на двох рівнях – на рівні TLS з'єднання та на рівні HTTP заголовків, що є достатнім для обходу більшості сучасних антибот систем без використання повноцінної браузерної автоматизації.

3.2.3 Розподілена обробка через Taskiq

Паралельний пошук у декількох магазинах одночасно є ключовою вимогою системи – послідовне звернення до кожного магазину було б

неприйнятно повільним. Для реалізації розподіленої обробки використовується Taskiq [2] з Redis [6] як брокером черги задач.

Для кожної торгової мережі визначена власна задача – *search_atb_task*, *search_varus_task* тощо. Задача отримує словник магазину та пошуковий запит, створює відповідний клієнт, виконує пошук та повертає результат у стандартизованому форматі що включає назву магазину, бренд та список знайдених товарів з цінами.

```
@broker.task
async def search_varus_task(shop: dict, query: str) -> dict:
    client = VarusClient()
    products = await client.search_in_shop(VarusShop(**shop), query)
    return {
        "shop_id": shop["id"],
        "shop_name": shop["short_name"],
        "brand": "varus",
        "products": [_product_dict(p) for p in products],
    }
```

Рисунок 3.9 Приклад пошукової задачі у Варус

При отриманні пошукового запиту API сервіс через *ShopRegistry.resolve_shops* отримує список активних магазинів з обраних користувачем, для кожного магазину визначає відповідну задачу через *get_task* та додає її до черги Redis через метод *kiq*. Після додавання всіх задач API сервіс паралельно очікує результати через *wait_result* з таймаутом 30 секунд.

```
@router.post("/search")
async def search(req: SearchRequest, _: asyncpg.Record = Depends(get_current_user)) -> list[Any]:
    grouped = registry.resolve_shops(req.shops)

    if not grouped:
        raise HTTPException(status_code=422, detail="No enabled shops matched the request")

    jobs = await asyncio.gather(*[
        registry.get_task(brand).kiq(shop, req.query)
        for brand, shops in grouped.items()
        for shop in shops
    ])
    results = await asyncio.gather(*[job.wait_result(timeout=30) for job in jobs])
    return [r.return_value for r in results]
```

Рисунок 3.10 Створення пошукових задач та очікування результату

Воркери є окремими процесами що постійно слухають чергу Redis та виконують задачі по мірі їх надходження.

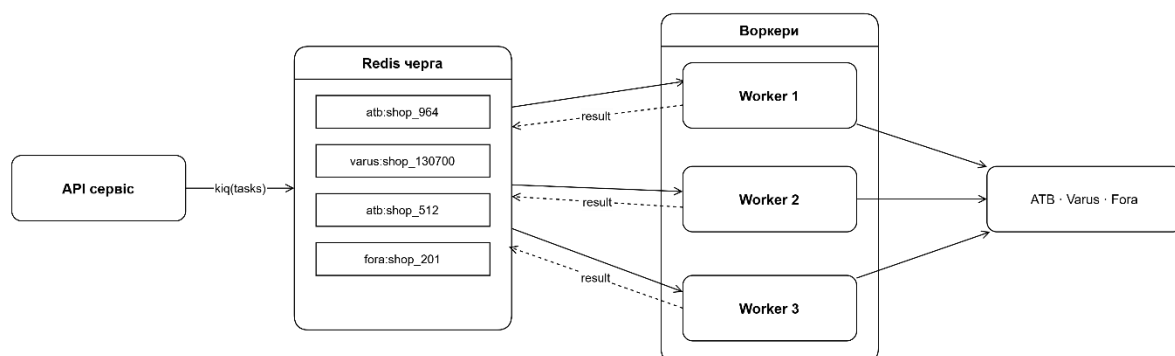


Рисунок 3.11 Схема розподілу задач між воркерами

Як видно з Рисунок 3.11 кожна задача є незалежною та не має стану –любий воркер може взяти любую задачу. У такому випадку кількість воркерів можна нарощувати горизонтально без будь-яких змін у коді. При збільшенні навантаження достатньо запустити додаткові екземпляри воркера.

3.3 LLM сервіс нормалізації

Після агрегації результатів пошуку з усіх торгових мереж виникає задача групування однакових товарів – один і той самий продукт може мати різну назву в різних мережах. Для вирішення цієї задачі виділено окремий мікросервіс що використовує велику мовну модель.

LLM сервіс реалізований як окремий FastAPI застосунок, який приймає список товарів з усіх магазинів, групує їх за схожістю та повертає структурований результат з канонічною назвою та пропозиціями від кожного магазину з актуальними цінами.

Якість групування напряму залежить від системного пром프트 що передається моделі. Промпт описує правила за якими товари вважаються

однаковими – однаковий бренд, тип, жирність, вага та інші характеристики. Промпт зберігається в RocketBase як налаштування PROMPT_STRICT що дозволяє редагувати його через адмін-панель без перезапуску сервісу. Крім промпту у адмін-панелі ще зберігається API-ключ, назва моделі та максимальна кількість токенів. Це дає змогу міняти моделі на ходу без перезапуску системи, що зручно для тестування та на випадок проблем з доступом до моделі.

Модель отримує список товарів у текстовому форматі де кожен товар має унікальний ключ у форматі *brand:shop_id:idx*. У відповіді модель повертає JSON з групами де кожна група містить канонічну назву та список ключів товарів що до неї належать. Для забезпечення стабільного формату відповіді використовується параметр *response_format* з типом *json_object*.

Для контролю якості групування розроблено систему регресійного тестування – тестові набори містять реальні приклади товарів з очікуваними результатами групування. Детальні результати тестування наведено у Розділі 4.

3.4 Система аутентифікації

Система аутентифікації забезпечує реєстрацію та авторизацію користувачів. Дані користувачів зберігаються у PostgreSQL, окремо від RocketBase для прямого контролю над запитами та зменшення навантаження на адмін-панель. Взаємодія з базою даних реалізована через *asyncpg* – асинхронний драйвер PostgreSQL що нативно підтримує *asyncio*.

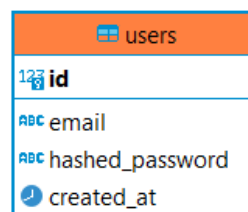


Рисунок 3.11 Схема таблиці *users*

Схема бази даних містить таблицю *users* з полями ідентифікатора, електронної пошти та хешованого пароля. Таблиця створюється автоматично при першому запуску сервісу через виклик у *lifespan*.

```
_CREATE_TABLES = """
CREATE TABLE IF NOT EXISTS users (
    id            SERIAL PRIMARY KEY,
    email         TEXT UNIQUE NOT NULL,
    hashed_password TEXT NOT NULL,
    created_at    TIMESTAMPTZ DEFAULT NOW()
);
"""

async def setup_pool() -> None:
    global _pool
    _pool = await asyncpg.create_pool(runtime.get("DATABASE_URL"))
    print(f"DATABASE_URL: {runtime.get("DATABASE_URL")}", flush=True)
    async with _pool.acquire() as conn:
        await conn.execute(_CREATE_TABLES)
```

Рисунок 3.12 Функція для створення таблички *users* у *lifespan*

Паролі зберігаються у хешованому вигляді через бібліотеку *bcrypt* [13] – при реєстрації пароль хешується та зберігається, при логіні введений пароль порівнюється з хешем. Це гарантує що навіть при компрометації бази даних реальні паролі залишаються захищеними.

```
async def get_current_user(
    access_token: str | None = Cookie(default=None),
    conn: asyncpg.Connection = Depends(get_conn),
) -> asyncpg.Record:
    if not access_token:
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Not authenticated")
    try:
        payload = decode_token(access_token)
        email: str = payload["email"]
    except (jwt.PyJWTError, KeyError):
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Invalid token")
    user = await get_by_email(conn, email)
    if not user:
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="User not found")
    return user

async def require_page_auth(access_token: str | None = Cookie(default=None)) -> None:
    if not access_token:
        raise HTTPException(status_code=307, headers={"Location": "/login"})
    try:
        decode_token(access_token)
    except jwt.PyJWTError:
        raise HTTPException(status_code=307, headers={"Location": "/login"})
```

Рисунок 3.13 Dependency функції для редіректу та отримання користувача

Після успішного логіну або реєстрації користувач отримує JWT токен що зберігається у `httpOnly cookie` [12] – такий підхід унеможлиблює читання токена через JavaScript та захищає від XSS атак. Варто відзначити що JWT секрет зберігається не у `.env`, а в `PocketBase` як і решта конфігурації системи – це дозволяє змінити секрет без перезапуску сервісу у разі потреби.

Захист ендпоінтів реалізований через `FastAPI Dependency Injection` – залежність `get_current_user` читає токен з `cookie`, декодує його та повертає поточного користувача, або повертає 401 якщо токен відсутній чи недійсний. Для сторінок веб-інтерфейсу реалізована окрема залежність `require_page_auth` що перенаправляє неавторизованого користувача на сторінку логіну через 307 редирект.

3.5 Email сервіс

Після порівняння цін користувач може отримати чек на електронну пошту – список товарів з цінами по кожному магазину з виділенням найвигіднішого варіанту. Відправка реалізована асинхронно через `Taskiq` – ендпоінт `/api/send-receipt` одразу повертає відповідь, а задача `send_receipt_task` виконується у фоні воркером.

Перед відправкою дані кошика збагачуються через функцію `_enrich` – для кожного магазину з `ShopRegistry` підтягуються координати для посилання на `Google Maps`, магазини сортуються за загальною вартістю та доступністю товарів, а найвигідніший варіант де всі товари є в наявності позначається як `is_best`.

```
@broker.task
async def send_receipt_task(to: str, receipts: list[dict], shop_ids: list[dict]) -> None:
    parsed = [ReceiptData(**r) for r in receipts]
    enriched = _enrich(parsed, shop_ids)
    await send_receipt_email(to, enriched)
```

Рисунок 3.14 задача відправки email

					ІАЛЦ.467200.003 ПЗ	Арк.
						58
Зм.	Арк.	№ докум.	Підпис	Дата		

HTML шаблон листа рендериться через Jinja2 з передачею збагачених даних, метаданих брендів з ShopRegistry та email користувача. Шаблон містить кастомний фільтр plural_uk для правильного відмінювання українських слів та фільтр maps_url що генерує посилання на Google Maps за координатами магазину.

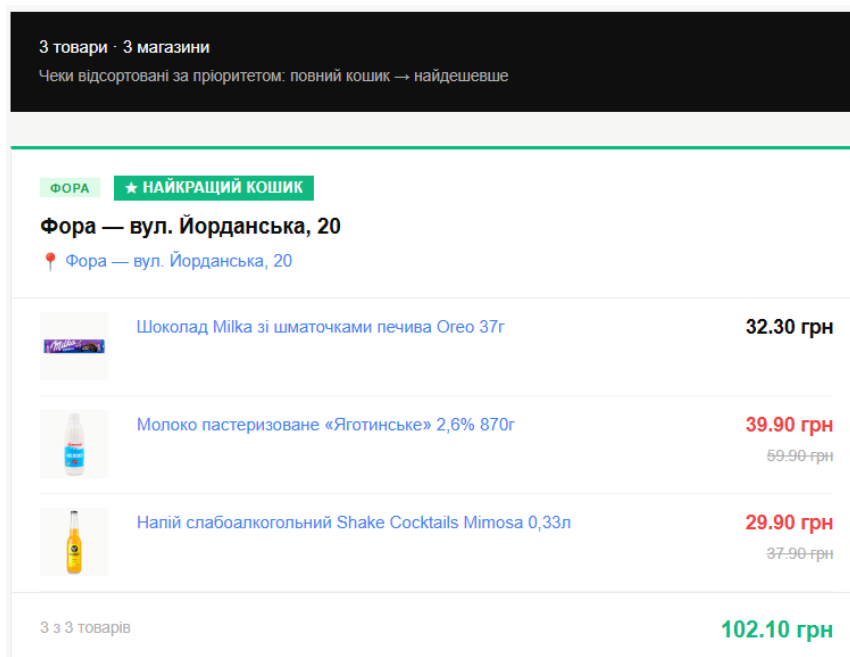


Рисунок 3.15 Приклад email чеку користувача

Відправка виконується через бібліотеку aiosmtplib [14] – асинхронний SMTP клієнт що нативно підтримує asyncio. SMTP параметри зберігаються в .env оскільки вони не потребують динамічної зміни в рантаймі. Приклад email який отримує користувач наведено у Рисунку 3.15.

3.6 Веб-інтерфейс

Веб-інтерфейс системи GroceryScout реалізований без використання фронтенд фреймворків – чистий HTML, CSS та JavaScript. Статичні файли

монтуються як Docker volume та роздаються FastAPI через StaticFiles [1], що дозволяє оновлювати інтерфейс без перезбірки контейнера. Спільні стилі винесені в окремий файл styles.css – змінні CSS з кольорами брендів, типографіка та компоненти є спільними для всіх сторінок.

Інтерфейс складається з трьох сторінок. Сторінка авторизації містить єдину форму з перемиканням між режимами входу та реєстрації через вкладки. При відкритті перевіряється наявність активної сесії – якщо користувач вже авторизований він одразу перенаправляється на головну сторінку.

Сторінка вибору магазинів є точкою входу після авторизації. Ліва панель містить список магазинів з фільтрацією за брендом та назвою, вибором через геолокацію з налаштуванням радіусу. Права частина – інтерактивна карта Leaflet.js [8Open] з маркерами магазинів. Вибрані магазини зберігаються в localStorage та передаються на наступну сторінку.

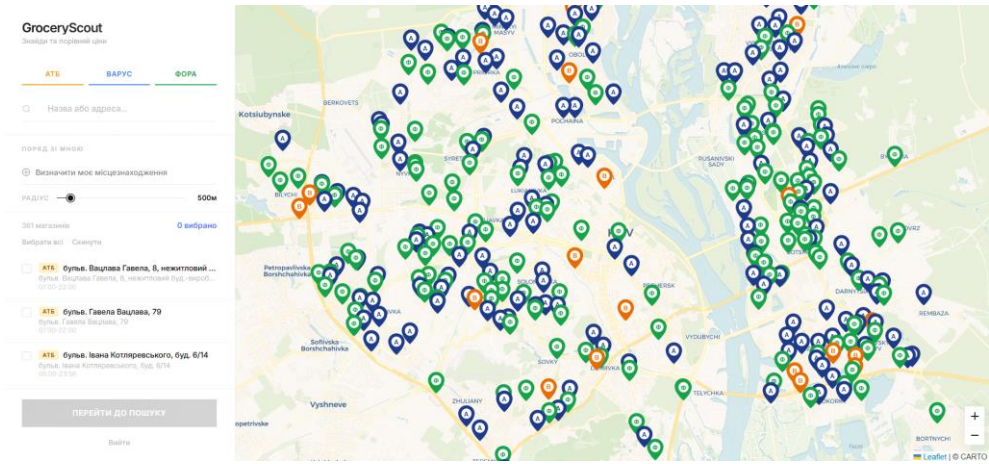


Рисунок 3.16 Головна сторінка та мапа

Сторінка пошуку є основною робочою сторінкою – поле пошуку, таблиця результатів де рядки це групи нормалізованих товарів а колонки це магазини, та панель кошика з порівнянням загальної вартості по кожному магазину. Детальна демонстрація роботи інтерфейсу наведена у Розділі 4.

ВИСНОВОК ДО РОЗДІЛУ

У третьому розділі описано проектування та реалізацію системи GroceryScout. Розроблена система складається з трьох власних мікросервісів та трьох інфраструктурних компонентів що розгортаються за допомогою Docker Compose.

Описано спільну бібліотеку *libs* що монтується як Docker volume у кожен контейнер та забезпечує єдину точку конфігурації через PocketBase з підтримкою оновлення налаштувань у реальному часі без перезапуску сервісів. Реалізовано ShopRegistry – реєстр торгових мереж що автоматично виявляє підтримувані мережі та надає уніфікований інтерфейс для роботи з ними.

Модуль збору даних реалізує клієнти для трьох торгових мереж – АТВ, Varus та Fora. Для обходу антибот-захисту використовується бібліотека curl_cffi з ротацією профілів браузерів що імітує TLS відбиток реального Chrome браузера. Розподілена обробка запитів реалізована через Taskiq та Redis – кожне звернення до магазину є окремою задачею що виконується паралельно на доступних воркерах.

Для нормалізації та групування товарів реалізовано окремий LLM мікросервіс що використовує велику мовну модель через LiteLLM. Якість групування контролюється через систему регресійного тестування промптів.

Система аутентифікації реалізована через JWT токени у httpOnly cookies з зберіганням даних користувачів у PostgreSQL. Email сервіс забезпечує відправку чеків з порівнянням цін через асинхронний SMTP клієнт з Jinja2 [15] шаблонами.

Веб-інтерфейс реалізований як односторінковий застосунок на чистому HTML та JavaScript з інтерактивною картою Leaflet.js для вибору торгових точок.

					ІАЛЦ.467200.003 ПЗ	Арк.
						61
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 4. ТЕСТУВАННЯ ТА ДЕМОНСТРАЦІЯ РОБОТИ СИСТЕМИ

4.1 Демонстрація роботи системи

У даному підрозділі наведено повний сценарій роботи системи GroceryScout від реєстрації користувача до отримання чеку на електронну пошту. Всі операції виконуються в реальному часі з актуальними даними торгових мереж.

4.1.1 Реєстрація та авторизація

При першому відкритті застосунку користувач потрапляє на сторінку авторизації. Для початку роботи необхідно створити обліковий запис – ввести електронну пошту та пароль у формі реєстрації. Після успішної реєстрації система автоматично авторизує користувача та перенаправляє на головну сторінку.

Рисунок 4.1 Форма реєстрації GroceryScout

Після натиснення кнопки “ЗАРЕЄСТРУВАТИСЯ” можемо перевірити нашу таблицю *users* – з’явився новий запис з нашим email

id	email	hashed_password	created_at
2	berezanskiydaniil2@gmail.com	\$2b\$12\$g0LnJBZX.M8hSmDzcMiZLZ969bytk7icm0IstF0PE7qmvzvPgcO	2026-05-28 00:20:30.509 +0400
3	berezanskiydaniil@gmail.com	\$2b\$12\$xoNYteGLyIPm4AGHuszR/O9LAs1VIPPS7fjIMWRvQNHVXAaH4CVHp6	2026-05-28 00:52:59.765 +0400
4	test_email@gmail.com	\$2b\$12\$N/3nYhVND4nj9U6.04pZiOWG8LUIFKLHrGizkN7zAHnAshF95800i	2026-06-02 23:34:02.858 +0400

Рисунок 4.2 Стан таблиці після реєстрації нового користувача

Як було описано раніше – ми зберігаємо користувачів не у PocketBase колекції, а в окремій базі даних, це рішення прийнято для того, щоб зчитування конфігурації відбувалось максимально швидко, тобто PocketBase зберігає лише конфіги, без інших таблиць.

Отже, реєстрація працює правильно, можемо рухатись до інших сторінок веб-застосунку.

4.1.2 Вибір торгових точок на мапі

Після авторизації користувач потрапляє на сторінку вибору магазинів. На карті відображаються маркери всіх увімкнених торгових точок. Користувач може фільтрувати магазини за брендом, шукати за назвою або адресою, а також скористатись геолокацією для вибору найближчих магазинів у заданому радіусі.

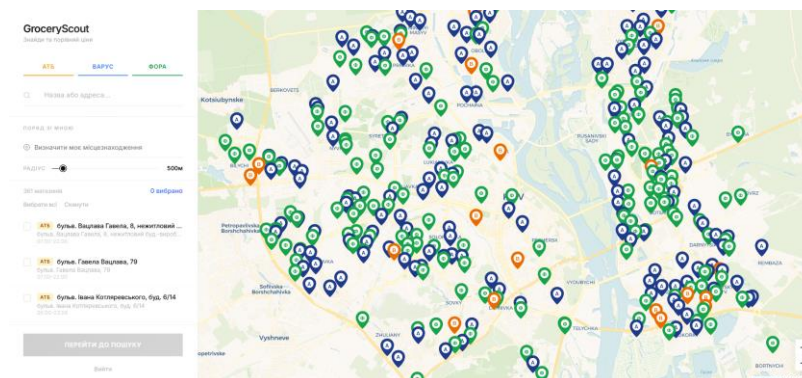


Рисунок 4.3 Головна сторінка та мапа

Далі виберемо по кожній фізичній точці з підтримуваних торгових мереж – Varus, АТВ та Fora.

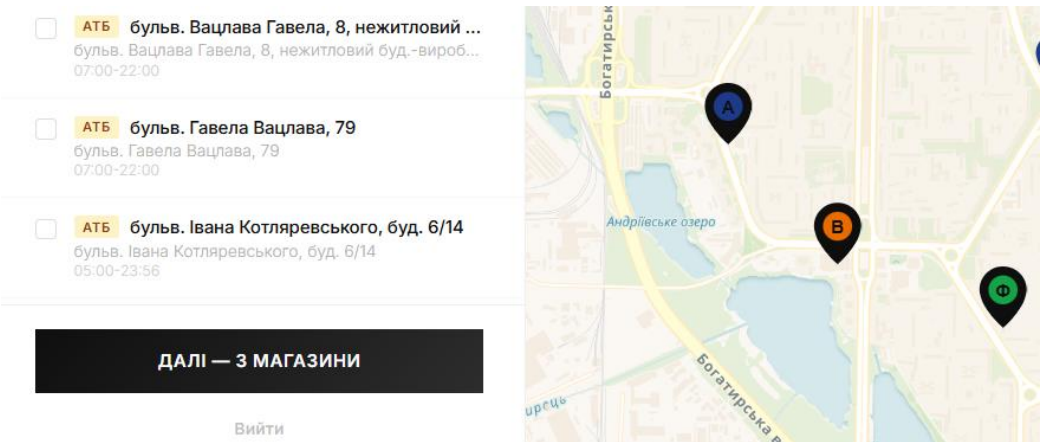


Рисунок 4.4 Вибір декількох маркерів

Після натиснення кнопки “ДАЛІ – 3 МАГАЗИНИ” користувача перенесе на сторінку пошуку по цих магазинах. У ході перевірки головної сторінки було показано, що дійсно усі магазини Києву відображаються на мапі, та користувач має змогу вибирати магазини на свій розсуд.

4.1.3 Пошук та нормалізація товарів

На сторінці пошуку користувач вводить назву товару. Система паралельно звертається до API кожного обраного магазину, агрегує результати та передає їх до LLM сервісу для нормалізації та групування. Однакові товари з різних мереж об'єднуються в одну групу з канонічною назвою.

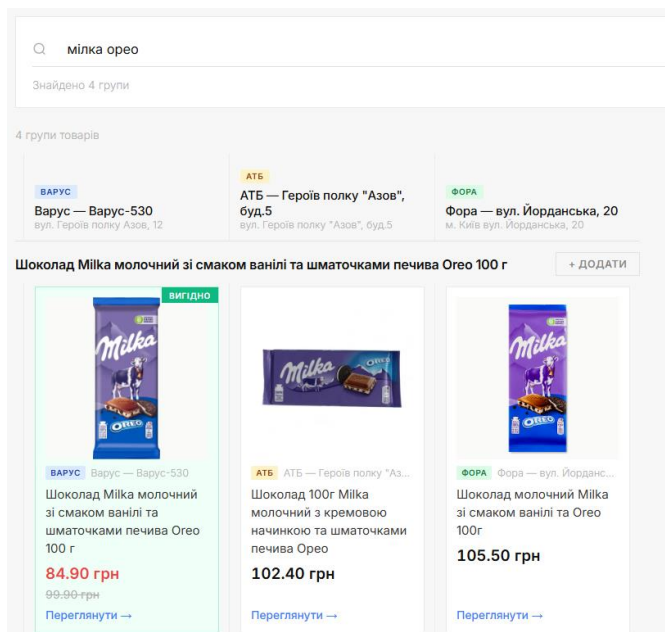


Рисунок 4.5 Пошукова видача GroceryScout за запитом “мілка орео”

Як бачимо з Рисунку 4.5 – перший товар “Шоколад молочний Milka зі смаком ванілі та Oreo 100г” був коректно визначений – уся група відображає правильний товар, при чому назви відрізняються – отже LLM мікросервіс справляється добре.

Також важливо перевірити чи ціни дійсно є правильними, для цього перейдемо на кожне з посилань та виберемо правильну фізичну точку

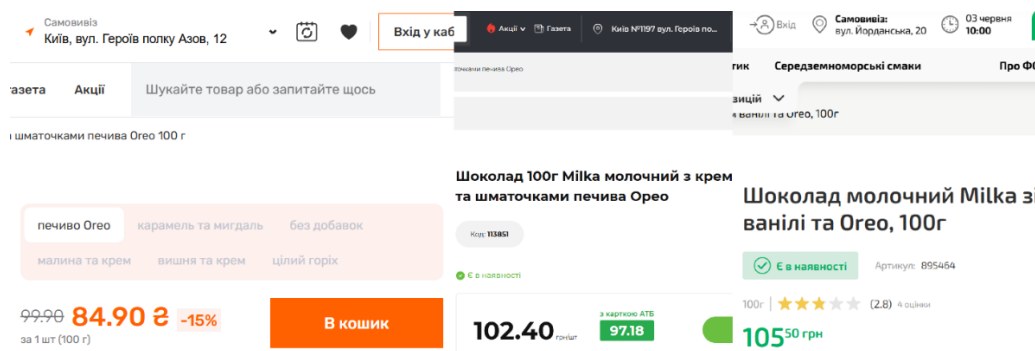


Рисунок 4.6 Порівняння цін з офіційних сайтів для “Шоколад молочний Milka зі смаком ванілі та Oreo 100г”

Як видно з Рисунку 4.6 – ціни та наявність на товар “Шоколад молочний Milka зі смаком ванілі та Oreo 100г” коректні, це гарантується в першу чергу тим що дані отримуються напряму з API торгових мереж, а не через власну базу даних, яка може бути некоретною.

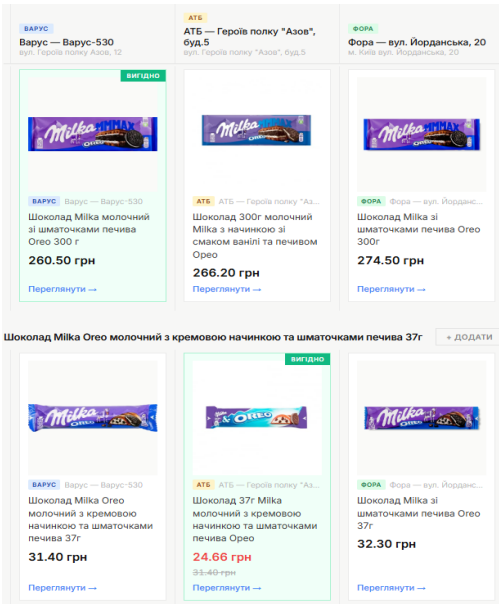


Рисунок 4.7 Решта пошукової видачі за запитом “мілка орео”

Найдешевша пропозиція в кожній групі підсвічується зеленим кольором що дозволяє користувачу миттєво визначити найвигідніший варіант. Спробуємо додати першу групу з Рисунку 4.7 до кошика

Кошик порівняння (1)	
3 магазини	
★ Варус — Варус-530	84.90 грн 1/1
✓ АТБ — Героїв полку "Азов", ...	102.40 грн 1/1
✓ Фора — вул. Йорданська, 20	105.50 грн 1/1

Рисунок 4.8 Кошик “Шоколад молочний Milka зі смаком ванілі та Oreo 100г”

З Рисунок 4.8 видно, що після того як користувач додав першу групу до кошика – у правій частині сторінки з’явилося порівняння по магазинах. У нашому випадку Varus – є найвигіднішим кошиком, відповідно він відображається першим та підсвічений зеленим кольором як вигідний варіант.

4.1.4 Вибір аналогів

Користувач також має можливість вибирати не одну лінію товарів, а по одному товару з кожного магазину, таким чином можна вибрати найвигідніші пропозиції серед кожного магазину.

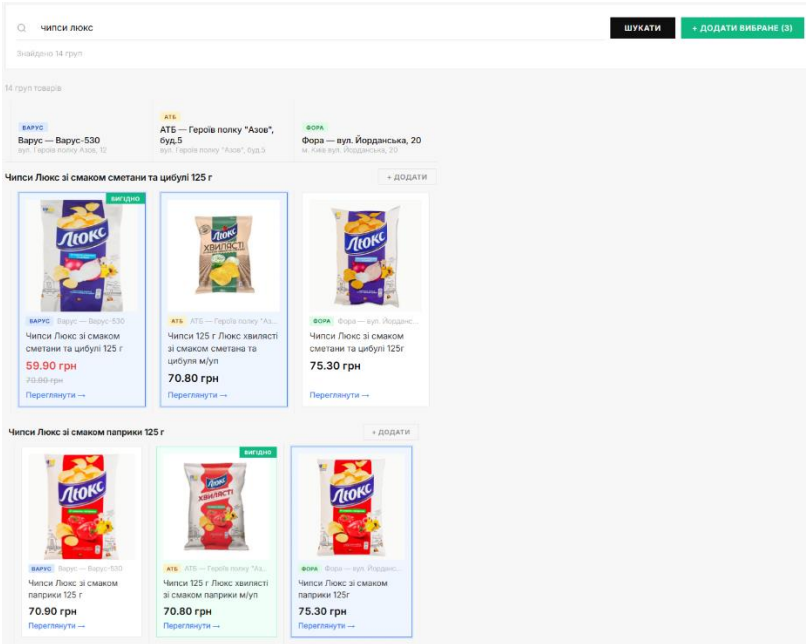


Рисунок 4.9 Вибір аналогів за запитом “чипси люкс”

Як видно з Рисунок 4.9 – користувач може обрати по варіанту з кожного магазину, такі пропозиції виділяються синім кольором. Таким чином користувач може вибирати аналоги, різноманітні смаки та найвигідніші акції. Щоб додати такий вибір потрібно натиснути зелену кнопку “+ ДОДАТИ ВИБРАНЕ” – після чого товари також потрапляють до кошика.

Кошик порівняння (2)	
3 магазини	
★ Варус — Варус-530	144.80 грн 2/2
Шоколад Milka молочний зі смако...	84.90 грн
Чипси Люкс зі смаком сметани та ...	59.90 грн
★ Найкращий кошик	144.80 грн
✓ АТБ — Героїв полку "Азов", ...	173.20 грн 2/2
✓ Фора — вул. Йорданська, 20	180.80 грн 2/2

Рисунок 4.10 Розширений кошик з декількох пропозицій

Отже, на етапі пошуку користувач має повну змогу вибирати будь-які товари – як цілу групу ідентичних товарів так і окремі пропозиції з різних магазинів. Такий підхід до формування кошика є унікальним серед розглянутих аналогів та дає користувачу максимальну гнучкість при плануванні покупок.

4.1.5 Відправка чеку на email

Після формування кошика користувач може надіслати чек на електронну пошту одним кліком на кнопку "НАДІСЛАТИ ЧЕК". Запит обробляється асинхронно — сторінка не блокується під час відправки.

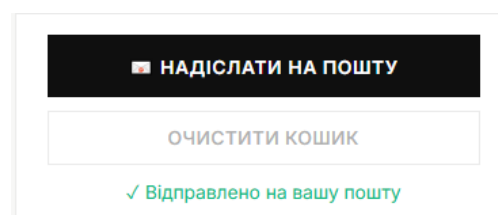


Рисунок 4.11 Кнопка відправки та повідомлення про успішну відправку

Отриманий лист містить список товарів з цінами по кожному магазину, загальну вартість кошика та посилання на Google Maps для кожної торгової

					ІАЛЦ.467200.003 ПЗ	Арк.
						68
Зм.	Арк.	№ докум.	Підпис	Дата		

точки. Магазины відсортовані від найдешевшого до найдорожчого, найвигідніший варіант виділено окремо.

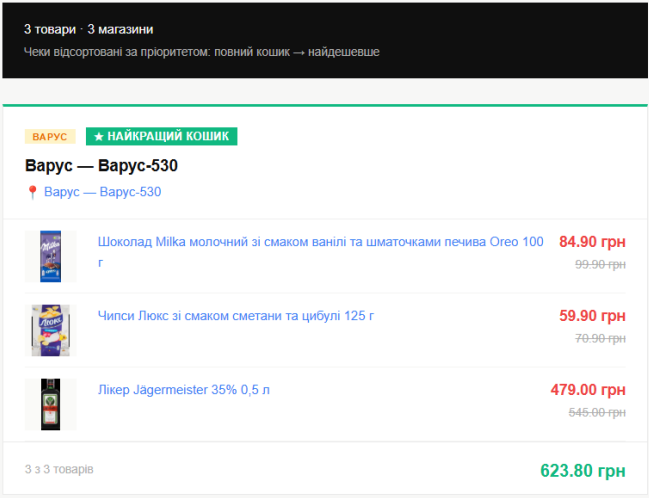


Рисунок 4.12 Отриманий email чек, найкращий кошик

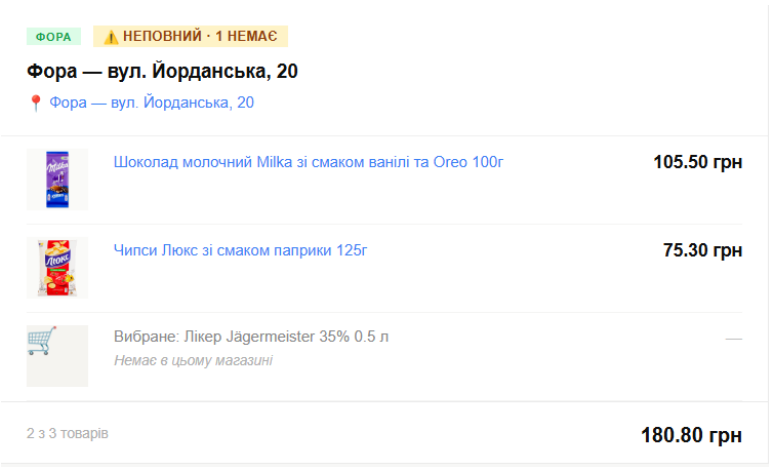


Рисунок 4.13 Отриманий email чек, неповний кошик

Таким чином продемонстровано повний сценарій роботи системи – від реєстрації до отримання готового чеку з порівнянням цін на пошту. Як видно з Рисуноків 4.12 та 4.13 – користувач отримує чек по кожному з обраних магазинів, навіть якщо певний товар відсутній у конкретній торговій точці – такий кошик позначається як неповний та відображається окремо. Це дає користувачу повну картину – він може самостійно вирішити чи варто обирати

магазин де є всі товари але дорожче, чи магазин де частина товарів відсутня але загальна вартість нижча.

4.2 Тестування адміністративних можливостей

Окрім основного функціоналу система надає адміністративні можливості керування через PocketBase – вмикання та вимикання торгових мереж, зміна налаштувань LLM та редагування промптів без перезапуску сервісів.

4.2.1 Вимкнення торгової мережі

Для перевірки динамічного вимкнення торгової мережі вимкнемо АТВ через адмін-панель PocketBase – змінимо значення ключа `enabled_brands` виключивши `atb` зі списку. Варто зазначити що зміна набуває чинності миттєво – після збереження всі мікросервіси отримують оновлення через SSE підписку без будь-якого перезапуску, оскільки значення `enabled_brands` зберігається у змінній `runtime` що оновлюється у реальному часі.

T key	T key
enabled_brands	enabled_brands
T value	T value
atb,varus,fora	varus,fora

Рисунок 4.14 Виключення atb з увімкнених торгових мереж

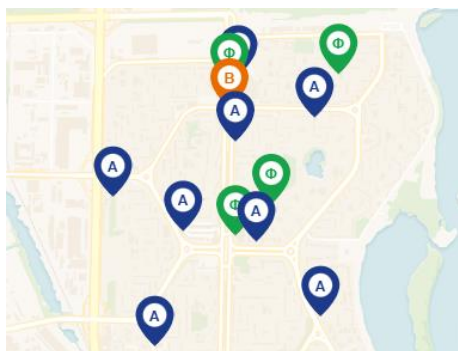


Рисунок 4.15 Оболонь – Varus, АТВ, Fora

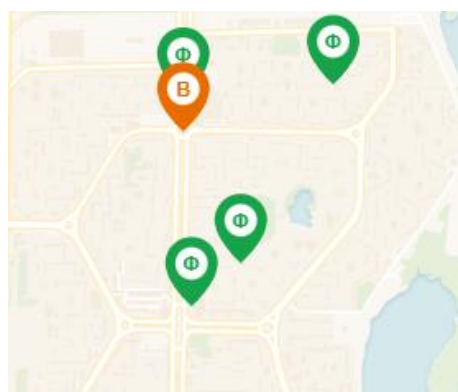


Рисунок 4.16 Оболонь – Varus, Fora

Після збереження зміни всі мікросервіси отримують оновлення через SSE підписку без перезапуску. Повернемось на головну сторінку –маркери АТВ зникли з карти, в списку магазинів відображаються лише Varus та Fora.

Якщо спробувати виконати пошук з магазином АТВ – API сервіс поверне помилку оскільки ShopRegistry відхиляє запити до вимкнених брендів. Таким чином вимкнення торгової мережі повністю виключає її з усіх операцій системи без будь-яких змін у коді та без перезапуску.

4.2.2 Зміна налаштувань LLM

Система підтримує динамічну зміну всіх параметрів LLM мікросервісу через RocketBase – назву моделі, API ключ, максимальну кількість токенів та системний промпт. При зміні будь-якого з цих параметрів LLM мікросервіс

отримує оновлення через SSE та використовує нові налаштування при наступному запиті без перезапуску. Це дозволяє швидко переключатись між провайдерами та моделями – наприклад з Groq на Google Gemini – або коригувати промпт для покращення якості групування без зупинки системи.

4.3 Регресійне тестування промтів

Для контролю якості групування товарів розроблено систему регресійного тестування що дозволяє оцінити точність LLM на наборі реальних тестових випадків. Тестування запускається через CLI скрипт *regression.py* та не потребує запусченої системи – достатньо доступу до LLM API.

Тестовий набір складається з реальних пошукових запитів з очікуваними результатами групування. Кожен тестовий випадок містить список товарів з кількох магазинів, перелік товарів що мають потрапити в одну групу (*expected*) та перелік товарів що не повинні бути згруповані разом (*must_not_group*).

```
"passed": false,
"checks_ok": 9,
"checks_total": 10,
"errors": [
  " X Milka Oreo 300г - однаковий товар, різне написання\n Expected together: ['131776_1', '2801']\n"
],
"groups": [
  [
    {
      "id": "113851_1",
      "name": "Шоколад 100г Milka молочний з кремовою начинкою та шматочками печива Oreo"
    },
    {
      "id": "43613",
      "name": "Шоколад Milka молочний зі смаком ванілі та шматочками печива Oreo 100 г"
    }
  ],
  [
    {
      "id": "132166_1",
      "name": "Шоколад 37г Milka молочний з кремовою начинкою та шматочками печива Oreo"
    }
  ]
]
```

Рисунок 4.17 Приклад одного з регресійних тестів

Результат показує кількість пройдених та провалених тестів з детальним описом кожної помилки – які товари були згруповані неправильно та де саме модель помилилась.

Система регресійного тестування також дозволяє порівнювати різні промпти та моделі – через параметр *--prompt* можна передати новий промпт та одразу побачити як він впливає на результат без необхідності змінювати налаштування в RocketBase. Це суттєво прискорює процес розробки та вдосконалення промптів.

					ІАЛЦ.467200.003 ПЗ	Арк.
						73
Зм.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВОК ДО РОЗДІЛУ

У четвертому розділі проведено тестування та демонстрацію роботи системи GroceryScout. Повний сценарій роботи від реєстрації до отримання чеку на електронну пошту був успішно продемонстрований – система коректно виконала всі ключові операції в реальному часі з актуальними даними трьох торгових мереж.

У ході демонстрації підтверджено що інтерактивна карта відображає всі торгові точки Києву, пошук паралельно звертається до API кожного обраного магазину, LLM мікросервіс коректно групує товари з різними найменуваннями в одну групу, кошик правильно сортує магазини за вартістю та виділяє найвигідніший варіант. Email чек містить повну інформацію по кожному магазину включно з посиланням на Google Maps.

Тестування адміністративних можливостей підтвердило що вимкнення торгової мережі через RocketBase миттєво відображається у всіх компонентах системи – маркери зникають з карти, бренд виключається з пошуку – без перезапуску жодного сервісу. Динамічна зміна налаштувань LLM дозволяє переключатись між моделями та редагувати промпти в реальному часі.

Регресійне тестування промптів підтвердило ефективність розробленої системи контролю якості – тестові набори з реальними прикладами товарів дозволяють оцінити вплив змін промпту на точність групування перед застосуванням у продакшні.

					ІАЛЦ.467200.003 ПЗ	Арк.
						74
Зм.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У ході виконання дипломної роботи було розроблено систему GroceryScout – вебсервіс для пошуку та порівняння цін на продовольчі товари у фізичних торгових точках Києву з використанням розподілених обчислень та технологій штучного інтелекту.

У першому розділі проведено комплексний аналіз предметної області – досліджено проблему порівняння цін у роздрібній торгівлі, розглянуто методи автоматизованого збору даних та механізми антибот-захисту. Проведено порівняльний аналіз існуючих рішень – Supermarkets.com.ua, Goods та Де Дешевше – та встановлено що жодне з них не поєднує одночасно пошук у реальному часі, порівняння повного кошика та прив'язку до конкретних фізичних торгових точок.

У другому розділі сформульовано вимоги до системи та обґрунтовано вибір технологічного стеку. Проведено порівняльний аналіз HTTP клієнтів з власним бенчмарком – обрано curl_cffi як єдине рішення що поєднує асинхронне API та імітацію браузерного TLS для обходу антибот-систем. Обґрунтовано вибір мікросервісної архітектури, Taskiq з Redis для розподіленої обробки, LiteLLM для роботи з LLM та PocketBase для динамічного керування конфігурацією.

У третьому розділі описано реалізацію всіх компонентів системи. Розроблено спільну бібліотеку libs з динамічною конфігурацією через SSE, реєстр торгових мереж ShopRegistry з підтримкою трьох мереж – АТБ, Varus та Fora, модуль збору даних з обходом антибот-захисту через ротацію TLS профілів, LLM мікросервіс нормалізації товарів з системою регресійного тестування промптів, систему аутентифікації на базі JWT та PostgreSQL, email сервіс для відправки чеків та веб-інтерфейс на базі Leaflet.js.

					ІАЛЦ.467200.003 ПЗ	Арк.
						75
Зм.	Арк.	№ докум.	Підпис	Дата		

У четвертому розділі проведено тестування та демонстрацію роботи системи. Повний сценарій від реєстрації до отримання email чеку підтвердив коректність роботи всіх компонентів. Тестування адміністративних можливостей підтвердило що вимкнення торгової мережі миттєво відображається у всіх сервісах без перезапуску. Регресійне тестування промптів підтвердило ефективність розробленого підходу до контролю якості групування товарів.

Всі вимоги сформульовані у технічному завданні були виконані – система забезпечує автоматизований пошук товарів у реальному часі, класифікацію та нормалізацію найменувань за допомогою LLM, порівняння цін на рівні повного кошика з урахуванням геолокації користувача та можливість масштабування системи.

					ІАЛЦ.467200.003 ПЗ	Арк.
						76
Зм.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. FastAPI documentation [Електронний ресурс] // Sebastián Ramírez. – 2024. – Режим доступу до ресурсу: <https://fastapi.tiangolo.com/>
2. Taskiq documentation [Електронний ресурс] // Taskiq contributors. – 2024. – Режим доступу до ресурсу: <https://taskiq-python.github.io/>
3. curl_cffi — Python binding for curl-impersonate [Електронний ресурс] // yifeikong. – 2024. – Режим доступу до ресурсу: https://github.com/yifeikong/curl_cffi
4. LiteLLM documentation [Електронний ресурс] // BerriAI. – 2024. – Режим доступу до ресурсу: <https://docs.litellm.ai/>
5. PocketBase documentation [Електронний ресурс] // Gani Georgiev. – 2024. – Режим доступу до ресурсу: <https://pocketbase.io/docs/>
6. Redis documentation [Електронний ресурс] // Redis Ltd. – 2024. – Режим доступу до ресурсу: <https://redis.io/docs/>
7. asyncpg — A fast PostgreSQL Database Client Library for Python [Електронний ресурс] // MagicStack. – 2024. – Режим доступу до ресурсу: <https://magicstack.github.io/asyncpg/>
8. Leaflet.js documentation [Електронний ресурс] // Vladimir Agafonkin. – 2024. – Режим доступу до ресурсу: <https://leafletjs.com/reference.html>
9. Docker documentation [Електронний ресурс] // Docker Inc. – 2024. – Режим доступу до ресурсу: <https://docs.docker.com/>
10. Python asyncio documentation [Електронний ресурс] // Python Software Foundation. – 2024. – Режим доступу до ресурсу: <https://docs.python.org/3/library/asyncio.html>
11. OpenStreetMap [Електронний ресурс] // OpenStreetMap contributors. – 2024. – Режим доступу до ресурсу: <https://www.openstreetmap.org/>

					ІАЛЦ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		77

- 12.JSON Web Token Introduction [Електронний ресурс] // Auth0. – 2024. – Режим доступу до ресурсу: <https://jwt.io/introduction>
- 13.bcrypt — Password hashing [Електронний ресурс] // Python Cryptographic Authority. – 2024. – Режим доступу до ресурсу: <https://github.com/pyca/bcrypt>
- 14.aiosmtplib documentation [Електронний ресурс] // Cole Maclean. – 2024. – Режим доступу до ресурсу: <https://aiosmtplib.readthedocs.io/>
- 15.Jinja2 documentation [Електронний ресурс] // Pallets Projects. – 2024. – Режим доступу до ресурсу: <https://jinja.palletsprojects.com/>
- 16.PostgreSQL documentation [Електронний ресурс] // PostgreSQL Global Development Group. – 2024. – Режим доступу до ресурсу: <https://www.postgresql.org/docs/>
- 17.TLS Fingerprinting with JA3 and JA3S [Електронний ресурс] // Salesforce Engineering. – 2019. – Режим доступу до ресурсу: <https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967/>
- 18.Server-Sent Events specification [Електронний ресурс] // WHATWG. – 2024. – Режим доступу до ресурсу: <https://html.spec.whatwg.org/multipage/server-sent-events.html>
- 19.Groq API documentation [Електронний ресурс] // Groq Inc. – 2024. – Режим доступу до ресурсу: <https://console.groq.com/docs/>
- 20.Microservices architecture style [Електронний ресурс] // Microsoft Azure Architecture Center. – 2024. – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices>

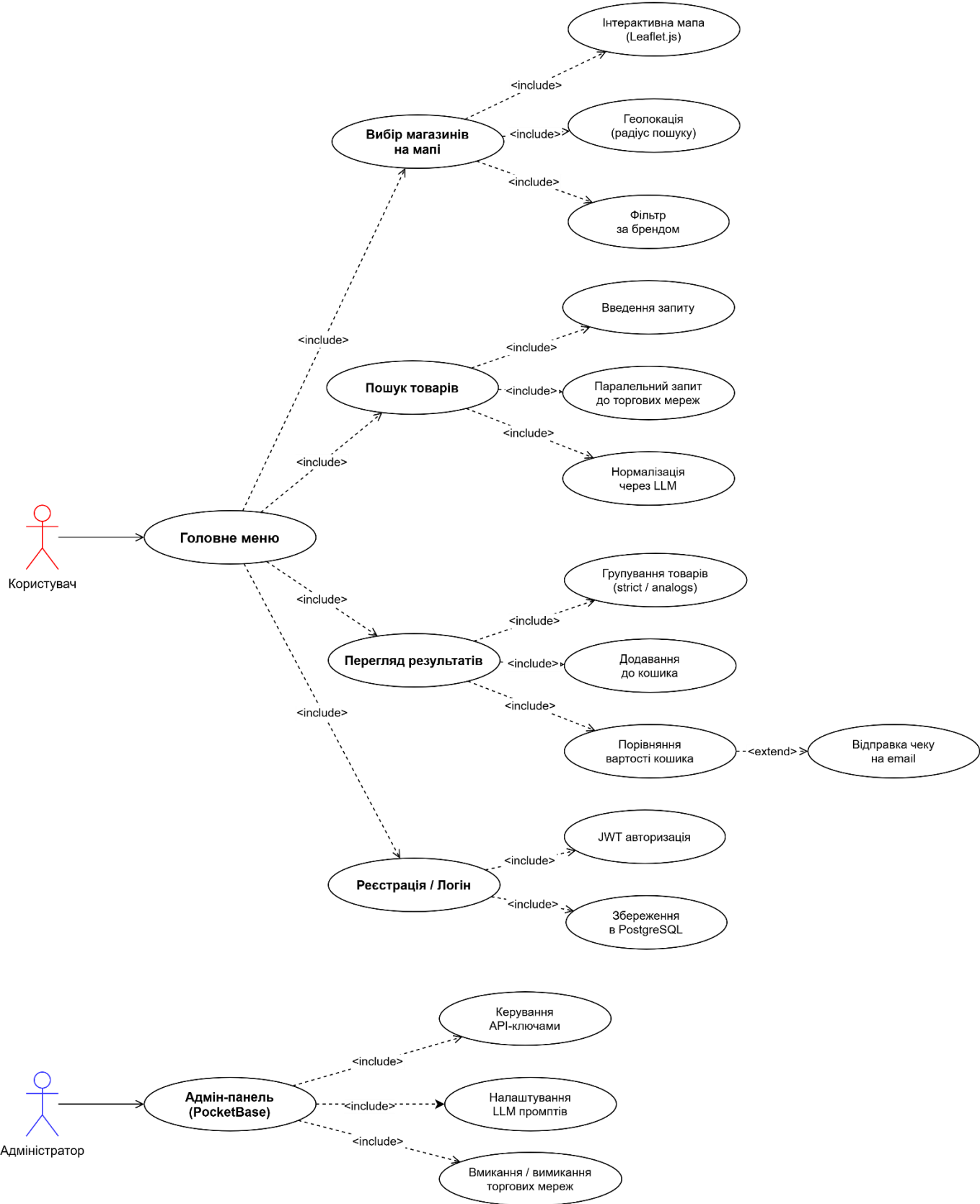
ДОДАТОК А

Система пошуку товарів з використанням розподілених обчислень та
штучного інтелекту для обробки товарів

Структурна схема системи
ІАЛЦ.467200.004 ДА

Аркушів 1

Київ 2026 р



					ІАЛЦ.467200.004 ДА						
		№ докум.	Підпис	Дата							
Розробив	Бережанський Д.В.				Система пошуку товарів з використанням розподілених обчислень та штучного інтелекту для обробки товарів Структурна схема системи	Літ.		Аркуш	Аркушів		
Перевірив	Гончаренко О.О							1	1		
						НТУУ “КПІ” ФІОТ ІО-24					
Н. Контр.	Гончаренко О.О										
Затвердив	Волокита А.М.										

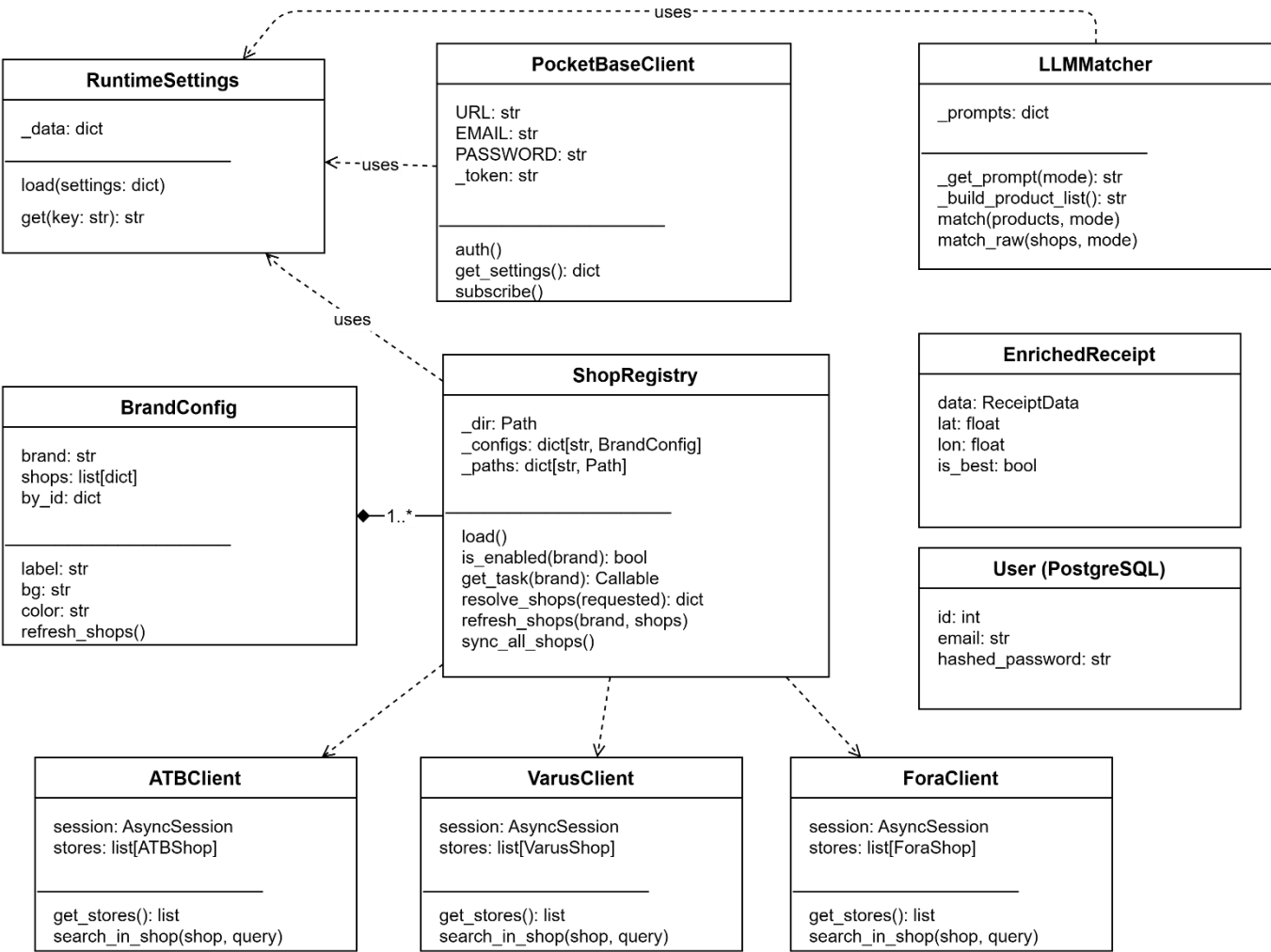
ДОДАТОК Б

Система пошуку товарів з використанням розподілених обчислень та
штучного інтелекту для обробки товарів

Функціональна схема
ІАЛЦ.467200.005 ДБ

Аркушів 1

Київ 2026 р



					ІАЛЦ.467200.005 ДБ						
		№ докум.	Підпис	Дата							
Розробив	Бережанський Д.В.				Система пошуку товарів з використанням розподілених обчислень та штучного інтелекту для обробки товарів Функціональна схема			Літ.	Аркуш	Аркушів	
Перевірив	Гончаренко О.О								1	1	
								НТУУ “КПІ” ФІОТ ІО-24			
Н. Контр.	Гончаренко О.О										
Затвердив	Волокита А.М.										

ДОДАТОК В

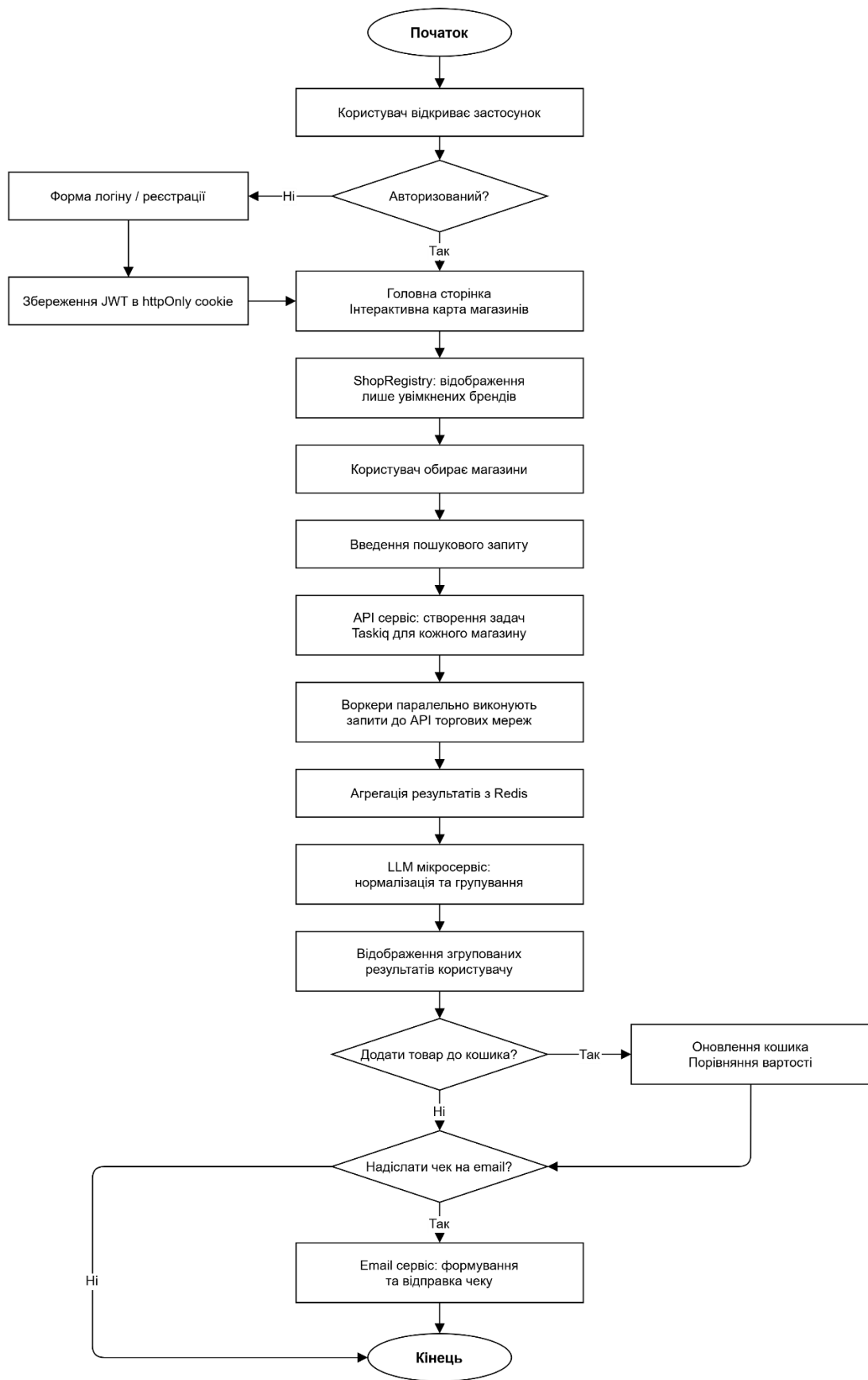
Система пошуку товарів з використанням розподілених обчислень та
штучного інтелекту для обробки товарів

Алгоритм дій програмного забезпечення

ІАЛЦ.467200.006 ДВ

Аркушів 1

Київ 2026 р



					ІАЛЦ.467200.006 ДВ				
		№ докум.	Підпис	Дата					
Розробив	Бережанський Д.В.				Система пошуку товарів з використанням розподілених обчислень та штучного інтелекту для обробки товарів Алгоритм дій програмного забезпечення	Літ.	Аркуш	Аркушів	
Перевірив	Гончаренко О.О						1	1	
						НТУУ “КПІ” ФІОТ			
Н. Контр.	Гончаренко О.О								
Затвердив	Волокита А.М.					ІО-24			

ДОДАТОК Г

Система пошуку товарів з використанням розподілених обчислень та
штучного інтелекту для обробки товарів

Текст програмного коду
ІАЛЦ.467200.007 ДГ

Аркушів 19

Київ 2026 р

```

from contextlib import asynccontextmanager

from fastapi import FastAPI
from fastapi.staticfiles import StaticFiles

from src.brands import registry
from libs.pb_client import start_pb
from src import db
from src.auth.router import router as auth_router
from src.email.router import router as email_router
from src.broker import broker
from src.routers import shops, search
from src.settings import BASE_DIR

@asynccontextmanager
async def lifespan(app: FastAPI):
    await start_pb()
    registry.load()
    await registry.sync_all_shops()
    await db.setup_pool()
    await broker.startup()
    yield
    await broker.shutdown()
    await db.teardown_pool()

app = FastAPI(title="Shop Search", lifespan=lifespan)

app.mount("/static", StaticFiles(directory=str(BASE_DIR / "src" / "static")),
name="static")

app.include_router(shops.router)
app.include_router(search.router)
app.include_router(auth_router)
app.include_router(email_router)

if __name__ == "__main__":
    import uvicorn
    uvicorn.run("src.main:app", host="0.0.0.0", port=8000, reload=True)

```

					ІАЛЦ.467200.007 ДГ			
		№ докум.	Підпис					
Розробив	Бережанський Д.В.				Система пошуку товарів з використанням розподілених обчислень та штучного інтелекту для обробки товарів Текст програмного коду	Літ.	Аркуш	Аркушів
Перевірив	Гончаренко О.О						1	19
						НТУУ “КПІ” ФІОТ		
Н. Контр.	Гончаренко О.О					ІО-24		
Затвердив	Волокита А.М.							

```

from itertools import cycle
import os
from pathlib import Path
from dotenv import load_dotenv
load_dotenv()

REDIS_URL = os.getenv("REDIS_URL", "redis://localhost:6379")

BASE_DIR = Path(__file__).parent.parent.resolve()
SHARED_DATA = BASE_DIR / "shared_data"
SHARED_DATA.mkdir(exist_ok=True)

TEMPLATES_DIR = BASE_DIR / "src" / "templates"
TEMPLATES_DIR.mkdir(exist_ok=True)

BROWSER_TYPES = [
    "chrome99", "chrome100", "chrome101", "chrome104",
    "chrome107", "chrome110", "chrome116", "chrome119",
    "chrome120", "chrome123", "chrome124", "chrome131",
    "chrome133a", "chrome136", "chrome142", "chrome145",
    "chrome146", "chrome99_android", "chrome131_android"
]
BROWSER_TYPES_CYCLE = cycle(BROWSER_TYPES)

ATB_SHOPS_CACHE = SHARED_DATA / "atb_shops.json"
VARUS_SHOPS_CACHE = SHARED_DATA / "varus_shops.json"

PB_ENCRYPTION_KEY=os.getenv("PB_ENCRYPTION_KEY")
PB_ADMIN_EMAIL=os.getenv("PB_ADMIN_EMAIL")
PB_ADMIN_PASSWORD=os.getenv("PB_ADMIN_PASSWORD")
PB_URL = os.getenv("PB_URL")

JWT_ALGORITHM = "HS256"
JWT_EXPIRE_MINUTES = 60 * 24 * 7 # 1 week

if __name__ == '__main__':
    print(BASE_DIR)

import asyncpg

```

					ІАЛЦ.467200.007 ДГ	Арк.
						2
Зм.	Арк.	№ докум.	Підпис	Дата		

```

from libs.pb_client import runtime

_pool: asyncpg.Pool | None = None

_CREATE_TABLES = """
CREATE TABLE IF NOT EXISTS users (
    id          SERIAL PRIMARY KEY,
    email       TEXT UNIQUE NOT NULL,
    hashed_password TEXT NOT NULL,
    created_at  TIMESTAMPTZ DEFAULT NOW()
);
"""

async def setup_pool() -> None:
    global _pool
    _pool = await asyncpg.create_pool(runtime.get("DATABASE_URL"))
    print(f"DATABASE_URL: {runtime.get("DATABASE_URL")}", flush=True)
    async with _pool.acquire() as conn:
        await conn.execute(_CREATE_TABLES)

async def teardown_pool() -> None:
    if _pool:
        await _pool.close()

async def get_conn():
    async with _pool.acquire() as conn:
        yield conn

from taskiq import TaskiqEvents, TaskiqState
from taskiq_redis import ListQueueBroker, RedisAsyncResultBackend

from src.settings import REDIS_URL

broker = ListQueueBroker(REDIS_URL).with_result_backend(
    RedisAsyncResultBackend(REDIS_URL)
)

@broker.on_event(TaskiqEvents.WORKER_STARTUP)
async def on_startup(state: TaskiqState) -> None:
    from libs.pb_client import start_pb

```

					ІАЛЦ.467200.007 ДГ	Арк.
						3
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    from src.brands import registry
    await start_pb()
    registry.load()

from src.broker import broker

from src.brands.atb.client import AtbClient
from src.brands.fora.client import ForaClient
from src.brands.varus.client import VarusClient

from src.models.shops import ATBShop, VarusShop, ForaShop

def _product_dict(p) -> dict:
    return {
        "id":          p.id,
        "name":         p.name,
        "price":        p.price,
        "special_price": p.special_price,
        "in_stock":     p.in_stock,
        "url":          p.url,
        "image_url":    p.image_url,
    }

@broker.task
async def search_atb_task(shop: dict, query: str) -> dict:
    client = AtbClient()
    products = await client.search_in_shop(ATBShop(**shop), query)
    return {
        "shop_id":    shop["id"],
        "shop_name":  shop["short_name"],
        "brand":      "atb",
        "products":   [_product_dict(p) for p in products],
    }

@broker.task
async def search_varus_task(shop: dict, query: str) -> dict:
    client = VarusClient()
    products = await client.search_in_shop(VarusShop(**shop), query)
    return {
        "shop_id":    shop["id"],

```

					ІАЛЦ.467200.007 ДГ	Арк.
						4
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        "shop_name": shop["short_name"],
        "brand":      "varus",
        "products":  [_product_dict(p) for p in products],
    }

@broker.task
async def search_fora_task(shop: dict, query: str) -> dict:
    client = ForaClient()
    products = await client.search_in_shop(ForaShop(**shop), query)
    return {
        "shop_id":    shop["id"],
        "shop_name":  shop["short_name"],
        "brand":      "fora",
        "products":  [_product_dict(p) for p in products],
    }

import asyncio

import asyncpg
from curl_cffi import AsyncSession
from fastapi import APIRouter, Depends, HTTPException

from libs.pb_client import runtime
from src.brands import registry
from src.auth.deps import get_current_user
from src.schemas.search import MatchRequest, SearchRequest

router = APIRouter(prefix="/api")

@router.post("/search")
async def search(req: SearchRequest, _: asyncpg.Record = Depends(get_current_user)):
    grouped = registry.resolve_shops(req.shops)

    if not grouped:
        raise HTTPException(status_code=422, detail="No enabled shops matched the
request")

    jobs = await asyncio.gather(*[
        registry.get_task(brand).kiq(shop, req.query)
        for brand, shops in grouped.items()
        for shop in shops
    ])

```

					ІАЛЦ.467200.007 ДГ	Арк.
						5
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        results = await asyncio.gather(*[job.wait_result(timeout=30) for job in jobs])
        return [r.return_value for r in results]

@router.post("/match")
async def match(req: MatchRequest, _: AsyncPgRecord = Depends(get_current_user)):
    shops = [
        {
            "shop_id": shop.shop_id,
            "shop_name": shop.shop_name,
            "brand": shop.brand,
            "products": [p.model_dump() for p in shop.products],
        }
        for shop in req.results
    ]
    async with AsyncSession() as session:
        resp = await session.post(
            f"{runtime.get('LLM_SERVICE_URL')}/match",
            json={"shops": shops, "mode": req.mode},
            headers={"Authorization": f"Bearer {runtime.get('LLM_SERVICE_API_KEY')}"},
        )
        resp.raise_for_status()
    return resp.json()

import asyncpg
from fastapi import APIRouter, Depends
from fastapi.responses import FileResponse

from src.auth.deps import get_current_user, require_page_auth
from src.brands import registry
from src.settings import BASE_DIR

router = APIRouter()

@router.get("/login")
def login_page():
    return FileResponse(BASE_DIR / "src" / "static" / "login.html")

@router.get("/")
def index(_: None = Depends(require_page_auth)):
    return FileResponse(BASE_DIR / "src" / "static" / "store_map.html")

```

					ІАЛЦ.467200.007 ДГ	Арк.
						6
Зм.	Арк.	№ докум.	Підпис	Дата		

```

@router.get("/checkout")
def checkout(_: None = Depends(require_page_auth)):
    return FileResponse(BASE_DIR / "src" / "static" / "checkout.html")

@router.get("/api/shops")
def get_shops(_: asyncpg.Record = Depends(get_current_user)):
    return [
        {**shop, "brand": brand}
        for brand in registry.enabled_brands
        for shop in registry.shops_for(brand)
    ]

from dataclasses import dataclass

@dataclass
class ATBProduct:
    id: str
    name: str
    price: float
    in_stock: bool
    url: str
    image_url: str | None = None
    special_price: float | None = None

@dataclass
class VarusProduct:
    id: str
    name: str
    sku: str
    price: float
    in_stock: bool
    url: str
    image_url: str | None = None
    special_price: float | None = None
    special_price_discount: int | None = None
    special_price_to_date: str | None = None

@dataclass
class ForaProduct:
    id: str

```

					ІАЛЦ.467200.007 ДГ	Арк.
						7
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    name: str
    price: float
    special_price: float | None
    in_stock: bool
    url: str
    image_url: str | None

@dataclass
class Product:
    id: str
    store: str          # "atb" | "varus"
    name: str
    price: float
    in_stock: bool
    url: str
    image_url: str | None = None
    special_price: float | None = None

    @property
    def actual_price(self) -> float:
        return self.special_price or self.price

def _to_product(brand: str, p) -> Product:
    return Product(
        id=p.id,
        store=brand,
        name=p.name,
        price=p.price,
        in_stock=p.in_stock,
        url=p.url,
        image_url=p.image_url,
        special_price=p.special_price,
    )

def from_atb(p: ATBProduct) -> Product:
    return _to_product("atb", p)

def from_varus(p: VarusProduct) -> Product:
    return _to_product("varus", p)

def from_fora(p: ForaProduct) -> Product:

```

					ІАЛЦ.467200.007 ДГ	Арк.
						8
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        return _to_product("fora", p)

from dataclasses import dataclass

@dataclass
class VarusShop:
    id: int
    tms_id: str
    short_name: str
    lat: float
    lon: float
    address: str

@dataclass
class ATBShop:
    id: int
    short_name: str
    lat: float
    lon: float
    address: str
    worktime: str

@dataclass
class ForaShop:
    id: int
    short_name: str
    lat: float
    lon: float
    address: str

from src.broker import broker
from src.brands import registry
from src.email.schemas import ReceiptData
from src.email.service import EnrichedReceipt, send_receipt_email

def _enrich(receipts: list[ReceiptData], shop_ids: list[dict]) ->
list[EnrichedReceipt]:
    coords: dict[str, tuple[float | None, float | None]] = {}
    for s in shop_ids:
        brand, sid = s["brand"], s["id"]
        shop = registry.by_id(brand, sid)

```

					ІАЛЦ.467200.007 ДГ	Арк.
						9
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        key = f"{brand}_{sid}"
        coords[key] = (shop["lat"], shop["lon"]) if shop else (None, None)

sorted_receipts = sorted(
    receipts,
    key=lambda r: (-(r.available_count == r.total_count), r.total),
)
best_total = next(
    (r.total for r in sorted_receipts if r.available_count == r.total_count),
    None,
)

result = []
for r in sorted_receipts:
    lat, lon = coords.get(r.shop_key, (None, None))
    is_best = best_total is not None and r.total == best_total and
r.available_count == r.total_count
    result.append(EnrichedReceipt(data=r, lat=lat, lon=lon, is_best=is_best))
return result

@broker.task
async def send_receipt_task(to: str, receipts: list[dict], shop_ids: list[dict]) ->
None:
    parsed = [ReceiptData(**r) for r in receipts]
    enriched = _enrich(parsed, shop_ids)
    await send_receipt_email(to, enriched)

import os
from dotenv import load_dotenv
from dataclasses import dataclass
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

import aiosmtplib
from jinja2 import Environment, FileSystemLoader

from src.brands import registry
from src.settings import TEMPLATES_DIR
from src.email.schemas import ReceiptData
load_dotenv()

```

					ІАЛЦ.467200.007 ДГ	Арк.
						10
Зм.	Арк.	№ докум.	Підпис	Дата		

```

SMTP_HOST = os.getenv("SMTP_HOST")
SMTP_PORT = int(os.getenv("SMTP_PORT", "587"))
SMTP_USER = os.getenv("SMTP_USER")
SMTP_PASSWORD = os.getenv("SMTP_PASSWORD")
SMTP_FROM = os.getenv("SMTP_FROM")

def _make_jinja_env() -> Environment:
    env = Environment(loader=FileSystemLoader(str(TEMPLATES_DIR)), autoescape=True)

    def plural_uk(n: int, one: str, few: str, many: str) -> str:
        if 11 <= n % 100 <= 14:
            return many
        if n % 10 == 1:
            return one
        if 2 <= n % 10 <= 4:
            return few
        return many

    def maps_url(r: "EnrichedReceipt") -> str:
        if r.lat and r.lon:
            return f"https://www.google.com/maps/search/?api=1&query={r.lat},{r.lon}"
        return
f"https://www.google.com/maps/search/?api=1&query={r.data.shop_name.replace(' ', '+')}}"

    env.filters["plural_uk"] = plural_uk
    env.filters["maps_url"] = maps_url
    return env

_jinja_env = _make_jinja_env()

@dataclass
class EnrichedReceipt:
    data: ReceiptData
    lat: float | None
    lon: float | None
    is_best: bool

def build_html(receipts: list[EnrichedReceipt], user_email: str) -> str:
    total_items = receipts[0].data.total_count if receipts else 0
    template = _jinja_env.get_template("receipt.html.jinja2")
    return template.render(

```

					ІАЛЦ.467200.007 ДГ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		11

```

        receipts=receipts,
        user_email=user_email,
        total_items=total_items,
        BRANDS=registry.email_brands(),
    )

async def send_receipt_email(to: str, receipts: list[EnrichedReceipt]) -> None:
    html = build_html(receipts, to)
    total_items = receipts[0].data.total_count if receipts else 0

    msg = MIMEMultipart("alternative")
    msg["Subject"] = f"GroceryScout - список покупок ({total_items} товарів)"
    msg["From"] = SMTP_FROM
    msg["To"] = to
    msg.attach(MIMEText(html, "html", "utf-8"))

    await aiosmtplib.send(
        msg,
        hostname=SMTP_HOST,
        port=SMTP_PORT,
        username=SMTP_USER,
        password=SMTP_PASSWORD,
        start_tls=True,
    )

from pydantic import BaseModel

class OfferData(BaseModel):
    name: str
    actual_price: float
    price: float
    special_price: float | None = None
    image_url: str | None = None
    url: str | None = None

class ReceiptLine(BaseModel):
    canonical: str
    offer: OfferData | None = None

class ReceiptData(BaseModel):
    shop_key: str

```

					ІАЛЦ.467200.007 ДГ	Арк.
						12
Зм.	Арк.	№ докум.	Підпис	Дата		

```

shop_name: str
brand: str
total: float
available_count: int
total_count: int
lines: list[ReceiptLine]

class SendReceiptRequest(BaseModel):
    receipts: list[ReceiptData]
    shop_ids: list[dict] # [{"brand": "atb", "id": 964}, ...]

import asyncpg
from fastapi import APIRouter, Depends, HTTPException, status

from src.auth.deps import get_current_user
from src.email.schemas import SendReceiptRequest
from src.email.tasks import send_receipt_task

router = APIRouter(prefix="/api")

@router.post("/send-receipt", status_code=status.HTTP_202_ACCEPTED)
async def send_receipt(
    req: SendReceiptRequest,
    current_user: asyncpg.Record = Depends(get_current_user),
):
    if not req.receipts:
        raise HTTPException(status_code=400, detail="Empty cart")

    await send_receipt_task.kiq(
        to=current_user["email"],
        receipts=[r.model_dump() for r in req.receipts],
        shop_ids=req.shop_ids,
    )
    return {"ok": True}
"""

```

Shop registry - single source of truth for all shop configurations.

Adding a new shop chain:

2. Add {BRANDNAME}_META = {"label": "...", "bg": "...", "color": "..."} in

PocketBase settings collection

3. Add src/<brand>_api/client.py implementing search_in_shop()

					ІАЛЦ.467200.007 ДГ	Арк.
						13
Зм.	Арк.	№ докум.	Підпис	Дата		

4. Add search_{brand}_task in src/tasks.py

File naming: {brand}_shops.json → brand is derived from filename automatically.

"""

```
import json
```

```
import importlib
```

```
from dataclasses import asdict
```

```
import logging
```

```
from pathlib import Path
```

```
from typing import Callable
```

```
from libs.pb_client import RuntimeSettings, runtime
```

```
from src.settings import SHARED_DATA
```

```
logger = logging.getLogger(__name__)
```

```
class BrandConfig:
```

```
    def __init__(self, brand: str, shops: list[dict], runtime: RuntimeSettings):
```

```
        self.brand: str = brand
```

```
        self._runtime = runtime
```

```
        self.shops: list[dict] = shops
```

```
        self.by_id: dict[int, dict] = {s["id"]: s for s in shops}
```

```
    def _meta(self) -> dict:
```

```
        meta_raw = self._runtime.get(f"{self.brand.upper()}_META", "{}")
```

```
        return json.loads(meta_raw) if meta_raw else {}
```

```
    @property
```

```
    def label(self) -> str:
```

```
        return self._meta().get("label", self.brand)
```

```
    @property
```

```
    def bg(self) -> str:
```

```
        return self._meta().get("bg", "#F3F4F6")
```

```
    @property
```

```
    def color(self) -> str:
```

```
        return self._meta().get("color", "#111827")
```

```
    @property
```

```
    def email_meta(self) -> dict:
```

					ІАЛЦ.467200.007 ДГ	Арк.
						14
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        meta = self._meta()
        return {
            "label": meta.get("label", self.brand),
            "bg": meta.get("bg", "#F3F4F6"),
            "color": meta.get("color", "#111827"),
        }

def refresh_shops(self, shops: list[dict], path: Path) -> None:
    """Update in-memory lookups and overwrite the JSON file."""
    self.shops = shops
    self.by_id = {s["id"]: s for s in shops}
    path.write_text(
        json.dumps(shops, indent=2, ensure_ascii=False),
        encoding="utf-8",
    )
    logger.info("ShopRegistry: refreshed %s (%d shops)", self.brand, len(shops))

class ShopRegistry:
    """
    Auto-discovers all *_shops.json files in `shops_dir`.
    Merges with meta from PocketBase runtime ({BRAND}_META key).
    Consults runtime `enabled_brands` to decide which brands are active.

    Startup order (important):
        await start_pb()      # runtime must be populated first
        registry.load()

    Usage:
        registry.is_enabled("atb")
        registry.shops_for("varus")
        registry.resolve_shops(req.shops)  # filters to enabled only
        registry.get_task("atb")           # returns Taskiq task
        registry.refresh_shops("varus", new_shops)
    """

    def __init__(self, shops_dir: Path, runtime: "RuntimeSettings"):
        self._dir = shops_dir
        self._runtime = runtime
        self._configs: dict[str, BrandConfig] = {}
        # Track file paths for refresh writes
        self._paths: dict[str, Path] = {}

```

```

def load(self) -> None:
    if not self._dir.exists():
        raise FileNotFoundError(f"shops_dir not found: {self._dir}")

    # Create missing shop files for known enabled brands
    enabled_raw = self._runtime.get("enabled_brands", "")
    if enabled_raw:
        for brand in {b.strip() for b in enabled_raw.split(",")}:
            path = self._dir / f"{brand}_shops.json"
            if not path.exists():
                path.write_text("[]", encoding="utf-8")
                logger.info("ShopRegistry: created empty %s", path.name)

    loaded: list[str] = []
    for path in sorted(self._dir.glob("*_shops.json")):
        brand = path.stem.replace("_shops", "")
        try:
            shops: list[dict] = json.loads(path.read_text(encoding="utf-8"))
            cfg = BrandConfig(brand, shops, self._runtime)
            self._configs[brand] = cfg
            self._paths[brand] = path
            loaded.append(brand)
        except Exception:
            logger.exception("ShopRegistry: failed to load %s", path)

    logger.info("ShopRegistry: loaded brands=%s", loaded)

def is_enabled(self, brand: str) -> bool:
    """
    Enabled when brand is loaded AND either:
    - `enabled_brands` key is absent/empty in PocketBase (all on), or
    - brand appears in the comma-separated `enabled_brands` value.
    """
    if brand not in self._configs:
        return False
    enabled = self._runtime.get("enabled_brands", "")
    if not enabled:
        return True
    return brand in {b.strip() for b in enabled.split(",")}

```

					ІАЛЦ.467200.007 ДГ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		16

```

@property
def enabled_brands(self) -> list[str]:
    return [b for b in self._configs if self.is_enabled(b)]

def shops_for(self, brand: str) -> list[dict]:
    return self._get(brand).shops

def by_id(self, brand: str, shop_id: int) -> dict | None:
    return self._get(brand).by_id.get(shop_id)

def config(self, brand: str) -> BrandConfig:
    return self._get(brand)

def all_configs(self) -> dict[str, BrandConfig]:
    return dict(self._configs)

def refresh_shops(self, brand: str, shops: list[dict]) -> None:
    """
    Update in-memory lookups and persist to JSON.
    Clients call this instead of writing the file themselves:

        registry.refresh_shops("varus", [asdict(s) for s in stores])
    """
    cfg = self._get(brand)
    cfg.refresh_shops(shops, self._paths[brand])

async def sync_all_shops(self) -> None:
    for brand in self.enabled_brands:
        try:
            module = importlib.import_module(f"src.brands.{brand}.client")
            client_class = getattr(module, f"{brand.capitalize()}Client")
            client = client_class()
            shops = await client.get_stores()
            self.refresh_shops(brand, [asdict(s) for s in shops])
        except Exception:
            logger.exception("Failed to sync shops for brand '%s'", brand)

def email_brands(self) -> dict[str, dict]:
    """Drop-in for the old hardcoded BRANDS dict in email/sender.py."""
    return {brand: cfg.email_meta for brand, cfg in self._configs.items()}

```

					ІАЛЦ.467200.007 ДГ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		17

```

def resolve_shops(self, requested: list) -> dict[str, list[dict]]:
    """
    Given a list of ShopRef(brand, id) from a search request,
    returns { brand: [shop_dict, ...] } filtered to enabled brands only.
    Items can be dicts or objects with .brand / .id attributes.
    """
    grouped: dict[str, list[dict]] = {}
    for ref in requested:
        brand = ref["brand"] if isinstance(ref, dict) else ref.brand
        shop_id = ref["id"] if isinstance(ref, dict) else ref.id

        if not self.is_enabled(brand):
            continue
        shop = self.by_id(brand, shop_id)
        if shop:
            grouped.setdefault(brand, []).append(shop)
    return grouped

def get_task(self, brand: str) -> Callable:
    """
    Returns the Taskiq task for `brand` by convention:
    search_{brand}_task must exist in src/tasks.py
    """
    from src import tasks # late import - avoids circular deps
    task_name = f"search_{brand}_task"
    task = getattr(tasks, task_name, None)
    if task is None:
        raise KeyError(
            f"No task '{task_name}' in src.tasks. "
            f"Add it to support brand '{brand}'."
        )
    return task

def _get(self, brand: str) -> BrandConfig:
    if brand not in self._configs:
        raise KeyError(
            f"Unknown brand '{brand}'. "
            f"Known: {list(self._configs)}. "
            f"Add shared_data/shops/{brand}_shops.json to register it."
        )
    return self._configs[brand]

```

```

registry = ShopRegistry(SHARED_DATA, runtime)

if __name__ == '__main__':
    import asyncio
    from libs.pb_client import start_pb

    async def main():
        await start_pb()
        registry.load()

        for brand in registry.enabled_brands:
            cfg = registry.config(brand)
            print(f"{brand}: {cfg.label} - {len(cfg.shops)} shops")
            print(f"  meta: bg={cfg.bg} color={cfg.color}")
            print(f"  first shop: {cfg.shops[0] if cfg.shops else 'none'}")

    asyncio.run(main())

```

					ІАЛЦ.467200.007 ДГ	Арк.
						19
Зм.	Арк.	№ докум.	Підпис	Дата		