

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

**«Аналіз ринкового пошуку на базі веб-сайту
електротехніки»**

Виконав студент групи ІПЗм-23-2
ГУЛАК Іван Володимирович

Керівник роботи:
САВІЦЬКИЙ Роман Святославович

Рецензент:
СИДОРОВ Олег Іванович

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«26» лютого 2024р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ГУЛАК Іван Володимирович**

Керівник роботи: **САВІЦЬКИЙ Роман Святославович**

Тема роботи: **«Аналіз ринкового пошуку на базі веб-сайту електротехніки»**,
затверджена Наказом закладу вищої освіти від «23» лютого 2024 р. №74с

Вихідні дані для роботи: метою є дослідження розробки автоматизованого підходу для аналізу ринку на основі даних конкурентів, з подальшим використанням зібраної та проаналізованої інформації для відображення в асортименті інтернет-магазину електротехніки. Об'єктом дослідження є процес формування та оптимізації асортименту інтернет-магазинів електротехніки в умовах високої конкуренції. Предметом дослідження виступають методи автоматизованого збору, аналізу та обробки ринкових даних з вебсайтів конкурентів, а також впровадження інструментів штучного інтелекту для покращення асортименту і цінової політики інтернет-магазину

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Савіцький Р.С.	02.03.2024	02.03.2024
2	Савіцький Р.С.	04.04.2024	04.04.2024
3	Савіцький Р.С.	02.05.2024	02.05.2024

РЕФЕРАТ

Випускна кваліфікаційна робота складається з програмної реалізації автоматизованого підходу до аналізу ринку для інтернет-магазинів електротехніки, а також пояснювальної записки. Пояснювальна записка до дипломної роботи містить 83 сторінки, 23 ілюстрації, 2 таблиці, 4 додатки та 41 літературне джерело.

Метою роботи є дослідження та розробка автоматизованої системи, яка забезпечує ефективний збір, аналіз та обробку даних конкурентів для оптимізації асортименту та підвищення конкурентоспроможності інтернет-магазину електротехніки.

Основним інструментарієм при розробці є мови програмування JavaScript, фреймворки Next.js та Nest.js. Для реалізації єдиного, зрозумілого та ергономічного інтерфейсу використано бібліотеки TailwindCSS та Headless UI. Для оптимального збереження даних з великою кількістю зв'язків обрано базу даних MongoDB, яка забезпечує гнучке моделювання даних. Для зручного та декларативного визначення структури даних, а також безпечного доступу до них, використано бібліотеку Prisma, яка інтегрується з MongoDB та спрощує роботу з базою даних у середовищі JavaScript.

В роботі описано процес аналізу технічного завдання та вимог системи, оглянуто існуюче програмне забезпечення за тематикою проєкту. Також описано процес розробки бази даних та реалізації автоматизованої системи аналізу ринку. Далі зазначено порядок встановлення та налаштування системи, опис інтерфейсу вебмагазину та приклади тестування.

КЛЮЧОВІ СЛОВА: ВЕБЗАСТОСУНОК, АНАЛІЗ РИНКУ, АВТОМАТИЗОВАНИЙ ЗБІР, ОБРОБКА ДАНИХ, NEXTJS, NESTJS, MONGODB.

ABSTRACT

The graduation qualification work consists of the software implementation of an automated approach to market analysis for online electronics stores, as well as an explanatory report. The explanatory report comprises 83 pages, 22 illustrations, 2 tables, 4 appendices, and 31 references.

The objective of the work is to research and develop an automated system that ensures efficient collection, analysis, and processing of competitor data to optimize the product assortment and enhance the competitiveness of an online electronics store.

The primary tools used in the development include the programming languages JavaScript along with the frameworks Next.js and Nest.js. To create a unified, intuitive, and ergonomic interface, libraries such as TailwindCSS and Headless UI were utilized.

For the optimal storage of data with numerous relationships, the MongoDB database was selected, offering flexible data modeling. To enable convenient and declarative data structure definition, as well as secure access, the Prisma library was employed. Prisma integrates seamlessly with MongoDB and simplifies database management in JavaScript environments.

The work details the process of analyzing the technical requirements and system specifications, provides an overview of existing software solutions related to the project's topic, and describes the development of the database and implementation of the automated market analysis system. Additionally, it outlines the system's installation and configuration procedures, presents an overview of the web store's interface, and includes examples of system testing.

KEYWORDS: WEB APPLICATION, MARKET ANALYSIS, AUTOMATED DATA COLLECTION, DATA PROCESSING, NEXT JS, NEST JS, MONGO DB.

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ТЕХНОЛОГІЙ ТА ПІДХОДІВ ДО РИНКОВОГО ПОШУКУ В ЕЛЕКТРОННІЙ КОМЕРЦІЇ	9
1.1. Постановка задачі.....	9
1.2. Аналіз сучасних підходів до ринкового пошуку в електронній комерції.....	10
1.3. Інструменти для збору, фільтрації та обробки зібраних даних	13
1.4. Використання AI для аналізу ринкових даних	18
Висновки до першого розділу.	24
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ АНАЛІЗУ РИНКОВОГО ПОШУКУ	26
2.1. Проєктування архітектури системи.....	26
2.2. Інтеграція AI для автоматизованого аналізу ринкового пошуку....	34
2.3. Вибір мови програмування для парсеру та їх ключові відмінності	40
2.4. Інтеграція результатів аналізу в систему популярних товарів	43
Висновки до другого розділу	51
РОЗДІЛ 3. ДОСЛІДЖЕННЯ СИСТЕМИ ТА АНАЛІЗ РЕЗУЛЬТАТІВ....	53
3.1. Перевірка працездатності системи аналізу ринкового пошуку	53
3.2. Оцінка ефективності застосованих методів і алгоритмів.....	56
3.3. Виявлені обмеження системи та перспективи її вдосконалення	58
Висновки до третього розділу	61
ВИСНОВКИ.....	63
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	65
ДОДАТКИ.....	69

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI (Artificial Intelligence) – штучний інтелект

API (Application Program Interface) – прикладний програмний інтерфейс

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) – це система, яка використовується для перевірки, чи є користувач людиною, а не автоматизованою програмою (ботом)

CSV (Comma-Separated Values) — це текстовий формат для зберігання табличних даних, де значення в рядках розділені комами

HTTP (HyperText Transfer Protocol) — це протокол для передачі гіпертекстових документів через Інтернет

IP (Internet Protocol) – це набір правил, що визначають, як дані передаються через мережу Інтернет

JIT (Just-In-Time) – різновид компіляції, який поєднує елементи компіляції та інтерпретації

JSON (JavaScript Object Notation) – формат обміну даними, який використовує текстове представлення для зберігання та передачі структурованої інформації

NLP (Natural Language Processing) – це галузь штучного інтелекту, яка займається обробкою та аналізом природних мов

REST (Representational State Transfer) – це стиль архітектури для створення вебсервісів, який базується на використанні стандартних HTTP-запитів (GET, POST, PUT, DELETE) для обміну даними між клієнтом і сервером.

XML (eXtensible Markup Language) – це мова розмітки, яка використовується для зберігання, передачі та опису структурованих даних у вигляді текстового формату

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

ВСТУП

Актуальність теми. У сучасних умовах жорсткої конкуренції на ринку електротехніки інтернет-магазини мають постійно адаптуватися до змін ринку, слідкувати за трендами та пропонувати своїм клієнтам найвигідніші умови. Розвиток інформаційних технологій і зростання популярності онлайн-покупок сприяють тому, що більшість споживачів здійснюють свої покупки через інтернет-магазини, шукаючи зручності, доступності та оптимальні цінові пропозиції.

Для успішного існування, інтернет-магазинам необхідно підтримувати актуальний асортимент, аналізувати поведінку конкурентів, визначати трендові товари та швидко реагувати на зміни попиту. Ефективність цього процесу залежить від використання сучасних інструментів автоматизації, аналізу даних і штучного інтелекту, адже вони дають змогу швидко збирати, обробляти й інтерпретувати великі обсяги інформації.

Застосування автоматизованих систем для аналізу ринкових даних дозволяє інтернет-магазинам формувати конкурентоспроможний асортимент, адаптувати цінову політику та залишатися актуальними на ринку. Це особливо важливо в сегменті електротехніки, де нові технології та товари з'являються з високою швидкістю.

Таким чином, актуальність роботи зумовлена необхідністю використання сучасних технологій для підвищення ефективності управління інтернет-магазинами електротехніки, що дасть змогу зберігати конкурентні позиції та забезпечувати їх подальший розвиток.

Метою роботи є дослідження та розробка автоматизованого підходу для аналізу ринку на основі даних конкурентів із подальшим використанням зібраної та проаналізованої інформації для формування конкурентоспроможного асортименту інтернет-магазину електротехніки.

Досягнення поставленої мети передбачає виконання таких завдань:

- дослідити сучасні методи збору та аналізу ринкових даних;
- вивчити підходи до автоматизованого аналізу даних конкурентів;

- розробити алгоритми для обробки даних і визначення трендів у сфері електротехніки;

- створити структуру для збирання, зберігання та обробки інформації з конкурентних вебсайтів;

- реалізувати автоматизований підхід для оновлення асортименту інтернет-магазину електротехніки.

Об'єктом дослідження є процес аналізу ринку та формування асортименту інтернет-магазинів електротехніки в умовах високої конкуренції.

Предметом дослідження є методи автоматизованого збору, аналізу та обробки ринкових даних з вебсайтів конкурентів.

Наукова новизна отриманих результатів полягає в такому:

- розроблено автоматизований підхід аналізу ринкових даних, що забезпечує точне та регулярне оновлення асортименту інтернет-магазину електротехніки;

- досліджено сучасні методи збору та обробки даних з вебсайтів конкурентів у контексті їх застосування для аналізу ринку;

- запропоновано інтеграцію автоматизованого аналізу ринкових даних у систему управління асортиментом інтернет-магазину.

РОЗДІЛ 1. АНАЛІЗ ТЕХНОЛОГІЙ ТА ПІДХОДІВ ДО РИНКОВОГО ПОШУКУ В ЕЛЕКТРОННІЙ КОМЕРЦІЇ

1.1. Постановка задачі

Стрімкий розвиток цифрових технологій та активне впровадження інтернет-комерції сприяє зростанню конкуренції між онлайн-магазинами та починає досягати максимального рівня.

Високий рівень сервісу, широкий асортимент товарів, конкурентоспроможні ціни, оперативність у задоволенні клієнтських потреб стали базовими потребами, на які очікує клієнт при користуванні сервісом. Для того аби підприємствам залишатися конкурентоздатними необхідно використовувати автоматизовані інструменти для збору та аналізу ринкових даних, що безперечно дасть змогу оперативно реагувати на зміну ринкових трендів і уподобань клієнтів.

Кваліфікаційна магістерська робота на тему «Аналіз ринкового пошуку на базі веб-сайту електротехніки» має на меті створення інтегрованого рішення для автоматизації процесів збору, аналізу та обробки ринкових даних із конкурентних джерел. Основна мета роботи полягає в дослідженні методів збору та обробки даних необхідних для аналізу ринку; створення системи для реалізації роботи дослідження, а також автоматизація роботи систему аналізу ринку. Такий підхід допоможе покращити управління асортиментом і ціноутворенням в інтернет-магазині, а також дозволить швидко реагувати на зміни в умовах конкурентного ринку.

Автоматизований підхід дає можливість не лише зекономити час і ресурси, але й зменшити ймовірність допуску помилок, які зазвичай виникають через людський фактор.

Актуальність теми роботи визначається потребою в ефективних інструментах, що здатні зробити процес збору та аналізу даних автоматизованим. Динамічна зміна ринкових трендів та зростання конкуренції роблять рішення ключовим, для забезпечення конкурентоспроможності компаній.

Впродовж роботи, був проведений аналіз автоматизації збору даних та подальшої інтеграції отриманих результатів у бізнес-процеси. Для цього використовувалися передові технології, такі як MongoDB, Prisma, Selenium, Nest.js і Next.js, що забезпечують гнучкість, масштабованість та високу продуктивність розроблених рішень.

Для онлайн-магазинів електротехніки важливо не лише відстежувати асортимент і ціни конкурентів, але й ефективно використовувати дані для прийняття стратегічних рішень. Автоматизація процесів дозволяє зменшити часові та фінансові витрати, одночасно підвищуючи точність аналізу та швидкість адаптації до ринкових змін.

Результатом роботи стала розробка системи, яка забезпечує ефективний збір, аналіз та обробку ринкових даних, що дозволяє оптимізувати асортимент інтернет-магазину електротехніки, підвищити конкурентоспроможність підприємства та сприяти довгостроковому розвитку.

1.2. Аналіз сучасних підходів до ринкового пошуку в електронній комерції

Цифрова трансформація, зокрема в електронній комерції, значно змінила підходи до аналізу ринкових процесів і стратегій залучення клієнтів. У сучасному світі інформаційні технології та аналіз великих даних стали основою успішних бізнес-моделей. Раніше ринкові дослідження проводилися традиційними методами, такими як телефонні опитування або ж паперові анкети. Сьогодні ж, даний підхід став обмеженим через свою низьку швидкість, високу вартість та обмежений обсяг збору даних [2]. Зокрема, виникла проблема неможливості задовольнити потреби швидкого отримання даних в умовах змінного ринку, особливо в умовах електронної комерції, де конкуренція постійно зростає. Такі підходи вважаються не ефективними та застарілими. Але на зміну прийшли технології автоматизованого збору й аналізу даних, які значно підвищили ефективність та точність результатів, а головне з кожним днем прогресують та відкривають нові можливості для вдосконалення процесу ринкового аналізу.

Одним із інструментів сучасного аналізу ринку є вебскрейпінг [3]. За допомогою вебскрейпінгу можна автоматизувати процес збору даних з вебсайтів, що дає змогу отримувати актуальну інформацію в реальному часі. Наприклад, такі інструменти, як Selenium [4] та Puppeteer, дозволяють працювати з динамічними вебсторінками, автоматизуючи процес взаємодії з ними і збираючи дані, що генеруються за допомогою мови програмування JavaScript [5]. Можливість постійно моніторити ціни, асортимент та інші важливі аспекти ринку – є основною перевагою вебскрейпінгу. Використання вебскрейпінгу дозволяє компаніям бути більш гнучкими та швидко реагувати на ринкові зміни.

Важливим аспектом сучасних підходів до ринкового пошуку є аналіз соціальних медіа та емоційна аналітика. Технологія NLP [6], яка займається обробкою природної мови дозволяє компаніям не лише стежити за уподобаннями користувачів, але й передбачати нові тренди, визначати настрої аудиторії та оптимізувати стратегії маркетингу. Використовуючи дану технологію, компанії можуть збирати статистичні дані про продукти чи бренди, глибше розуміти емоційну реакцію споживачів, що є важливим аспектом для налаштування персоналізованих пропозицій.

За допомогою застосування аналітики великих даних можна більш глибоко вивчати ринок і поведінку споживачів. Використовуючи машинне навчання та прогностичну аналітику, компанії можуть формувати більш точні прогнози змін на ринку, здійснювати персоналізацію пропозицій і створювати маркетингові стратегії, що ґрунтуються на конкретних, актуальних даних. Це дає змогу підвищити ефективність маркетингових кампаній та задовольнити індивідуальні потреби клієнтів, що, у свою чергу, сприяє збільшенню лояльності та продажів.

Картографія шляху клієнта є важливим інструментом для розуміння всіх точок взаємодії споживачів з брендом, починаючи від першого знайомства до завершення покупки. Картографія шляху клієнта дозволяє компаніям оптимізувати клієнтський досвід і підвищити ефективність взаємодії з клієнтами на кожному етапі їх шляху. Також, підхід дозволяє враховувати не лише прямі

взаємодії з брендом, але й аналізувати різні фактори, що впливають на поведінку споживачів.

Безперечно, кожен із розглянутих підходів (див. табл. 1.1) має свої переваги, але їх ефективність сильно залежить від типу бізнесу та цілей дослідження. Традиційні методи, такі як телефонні прозвони, різноманітні опитування або ж заповнення паперових анкет, можуть бути корисними для глибоких якісних досліджень, але часто не забезпечують потрібної швидкості й обсягу даних. Вебскрейпінг і аналіз соціальних медіа дають змогу отримати дані в реальному часі, однак потребують висококваліфікованих спеціалістів для налаштування і аналізу результатів. Використання великих даних і прогнозної аналітики надає компаніям змогу не лише аналізувати ринок, але й передбачати майбутні тенденції.

Таблиця 1.1

Таблиця огляду методів аналізу ринку та приклади можливого використання в майбутній розроблюваній системі.

Метод аналізу ринку	Приклад реалізації в розроблюваній системі	Планується залучити у системі (+/-)
Картографія шляху (Customer Journey Mapping)	Створення карти шляхів користувачів на сайті: від моменту першого відвідування до покупки. Це дозволяє виявити ключові точки взаємодії і вдосконалити UX/UI, а також підвищити конверсію.	-
Вебскрейпінг	Приклад реалізації в розроблюваній системі	Планується залучити у системі (+/-)
Картографія шляху (Customer Journey Mapping)	Парсинг сайту конкурента по визначених категоріях для збору, аналізу даних та визначення популярних товарів. Обробка та трансформація даних для подальшого сортування та відображення схожих товарів на головній сторінці в секції «Популярні товари»	+
Аналіз соціальних медіа та емоційна	Використання алгоритмів для аналізу коментарів, постів і відгуків на платформах (Facebook,	-

Продовження таблиці 1.1

аналітика	Instagram, Twitter) для визначення загальних настроїв щодо продуктів. Це може допомогти зрозуміти популярність товарів і прогнозувати попит.	
Використання великих даних та прогнозової аналітики	Використання історичних даних про покупки покупців для виявлення та показу сезонних продуктів. Проходження по товарам з власної БД, фільтрація даних по заданій умові, виявлення сезонних товарів та показ в секції «Сезонні товари» або в «В цей час купують»	+

Дана таблиця демонструє можливі методи та підходи аналізу ринку, а також чітко демонструє приклади реалізації даних підходів.

Отже, оглянувши та проаналізувавши популярні та ефективні на сьогоднішній день методи аналізу ринку, необхідно зробити вибір окремих методів та алгоритмів інтеграції методу в систему. Головне завдання дослідження – розробити систему, яка дасть змогу аналізувати ринок в напрямку електротехнічних товарів, використовуючи розглянуті вище методи та підходи щодо аналізу ринку. Потрібно інтегрувати вебскрейпінг для моніторингу конкурентів та трендових товарів, а також використовувати дані власної БД системи для генерації корисних підбірок товарів для користувачів, наприклад, підбірка сезонних товарів. Це дозволить швидко реагувати на зміни та підвищити продажі вебмагазину.

1.3. Інструменти для збору, фільтрації та обробки зібраних даних

В умовах стрімкого розвитку цифрових технологій та постійного зростання конкуренції на ринку, інструменти для збору та аналізу даних набувають вирішального значення. Одним із таких інструментів є розглянутий раніше вебскрейпінг. Інструмент дозволяє автоматизувати процес збору інформації з вебсайтів. Тобто, стає можливим сценарій парсингу даних із сайтів конкурентів в автоматичному режимі. Це дає ряд переваг компаніям, які

використовують інструмент, а саме: змогу краще розуміти ринкову ситуацію, стежити за конкурентами та адаптуватися до змін у реальному часі.

Основою вебскрейпінгу є використання сучасних технологій для вилучення даних, які можуть бути динамічними, складними для доступу або прихованими за захисними механізмами. Наприклад, Selenium і Puppeteer забезпечують ефективну роботу з вебконтентом, який генерується динамічно за допомогою JavaScript. Вони дозволяють автоматизувати процес взаємодії з вебсторінками, адже імітують поведінку реального користувача. Алгоритми машинного навчання можуть бути дуже зручними та ефективними під час роботи з великими обсягами даних. Вони не лише обробляють зібрану інформацію, але й допомагають знаходити приховані тренди, формувати прогнози та аналізувати поведінку конкурентів.

Серед переваг вебскрейпінгу можна виділити можливість виконувати аналіз цін у реальному часі. Дана можливість дозволяє компаніям оперативно змінювати власну цінову політику відповідно до змін у пропозиціях конкурентів. Крім того, завдяки моніторингу асортименту продукції конкурентів, підприємства можуть ідентифікувати найбільш популярні товари, аналізувати прогалини у власній пропозиції та швидко оновлювати свій каталог.

Слід бути обережним, адже під час використання вебскрейпінгу можна зустрітися з певними етичними та технічними викликами. Одним із головних аспектів є дотримання законодавчих норм. В багатьох країнах збирання даних із вебсайтів регулюється правовими актами, які встановлюють жорсткі обмеження на використання автоматизованих інструментів. Ігнорування таких правил може призвести до юридичних санкцій або порушення прав власників вебсайтів. Сьогодні, багато існуючих вебмагазинів знають про існування даного інструменту, тому вони намагаються захистити свій вебресурс від автоматичного збирання даних, використовуючи різноманітні механізми для захисту. Це можуть бути CAPTCHA, блокування IP-адрес чи обмеження швидкості доступу до даних.

Не менш важливим є питання точності та надійності отриманих даних. Автоматизовані системи можуть працювати некоректно через зміни у структурі

вебсайтів, помилки у коді або інші технічні проблеми. Тому створення ефективної архітектури, яка забезпечить гнучкість та адаптивність до змін, має ключове значення для успішного впровадження вебскрейпінгу.

Крім технічних аспектів, значну увагу необхідно приділити етичним стандартам. Збирання даних має здійснюватися з урахуванням конфіденційності та прав власників інформації. Використання даних без згоди може не лише завдати шкоди репутації компанії, але й порушити довіру клієнтів.

Використання вебскрейпінгу можна порівняти із процесом розвідки. Інструмент дозволяє компаніям отримати низку переваг: адаптуватися до змін на ринку, проактивно впливати на ринкову ситуацію; формувати стратегії розвитку та приймати обґрунтовані рішення. В умовах швидкоплинних змін у споживчих уподобаннях та високої конкуренції, такі можливості стають вирішальними для бізнесу, який прагне залишатися актуальним і конкурентоспроможним.

Інструмент має ряд проблем та складнощів, про які варто знати для більшої ефективності в застосуванні.

По-перше, багато вебсайтів використовують різноманітні методи захисту від автоматизованого збору даних, такі як CAPTCHA, обмеження кількості запитів з однієї IP-адреси або виявлення підозрілої активності. Це значно ускладнює процес збору даних, що інколи вимагає додаткових витрат зусиль з боку розробників ПЗ для обходу таких захисних механізмів. Гарним прикладом може стати використання проксі-серверів чи рішень для обходу CAPTCHA.

По-друге, структура та формат даних на вебсайтах можуть змінюватися без попередження, що може призвести до неактуальності існуючих парсерів. При налаштуванні збору даних зазвичай використовуються назви класів цільових елементів, однак їх періодична зміна назви або модифікація структури може зробити використання даного інструменту не результативним.

По-третє, варто пам'ятати про важливий нюанс, пов'язаний із правовими аспектами. Вебскрейпінг може порушувати умови використання сайтів, що тягне за собою юридичні наслідки. Саме тому, дуже важливо враховувати дану проблему та дотримуватися правових норм, працюючи з даними, отриманими з вебсайтів конкурентів.

Незважаючи на вищеописані труднощі та проблеми, вебскрейпінг має ряд важливих переваг, які роблять його корисним інструментом для електронної комерції.

Передусім, завдяки автоматизованому збору даних з вебсайтів конкурентів, підприємства отримують можливість ефективно аналізувати асортимент товарів, ціни, акції та інші фактори. Дана можливість значно допомагає у коригуванні стратегії ціноутворення та асортименту.

Окрім того, вебскрейпінг дозволяє автоматизувати процес збору великих обсягів даних – зробити процес регулярним, завдяки чому, значно знижуються витрати часу та ресурсів, якщо порівнювати із підходом ручного збору даних.

До того ж, зібрані дані можуть бути використані для прогнозування попиту на товари, зокрема для оцінки сезонних трендів та визначення найбільш популярних товарів на основі аналізу конкурентів.

Таким чином, розглянувши переваги та недоліки технології вебскрейпінгу, можна зробити висновок, що даний інструмент стає свого роду стратегічним активом для сучасних компаній, які прагнуть ефективно управляти своїми ресурсами та аналізувати ринок аби забезпечувати власний довгостроковий успіх.

Після збору даних важливим та невід’ємним завданням є обробка та фільтрація даних відповідно до цілей, для подальшого аналізу та використання в системах електронної комерції. В ході даного дослідження пропонується розглянути один із найпоширеніших форматів для збереження і передачі таких даних – JSON.

JSON дозволяє зберігати як прості, так і складні структури даних, що робить його ідеальним для представлення результатів парсингу (рис.1.1).

```
{
  "product_id": "12345",
  "name": "Телевізор Samsung 50\"",
  "price": 999.99,
  "category": "Електроніка",
  "rating": 4.5,
  "reviews": 250,
  "availability": true,
  "date_added": "2024-11-01"
}
```

Рис. 1.1 Приклад структури JSON для збереження даних про товари

Важливо фільтрувати та аналізувати зібрані дані для подальшого використання. Тому можна виділити кілька необхідних кроків, які можуть бути реалізовані для ефективного аналізу даних.

Передусім, фільтрація даних – процес виділення товарів, які відповідають певним критеріям, наприклад, таким як ціна, наявність або рейтинг. Фільтрація дозволяє обрати найбільш релевантні товари для відображення на вебсайті, що є корисним при формуванні списків «Популярні товари» або «Акції».

Окрім того можна використовувати статистичні методи чи алгоритми машинного навчання [7, 8]. Наприклад, за допомогою моделі машинного навчання можна класифікувати товари або, наприклад, визначати їх схожість із товарами конкурентів та прогнозувати попит на основі тенденцій.

До того ж, результати аналізу можуть бути використані для формування списків популярних товарів, що можуть бути відображені на вебсайті у відповідних секціях, наприклад, у розділі «Популярні товари».

Аналіз даних із використанням машинного навчання є доволі цікавою темою для вивчення з метою подальшого використання та інтеграції у систему. Наприклад, розглянемо можливі кроки аналізу за допомогою OpenAI [9] чи інших інструментів:

- зібрати дані про товари конкурентів і порівняти їх з товарами власного асортименту;
- використовувати інструменти аналізу для визначення найбільш релевантних товарів з метою включення їх в асортимент магазину.

Для ефективного використання зібраних даних необхідно порівнювати їх з власним асортиментом товарів. Це може дозволити обрати товари, які є найбільш релевантними, тобто які відповідають запитам користувачів. Гарним варіантом є інтеграція нейронної мережі ChatGPT [10], яка надає відкрите API для використання її методів та функцій. Інтеграція відбувається із використанням бібліотеки OpenAI, яка може бути дуже зручною для виконання поставлених в роботі завдань, адже бібліотека легко встановлюється та має детальну документацію з прикладами використання. Після збору та фільтрації даних з вебсайтів конкурентів виникне потреба в ефективному аналізі зібраних

продуктів, а також в їх відображенні в системі. Планується, показ товарів, які, за характеристиками, максимально відповідають отриманим даним. Це дозволить здійснювати коректний вебскрейпінг. Під словосполученням «коректний вебскрейпінг» мається на увазі аналізу та оптимізації асортименту на основі отриманої інформації, а не порушення конфіденційності даних конкурентів.

Отож, в результаті дослідження можна виділити ряд необхідних кроків для досягнення поставленої задачі:

1. Здійснення парсингу конкурентних сайтів з метою збору даних для подальшого аналізу.
2. Фільтрація отриманих даних за допомогою власних функцій та підготовлених інструментів.
3. Порівняння отриманих результатів із власними товарами за різними критеріями, такими як ціна, рейтинг, доступність, кількість коментарів.
4. Вибір товарів для відображення, які найбільше підходять за характеристиками до проаналізованих конкурентних товарів, з метою подальшого відображення в секціях «Популярні товари» або «Акційні пропозиції».

1.4. Використання AI для аналізу ринкових даних

Сучасні технології штучного інтелекту (AI) відіграють ключову роль у розробці інструментів для аналізу ринкових даних. Їх використання дозволяє автоматизувати складні процеси, забезпечувати точність аналізу та надавати цінну інформацію для прийняття стратегічних рішень. У цьому підрозділі буде розглянуто застосування AI у контексті аналізу ринкових даних, переваги цього підходу та його реалізація в межах розроблюваної системи.

Застосування AI в контексті аналізу ринкових даних дає можливість опрацьовувати великі обсяги отриманої інформації в найкоротші терміни. Швидкість обробки дозволяє оперативно отримувати результати. Іншою важливою перевагою є можливість розпізнавання прихованих трендів та тенденцій, які можуть бути непомітними для людини.

Як відомо нейронні мережі навчаються на основі історичних даних, тобто інформації зібраної в минулому, яка відображає поведінку, події чи результати. Це дає їм змогу удосконалювати результати аналізу з часом. Останньою властивістю є персоналізація отриманих результатів. Це означає, що впродовж процесу аналізу даних про користувачів, AI може допомагати в адаптації маркетингових кампаній або підборі індивідуальних рекомендацій для клієнтів.

У межах розроблюваної системи для аналізу ринкових даних було обрано інтеграцію API нейронної мережі ChatGPT, яка реалізується за допомогою бібліотеки openai [11]. Прийняте рішення дозволяє розширити функціонал системи та виконувати складні задачі щодо аналізу даних. Використання AI може забезпечити якісну обробку зібраних даних, аналіз отриманої інформації за певними критеріями та автоматичну реалізацію підбору товарів із власної БД, які найбільше відповідають характеристикам конкурентних продуктів, для подальшого відображення у системі.

Наприклад, система отримує дані про популярні товари на конкурентних вебресурсах. Після цього AI аналізує ці товари та допомагає обрати подібні продукти зі списку власних товарів. Зокрема, нейронна мережа може враховувати сезонність, популярність товарів у різних регіонах та інші параметри. Це дозволяє формувати секцію «Популярні товари» або інші тематичні блоки на головній сторінці.

Але, незважаючи на очевидні переваги, використання AI має певні виклики. Якість отриманих даних не завжди може відповідати очікуваним результатам, через безпосередню залежність від точності та якості вхідних даних. Також, використання штучного інтелекту вимагає значних обчислювальних ресурсів, особливо при обробці великих обсягів даних. Загалом для успішного досягнення поставлених цілей необхідно проводити адаптацію моделей під конкретні потреби системи.

Для реалізації головної мети кваліфікаційної роботи необхідно навчитися користуватися інструментами, які пропонує штучний інтелект. Штучний інтелект пропонує низку інструментів і функцій, які підвищують ефективність

аналізу ринкової інформації. Зокрема, до основних інструментів, які можуть бути використані в таких системах, належать `function calling` [12] та `file search` [13].

`Function calling` дозволяє інтегрувати AI в систему з мінімальними витратами часу та ресурсів. Даний підхід забезпечує передачу специфічних запитів до AI у вигляді заздалегідь підготовлених функцій, які виконуються на стороні моделі. Наприклад, можна викликати функції для класифікації товарів, визначення їхньої популярності або формування рекомендацій для клієнтів.

Підхід `function calling` відмінно мінімізує необхідність в окремому програмуванні складних аналітичних алгоритмів; підвищує швидкість обробки завдяки використанню вже готових моделей та забезпечує адаптивність, оскільки функції можуть бути легко оновлені або доповнені новими.

Щодо підходу із використанням інструменту для роботи з файлами під назвою `file search` можна розглянути можливість використання AI для пошуку та аналізу даних у великих наборах файлів, що зберігаються у різних форматах (JSON, CSV, XML тощо). Як приклад, можна розглянути інтеграцію інструменту для пошуку інформації про товари-конкуренти у зібраних даних або для аналізу історичних даних покупців.

Даний інструмент володіє відмінною підтримкою різних форматів даних, що робить його універсальним. Більше того, він є автоматизованим та легко інтегрованим, що робить його надзвичайно привабливим у процесі вибору інструментів для аналізу даних із використанням штучного інтелекту.

Останнім потенційним інструментом для розгляду є `code interpreter` [14], або, як його ще називають – Python-інтерпретатор. Даний інструмент дозволяє виконувати штучному інтелекту складні математичні обчислення, обробляти файли, будувати графіки та аналізувати дані. Перевагою є відсутність необхідності розробляти окремі скрипти. Наприклад, за допомогою цього інструменту можна візуалізувати ринкові тенденції, прогнозувати попит на товари або визначати аномалії у даних.

`Code interpreter` є гнучким у виконанні різноманітних завдань, від базових обчислень до складного аналізу. Можна безпосередньо працювати з даними у

різних форматах, таких як таблиці, графіки або файли. А також, він є доволі простим при інтеграції в інфраструктуру штучного інтелекту.

Протягом процесу інтеграції AI в систему для аналізу ринкових даних було виявлено низку проблем, які ускладнювали отримання стабільних і коректних результатів. Основні труднощі були пов'язані з використанням певних інструментів AI, які потребують точного налаштування вхідних даних і мають обмеження, що впливають на їхню ефективність у конкретних задачах.

Одним із перших інструментів, які були протестовані в межах інтеграції, став `function calling`. Основною особливістю є необхідність передачі чітко описаних функцій і схем вхідних і вихідних даних. Це дозволяє AI виконувати функції на основі переданих запитів. Попри високий потенціал цього інструменту виникли труднощі з підготовкою запитів для складних структурованих даних. Для роботи із запитами необхідно заздалегідь створити схему, яка детально описує очікувані результати, та підготувати функції, які AI використовуватиме для обробки цих даних. Виявилося, що `function calling` чудово підходить для завдань, як-от побудова чат-ботів або персональних помічників у вебмагазинах, але інструмент не зміг забезпечити потрібну гнучкість для аналізу ринкових даних. Таким чином, його використання було визнано недоцільним для аналізу ринкового пошуку.

Наступний інструмент, який теоретично міг би бути застосований, — це `code interpreter`. Цей інструмент призначений для виконання складних математичних обчислень, побудови графіків і аналізу даних. Проте через його специфіку та спрямованість на інші типи задач, наприклад, прогнозування трендів або візуалізацію даних, його використання не було розглянуто для аналізу ринкового пошуку. Такий інструмент краще підходить для дослідницьких завдань або оптимізації бізнес-процесів, а не для задач, пов'язаних із пошуком і класифікацією товарів.

Останнім інструментом на розгляді став інструмент `file search`. Як було зазначено раніше, підхід дозволяє аналізувати великі обсяги даних у форматах JSON, CSV або XML. На відміну від `function calling`, `file search` виявився менш

чутливим до формату вхідних даних і забезпечив більш гнучке налаштування для задач аналізу.

Завдяки цьому інструменту вдалося досягти бажаного результату. Протягом виконання поставленого завдання отриманий шляхом парсингу вебресурсів файл з товарами завантажувався до нейронної мережі (AI), яка успішно виконувала процес обробки та повертала масив ідентифікаторів продуктів, що відповідали критеріям пошуку. Простота налаштування та гнучкість зробили file search оптимальним вибором для виконання задач аналізу ринкових даних.

Дослідження інструментів, які пропонує штучний інтелект, дало змогу здійснити детальний аналіз їхніх можливостей та визначити найоптимальніший варіант для розв'язання поставлених задач. Зведена таблиця нижче (див. табл. 1.2) демонструє порівняльну характеристику ключових інструментів. Критерії вибору враховували специфіку поставлених задач, зокрема швидкість обробки, гнучкість у налаштуванні, сферу застосування та переваги кожного інструменту. На основі аналізу зроблено висновок, що File Search забезпечує найвищу ефективність для інтеграції в систему, завдяки своїй масштабованості та здатності працювати з великими обсягами неструктурованих даних.

Таблиця 1.2.

Порівняння інструментів для інтеграції AI

Критерій	Function Calling	Code Interpreter	File Search
Призначення	Виконання окремих функціональних запитів	Аналіз даних, розрахунки та прототипування	Пошук та обробка неструктурованих даних
Швидкість обробки	Висока	Середня	Висока
Складність інтеграції	Низька	Висока	Низька
Гнучкість у налаштуванні	Обмежена	Висока	Висока
Сфера застосування	Автоматизація рутинних завдань	Складні алгоритмічні завдання	Робота з файлами та аналізом отриманих даних

Продовження таблиці 1.2

Переваги	Простота використання	Потужність для складних розрахунків	Масштабованість адаптація до різних форматів
Обмеження	Менше підходить для роботи з великими обсягами	Не оптимізовано для масивних даних	Вимагає правильної організації даних
Загальна оцінка	7/10	8/10	9/10

У процесі інтеграції штучного інтелекту було ретельно розглянуто можливості інструментів, а також проведено їх попереднє тестування. Кожен із розглянутих інструментів має свої переваги й недоліки. Function calling виявився ефективним у завданнях із чітко визначеною структурою даних, але не зміг забезпечити необхідну гнучкість для аналізу ринкових пошуків. File search, навпаки, показав високу ефективність і став основним інструментом для вирішення поставлених задач. Щодо code interpreter, цей інструмент не використовувався, оскільки його функціонал не відповідає потребам системи.

Отже, інтеграція штучного інтелекту (AI) у систему аналізу ринку дозволяє суттєво підвищити ефективність обробки даних та забезпечити конкурентні переваги для бізнесу. Розглянуті інструменти, такі як function calling, code interpreter та file search, демонструють широкий спектр можливостей для автоматизації та глибокого аналізу.

Кожен із підходів безперечно має свої сильні сторони. Function calling ефективно обробляє окремі запити, забезпечуючи гнучкість інтеграції, а code interpreter дозволяє проводити складні математичні обчислення та швидко прототипувати рішення. Водночас, саме file search показав себе найкращим вибором для розроблюваної системи через здатність працювати з великими масивами неструктурованих даних.

File search не лише спрощує доступ до релевантної інформації, але й забезпечує високу швидкість обробки, що критично важливо для динамічного ринкового середовища. Крім того, даний підхід дозволяє легко адаптувати систему до змін у бізнес-процесах, таких як оновлення даних або перегляд маркетингових стратегій.

Таким чином, аналіз можливостей та обмежень інструментів підтвердив, що file search є оптимальним вибором для вирішення поставлених задач. Його впровадження створює основу для ефективного управління інформацією та підтримки стратегічних рішень у конкурентному середовищі.

Висновки до першого розділу.

У першому розділі кваліфікаційної роботи було проведено комплексний аналіз сучасних підходів направлених на ринковий пошук та розробку системи для оптимізації роботи інтернет-магазинів електротехнічної тематики. Розділ складається з чотирьох підрозділів, у кожному з яких детально розглянуто окремі аспекти проблематики, поставлені задачі та запропоновані шляхи їх вирішення.

У підрозділі, присвяченому постановці задачі, було сформульовано мету розробки системи, що включає забезпечення ефективного збору, аналізу та обробки ринкових даних. В результаті досягнуто концептуальне розуміння шляхів оптимізації асортименту продукції, підвищення конкурентоспроможності підприємства та забезпечення його довгострокового розвитку в умовах динамічного ринкового середовища.

Другий підрозділ, що фокусується на аналізі сучасних методів ринкового пошуку, розкрив актуальні підходи до моніторингу конкурентів, аналізу трендів і використання великих даних. У результаті було запропоновано інтеграцію вебскрейпінгу, який дає можливість збирати інформацію про конкурентів і використання внутрішніх даних для створення адаптивного підбору товарів. Це, насамперед, дозволяє швидко реагувати на ринкові зміни та підвищувати рівень продажів.

У третьому підрозділі розглянуто інструменти для збору, фільтрації та обробки даних, такі як парсинг вебсайтів. Було визначено ключові етапи роботи з даними: збір інформації, її порівняння з внутрішніми показниками, а також вибір оптимальних товарів для виведення на сторінки сайту.

Четвертий підрозділ присвячено використанню штучного інтелекту (AI) для аналізу ринкових даних. В ході дослідження було протестовано кілька підходів, серед яких function calling, code interpreter і file search. Основним

інструментом, який довів свою ефективність для роботи з великими масивами неструктурованих даних, став file search.

Таким чином, у розділі досліджено та обґрунтовано вибір сучасних методів і технологій, необхідних для вирішення поставлених задач. Проведені дослідження дозволяють окреслити основні напрямки для майбутньої реалізації проєкту. Це формує основу для подальшого вдосконалення системи та її адаптації до умов конкурентного середовища, а також дає можливість перейти до наступних етапів роботи, зокрема до практичної розробки та тестування обраних рішень.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ АНАЛІЗУ РИНКОВОГО ПОШУКУ

2.1. Проєктування архітектури системи

Процес проєктування архітектури системи аналізу ринкового пошуку є фундаментальним етапом, який визначає подальшу ефективність її роботи, можливість масштабування та адаптації до змінюваних умов. Основною метою проєктування є створення багатокомпонентної структури, яка зможе забезпечити автоматизований аналіз великих обсягів даних, інтеграцію алгоритмів штучного інтелекту (AI) та зручний доступ до результатів обробки.

Розробка архітектури [15] системи вимагає врахування специфічних потреб та викликів, що стоять перед ринковим аналізом. Насамперед система має включати декілька ключових компонентів. Одним із основних модулів є система збору даних, яка повинна забезпечувати можливість отримання інформації з різних джерел, таких як відкриті API, вебсайти, платформи електронної комерції або внутрішні бази даних. Модуль повинен підтримувати багатоформатність даних, наприклад JSON, XML, CSV, а також забезпечувати масштабованість, тобто здатність ефективно працювати з одночасним збором інформації з різних джерел.

Не менш важливим компонентом є модуль попередньої обробки даних, який відповідає за очищення отриманої інформації, її нормалізацію та підготовку до подальшого аналізу. Цей етап включає видалення дублікатів, заповнення пропущених значень, уніфікацію форматів даних та інші операції, які забезпечують високу якість вхідних даних для аналітичних процесів.

Для реалізації цього модуля зазвичай використовуються спеціалізовані бібліотеки Python [16] або JavaScript, які забезпечують високу швидкість обробки та зручність у роботі з великими обсягами інформації. Для забезпечення збереження отриманих даних система повинна включати модуль для роботи із системами управління базами даних. Залежно від потреб, це можуть бути реляційні бази даних [17], такі як MySQL або PostgreSQL, або нереляційні бази

даних, наприклад, MongoDB [18]. Цей модуль відповідає за безпечне збереження інформації та доступ до неї за запитом (рис 2.1).

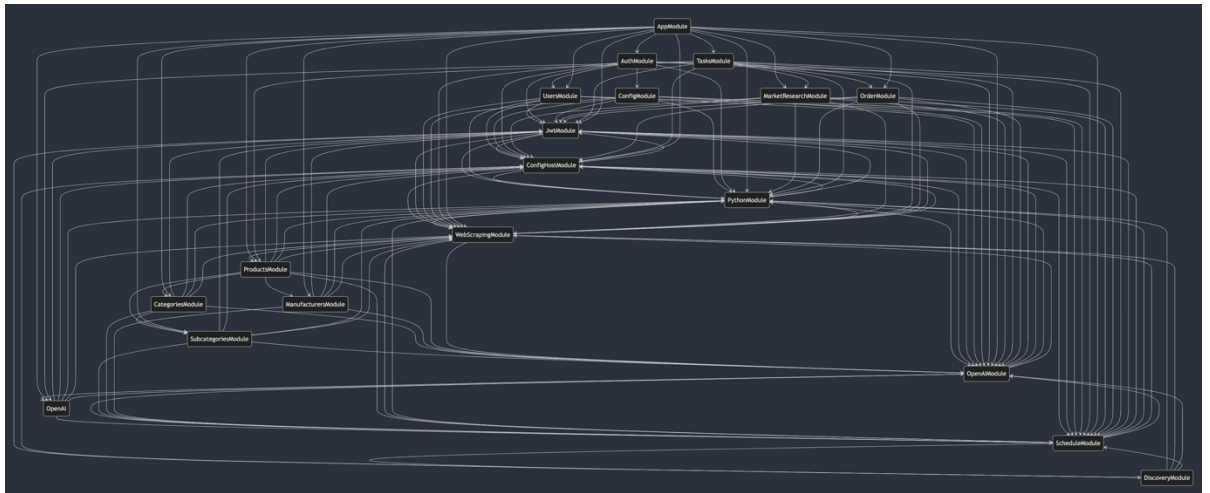


Рис.2.1 Схема зв'язків модулів проєкту

Останнім, але не менш важливим елементом архітектури є модуль інтеграції результатів аналізу. Він дозволяє передавати отримані аналітичні дані до інших систем або компонентів, таких як система управління популярними товарами. Наявність підтримки API інтеграції сприяє зручності використання системи у складі інших бізнес-рішень.

Проектування архітектури системи передбачає проходження кількох етапів. Спочатку необхідно провести аналіз вимог до системи, включаючи визначення цільових користувачів, їхніх потреб та завдань. На основі цього аналізу обґрунтовується вибір підходів до збору, обробки та аналізу даних.

Наступним етапом є безпосередньо проектування архітектури, яке включає визначення ключових компонентів системи, розробку схеми їхньої взаємодії, вибір інструментів та технологій. Після цього реалізуються окремі модулі, які інтегруються в єдину систему, проходять тестування на коректність роботи та оптимізацію продуктивності.

Завершальним етапом є розгортання системи на локальному сервері, її налаштування для роботи з реальними даними та забезпечення підтримки на випадок зміни ринкових умов.

Ключові вимоги до архітектури системи включають:

1. Масштабованість, що дозволяє обробляти великі обсяги даних.

2. Гнучкість для адаптації до змін вимог.
3. Інтеграцію AI для автоматизації процесів.
4. Захист даних для забезпечення безпеки.
5. Ефективність використання ресурсів та простоту у використанні.

Для створення ефективної системи аналізу ринкових даних необхідно врахувати низку ключових вимог, які забезпечують її функціональність, безпеку та здатність адаптуватися до змінних умов ринку.

Процес проєктування архітектури системи аналізу ринкового пошуку передбачає створення ефективної моделі взаємодії між усіма компонентами. Система повинна забезпечувати збір, обробку, фільтрацію та аналіз ринкових даних, а також інтеграцію результатів у кінцевий продукт. Враховуючи складність завдання, було прийнято рішення розділити систему на дві основні частини: серверну та клієнтську. Серверна частина, розроблена на основі фреймворка NestJS [19, 20, 21], виконує всю бізнес-логіку й аналіз ринкових даних, тоді як клієнтська частина на базі Next.js [22, 23] відповідає за відображення результатів і забезпечення зручності користувача.

Архітектура системи базується на принципах модульності [24], масштабованості та інтеграції штучного інтелекту. Модульність забезпечує поділ системи на логічно незалежні частини, кожна з яких виконує конкретну функцію. Масштабованість дозволяє легко додавати нові джерела даних або змінювати алгоритми обробки без потреби в радикальних переробках системи. Інтеграція OpenAI API розширює можливості аналізу, дозволяючи використовувати сучасні алгоритми штучного інтелекту для порівняння товарів конкурентів із власною базою даних.

Проєктування даних системи є важливим етапом розробки програмного забезпечення, оскільки визначаються ключові компоненти, їх взаємодія та структура даних. Основою зберігання даних у проєкті є база даних, яка забезпечує надійне та ефективне управління інформацією.

Для реалізації бази даних обрано MongoDB, яка є документно-орієнтованою NoSQL СУБД [25]. Її перевагами є масштабованість, гнучкість у зберіганні напівструктурованих даних та швидкість роботи. Для взаємодії з

базою даних використовується Prisma ORM [26, 27], що забезпечує зручну абстракцію, дозволяючи працювати з даними через типізовані запити та значно спрощуючи процес взаємодії між серверною частиною програми та базою даних.

Загальна структура бази даних відображає взаємозв'язки між таблицями, що забезпечують ефективну організацію та обробку даних (рис. 2.2).

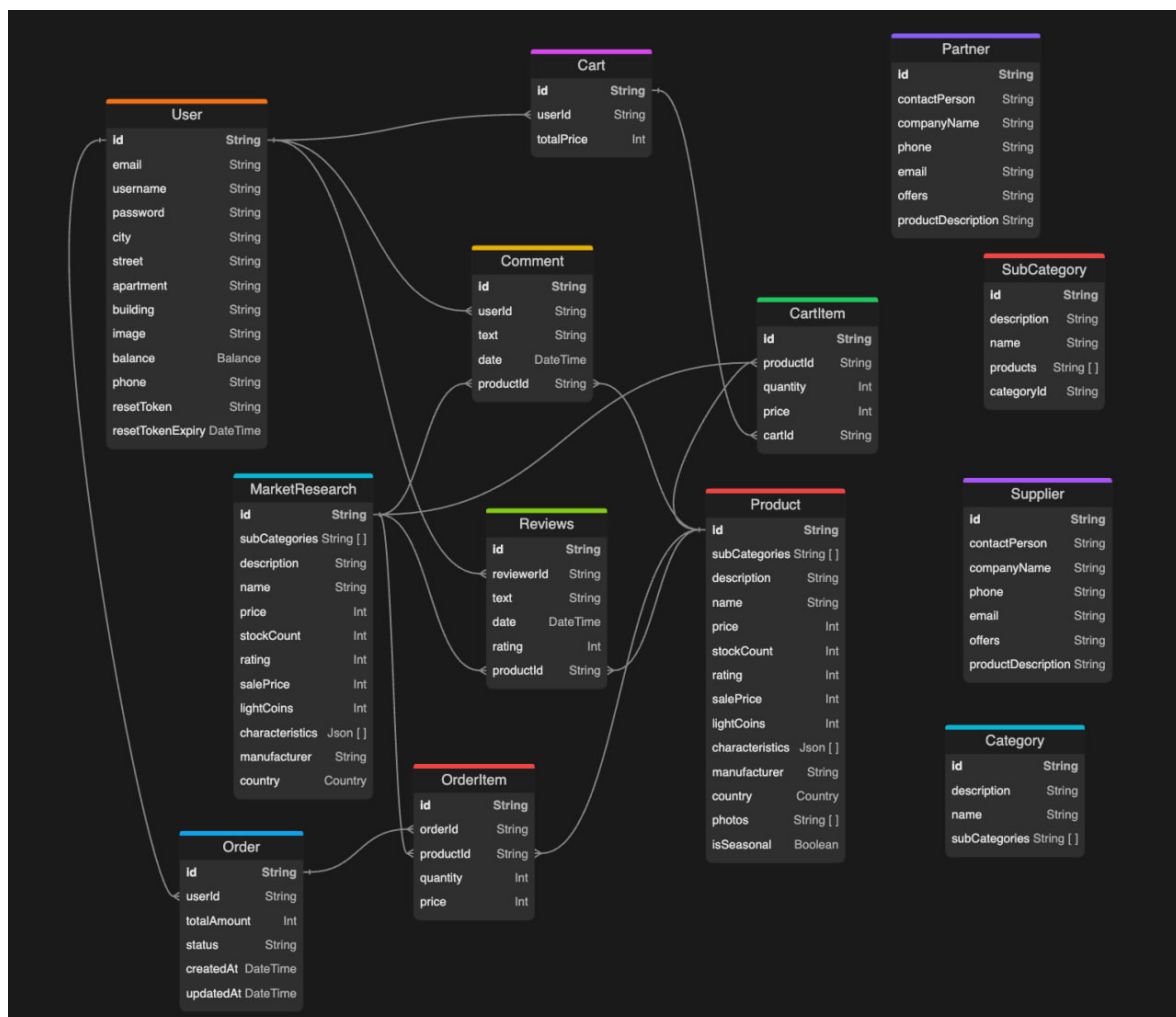


Рис 2.2. Таблиця зв'язків бази даних

Логіка роботи головного модуля системи, а саме серверна частина, виконує ключові етапи аналізу ринкових даних. Її робота починається зі збору даних за допомогою парсингу. Для автоматизації процесу використовується механізм cron-завдань [28], який у визначений проміжок часу запускає функцію парсера. На етапі тестування система збирає дані з сайту Comfy, що дозволяє отримати актуальну інформацію про популярні товари конкурентів. Зібрані дані зберігаються у форматі JSON у локальній файловій системі проекту, забезпечуючи доступність для подальшої обробки.

Після збору даних відбувається їхня обробка. Основна функція фільтрації була реалізована вручну й дозволяє виокремити з масиву отриманих товарів лише ті, що відповідають визначеним критеріям популярності.

Отримані результати є проміжними даними, які ще не інтегровані в систему власних продуктів. Для цього виконується порівняння відфільтрованих товарів із власною БД. Процес автоматизується за допомогою використання бібліотеки OpenAI та її відкритого API, до якого надсилається запит із переліком популярних товарів конкурентів. У відповідь система отримує масив ідентифікаторів продуктів із власної бази, які є схожими на товари конкурентів. Такий підхід дозволяє оптимізувати використання токенів OpenAI, оскільки у запитах передається мінімум інформації. Важливим аспектом є регулярний моніторинг особистого кабінету OpenAI для відстеження та аналізу лімітів і кількості використаних токенів, оскільки їх використання є платним (рис 2.3).

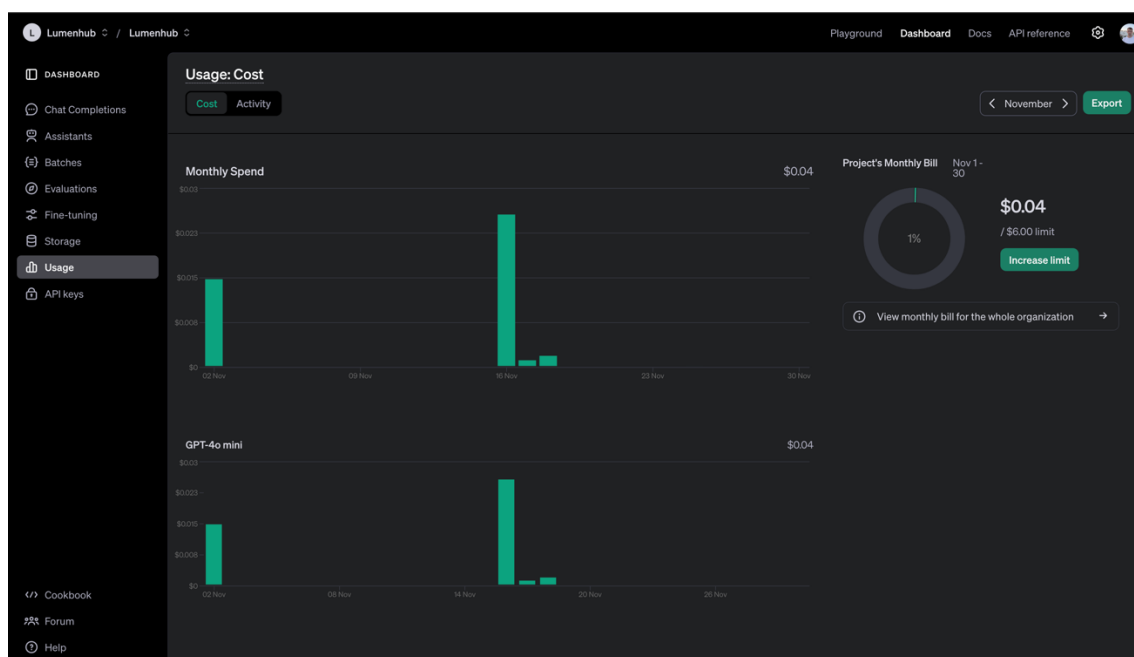


Рис 2.3. Графік використання токенів в особистому кабінеті OpenAI на вкладці Usage

На завершальному етапі результати аналізу записуються в модель Market-Research у БД. Це дозволяє зберігати історичні дані про популярні товари та використовувати їх у майбутньому для аналітики, наприклад, побудови графіків або створення рекомендацій.

Після розгляду загальної архітектури системи, варто звернути увагу на її клієнтську частину, яка відіграє ключову роль у взаємодії з користувачем і забезпечує ефективну комунікацію з серверною частиною. Отож, розроблений вебмагазин із продажу товарів електротехніки реалізує функціональну клієнтську частину на базі Next.js, що забезпечує зручність і ефективність взаємодії користувача з системою. Головна сторінка вебсайту містить кілька ключових секцій для демонстрації результатів аналізу ринку.

Перша секція, «Популярні товари», відображає товари, визначені на основі аналізу ринкових даних, отриманих через серверну частину системи. Користувач може швидко ознайомитися з найбільш актуальними пропозиціями, що підвищує ймовірність їхньої покупки (рис 2.4).

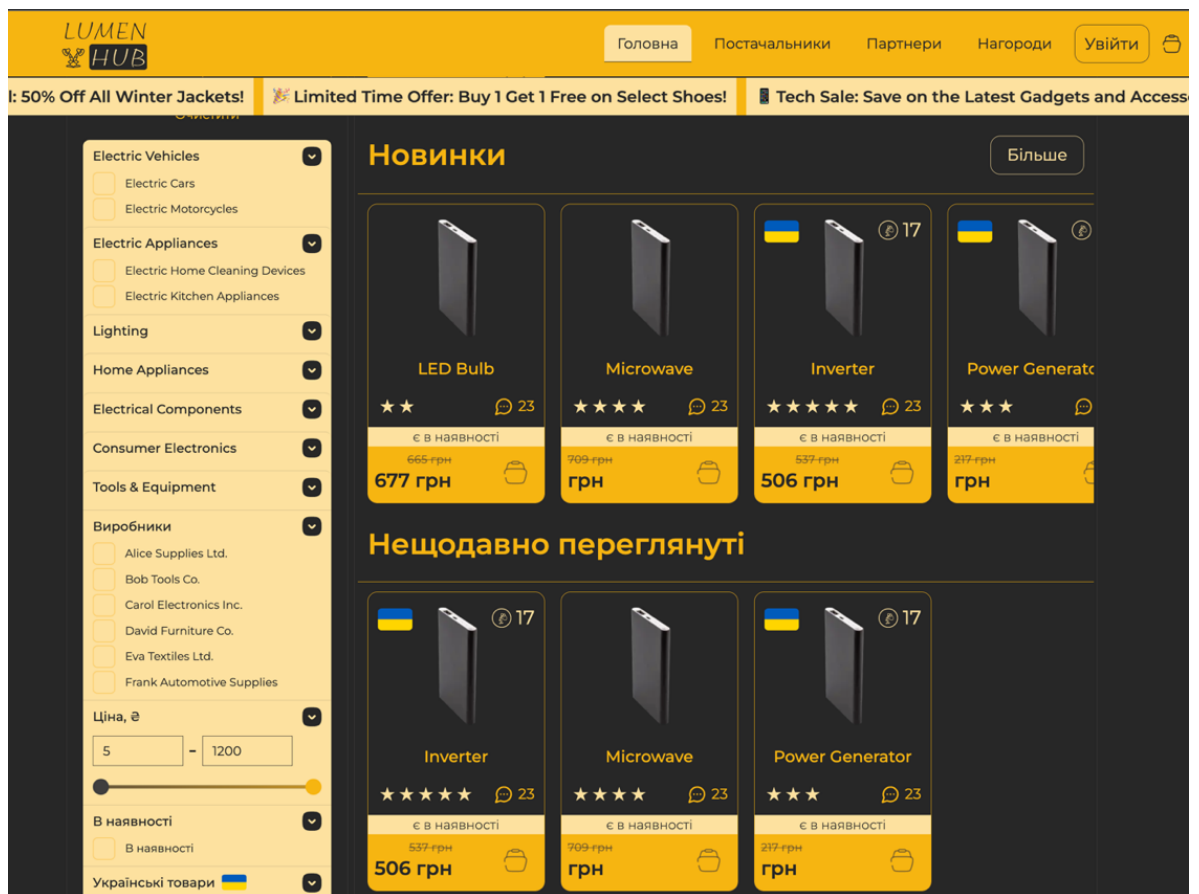


Рис 2.4. Приклад відображення секцій «Новинки» та «Нещодавно переглянуті»

Друга секція, «Сезонні товари», показує продукти, популярність яких зростає у певний сезон, що визначається шляхом аналізу даних покупок за поточний і майбутній місяці.

Окремо варто виділити секцію «Нещодавно переглянуті товари», яка обчислюється виключно на клієнтській стороні за допомогою локального сховища браузера. Коли користувач виявляє зацікавленість конкретним товаром, він додається до локального сховища, а потім відображається в цій секції на різних сторінках сайту. Це сприяє покращенню користувацького досвіду і підвищенню ймовірності остаточної покупки завдяки повторному залученню уваги користувача до переглянутого товару (див. рис 2.4).

Окрім головної сторінки, у магазині є повноцінна сторінка каталогу, яка дозволяє користувачам переглядати весь асортимент товарів із можливістю фільтрації й сортування за різними критеріями. Завдяки використанню переваг серверного рендерингу [29, 30] клієнтська частина забезпечує швидке завантаження сторінок і високу продуктивність навіть за значних обсягів даних. Це не лише покращує досвід користувача, але й робить систему оптимальною для роботи на різних пристроях.

Варто зазначити, що розроблена система аналізу ринкового пошуку функціонує у декілька кроків, які займають певний час. Це кроки, розглянуті раніше, а саме: парсинг даних, фільтрація та обробка за допомогою AI. Саме тому система не проводить аналіз у режимі реального часу для кожного запиту адже такий підхід міг би суттєво перевантажити сервер і негативно вплинути на продуктивність. Замість цього аналіз ринкових даних виконується періодично за допомогою cron-завдань, що запускають процес збору та обробки даних з інтервалами, визначеними заздалегідь. Отримані результати зберігаються у базі даних і залишаються доступними для запитів клієнтської частини протягом певного періоду, наприклад, до виконання наступного циклу аналізу. Це дозволяє користувачам отримувати актуальні дані майже миттєво, без затримок на виконання складних обчислювальних операцій під час кожного запиту. Цей підхід забезпечує високу ефективність і стабільність у роботі системи, дозволяючи обробляти великі обсяги даних без значного навантаження на сервер. Завдяки цьому клієнтська частина може ефективно працювати навіть при значному зростанні кількості користувачів або обсягу оброблюваних даних, що робить систему масштабованою і гнучкою.

Отже, в даному підрозділі були розглянуті основні важливі аспекти правильної побудови системи, такі як масштабованість, гнучкість, ефективність використання ресурсів, інтеграція з AI, захист даних та простота використання. Було показано, як ці принципи можуть бути реалізовані в архітектурі інтернет-магазину, що дозволяє забезпечити високий рівень продуктивності, зручності для користувачів та ефективного використання ресурсів.

Розроблена система повністю відповідає зазначеним критеріям. Вона демонструє високий рівень масштабованості завдяки використанню періодичних cron-завдань для аналізу ринкових даних, що дозволяє обробляти великі обсяги інформації без перевантаження серверів. Гнучкість системи забезпечена можливістю адаптуватися до змін вимог завдяки регулярному оновленню даних про популярні та сезонні товари. Інтеграція з AI для аналізу ринку дозволяє ефективно обробляти та класифікувати товари, підвищуючи точність прогнозів. Захист даних забезпечується надійними методами зберігання та передачі інформації, що гарантує безпеку користувачів. Окрім цього, система має зручний інтерфейсу, що сприяє покращенню загального користувацького досвіду.

Однією з ключових переваг розробленої системи є її модульність, що дозволяє легко замінювати або оновлювати окремі компоненти без змін в основній логіці роботи. Наприклад, замість інтеграції openai від ChatGPT, можна безперешкодно інтегрувати іншу нейронну мережу чи сервіс, і це не вплине на загальний процес роботи системи. Всі модулі системи є незалежними, що забезпечує гнучкість при адаптації до нових технологій або змін в бізнес-вимогах.

Таким чином, система є ефективним, масштабованим і гнучким рішенням, яке відповідає сучасним вимогам до вебмагазинів, зокрема у частині обробки великих обсягів даних, інтеграції AI, безпеки та зручності використання, а також надає можливість легко адаптуватися до змінних потреб ринку. Застосування зазначених принципів та вимог сприяє розробці інноваційного рішення, яке може слугувати інструментом підвищення конкурентоспроможності та ефективності бізнесу.

2.2. Інтеграція AI для автоматизованого аналізу ринкового пошуку

У сучасних умовах автоматизація процесів аналізу ринку є ключовим фактором для прийняття стратегічних рішень у бізнесі. Для реалізації завдань було інтегровано бібліотеку OpenAI, яка надає доступ до потужних інструментів штучного інтелекту через програмний інтерфейс (API). Бібліотека OpenAI надає широкий спектр інструментів для роботи зі штучним інтелектом, таких як генерація тексту, обробка даних, робота з файлами та взаємодія з користувачами через розширені мовні моделі. Основна перевага полягає у здатності швидко аналізувати великі обсяги інформації, генерувати зрозумілі та логічні відповіді, а також інтегруватися в різноманітні програмні рішення. Для інтеграції бібліотеки в проєкті був створений модуль (`openai.module.ts`) і сервіс (`openai.service.ts`), що дозволяють зручно використовувати функції OpenAI в межах архітектури NestJS.

OpenAI пропонує широкий спектр інструментів, які спрощують взаємодію з моделями штучного інтелекту. Серед них Function Calling, що дозволяє інтегрувати функції, які можуть виконуватися на запит моделі для автоматизації задач; Code Interpreter, призначений для виконання складних обчислень, аналізу даних та генерації візуалізацій; і File Search, що забезпечує можливість завантажувати файли для пошуку, аналізу та обробки інформації.

Під час роботи над інтеграцією OpenAI було досліджено можливості функцій інструменту Function Calling, який дозволяє штучному інтелекту викликати заздалегідь підготовлені функції на основі контексту користувацького запиту. Ідея полягає в тому, щоб навчити модель OpenAI аналізувати вхідні дані та викликати функцію для вибору схожих продуктів із власної бази даних. Для цього був написаний тестовий код із передачею AI заздалегідь підготовлених функцій (рис.2.5).

```

185   async findSimilarProductsFC() {
186       return [
187         {
188           name: 'findSimilarProducts',
189           description:
190             'Finds similar products in the database based on given parameters',
191           parameters: {
192             type: 'object',
193             properties: {
194               competitorProduct: {
195                 type: 'object',
196                 properties: {
197                   name: { type: 'string' },
198                   category: { type: 'string' },
199                   price: { type: 'number' },
200                   rating: { type: 'number' },
201                 },
202                 required: ['name', 'category'],
203               },
204             },
205           },
206         ],
207       ];
208     }

```

Рис. 2.5. Сервіс інтеграції інструменту function calling

Після тестування та спроби інтеграції методу в систему використання function calling виявилося недоцільним. Основна проблема полягала в складності реалізації підходу. Функції для пошуку схожих продуктів вимагали розробки складних алгоритмів, здатних ефективно порівнювати десятки параметрів. У той же час, Function Calling вимагає чіткої структури параметрів і правил, що обмежує її гнучкість.

Саме через виявлені проблеми було вирішено змінити підхід та використати інший інструмент під назвою File Search, який дозволяє завантажувати великі файли з даними та використовувати OpenAI для аналізу, пошуку й обчислень.

Після тестування двох інструментів, було вирішено зупинитися на інструменті File Search, оскільки його інтеграція продемонструвала більшу ефективність і відповідність поставленим завданням, а також частіше завершувалася позитивним результатом.

Після вибору інструментів було створено модуль OpenAIModule (openai.module.ts), як глобальний, що дозволяє використовувати сервіси OpenAI в будь-якій частині застосунку (рис. 2.6). У сервісі OpenAiService (openai.service.ts) реалізовано кілька методів для роботи з бібліотекою, а саме (рис. 2.7):

- uploadFile, який завантажує файл для подальшого аналізу;

- createThread, що створює сесію («thread» в перекладі з англ. «потік») для аналізу завантаженого файлу;
- createRun, який запускає процес аналізу в межах сесії.
- getListOfMessages, метод який отримує результати аналізу з сесії.

Модуль OpenAIModule робить сервіс OpenAI доступним у всьому застосунку через механізм глобальних провайдерів, що спрощує взаємодію з API (див. рис. 2.6).

```
src > openai > openai.module.ts > ...
1  import { Global, Module } from '@nestjs/common';
2  import { OpenAiService } from './openai.service';
3  import OpenAIApi from 'openai';
4
5  @Global()
6  @Module({
7    providers: [OpenAiService, OpenAIApi],
8    exports: [OpenAiService],
9    imports: [OpenAIApi],
10 })
11 export class OpenAIModule {}
12
```

Рис. 2.6. Модуль OpenAIModule

```
13 type Message = {
14   text: string;
15   ai?: boolean; // Indicate if the message is from the AI
16 };
17
18 @Injectable()
19 export class OpenAiService {
20   constructor(private readonly openai: OpenAIApi) {
21     this.openai = new OpenAIApi({
22       apiKey: process.env.OPENAI_API_KEY,
23       baseUrl: process.env.OPENAI_BASE_URL,
24       dangerouslyAllowBrowser: true,
25     });
26   }
27
28   async createThread(id: string) {
29     //
30   }
31
32   async createRun(threadId: string) {
33     //
34   }
35
36   async uploadFile() {
37     //
38   }
39
40   async getListOfMessages({ threadId }: { threadId: string }) {
41     //
42   }
43
44   async analyzeMarket() {
45     //
46   }
47 }
```

Рис. 2.7. Сервіс openAiService з головними методами

Кожен із методів сервісу (див. рис. 2.6) є невід’ємною складовою процесу роботи зі штучним інтелектом.

Першим кроком є завантаження JSON-файлу до AI-асистенту для аналізу та отримання бажаного результату. Асистента можна створити декількома способами. Можна скористатися інструментом для тестування API-запитів Postman або Insomnia [31, 32]. Наступним способом є створення асистенту через особистий кабінет OpenAI. Або ж можна створити подібний до описаних вище методів, який буде створювати асистента через програмний код. В даній роботі був обраний перший спосіб, через його простоту та раціональність. Достатньо один раз створити асистента, отримати його унікальний ідентифікатор та зберегти його, для подальшого використання. Отриманий унікальний ідентифікатор було записано у файл змінного оточення проєкту (.env).

Завантажений файл – це JSON файл, що містить набір продуктів, отриманих раніше за допомогою вебскрапінгу із сайтів конкурентів. На цьому етапі викликається метод `uploadFile`, основна задача якого – передати файл для подальшого аналізу та отримати унікальний ідентифікатор завантаженого файлу (рис. 2.8). Метод завантажує файл у сервіс OpenAI та повертає унікальний ідентифікатор (`fileId`). Отриманий ідентифікатор є необхідним для використання у наступних кроках.

```
69   async uploadFile() {  
70     try {  
71       const file = await this.openai.files.create({  
72         purpose: 'assistants',  
73         file: fs.createReadStream('products.json'),  
74       });  
75  
76       return file.id;  
77     } catch (err) {  
78       console.log('Error: ', err);  
79       throw new InternalServerErrorException('Error while trying upload file');  
80     }  
81   }
```

Рис. 2.8. Демонстрація коду `uploadFile`

Після успішного завантаження файлу необхідно створити потік, у якому буде відбуватися аналіз даних. Викликається метод `createThread`, завдання якого ініціювати сесію для роботи з файлом (рис. 2.9). Метод використовує `fileId`,

отриманий з попереднього кроку, для створення потоку. Потік зберігає контекст для аналізу даних.

```
28 async createThread(id: string) {
29   try {
30     const fileId = id ?? process.env.UPLOADED_FILE_ID;
31     const thread = await this.openai.beta.threads.create({
32       messages: [
33         {
34           role: 'user',
35           attachments: [
36             {
37               file_id: fileId,
38               tools: [{ type: 'file_search' }],
39             },
40           ],
41           content:
42             'Analyze my uploaded file and your answer must be only array of strings, nothing more. Go through the competitorProducts array, conduct calculations to assess popularity based on factors like product rate, pricing trends, comments count, category and identify the most popular products from this data. No matter what language is in product name, translate this into your own language to continue do comparisons between competitorProducts and databaseProducts. Return me only an array of strings (product ids) from databaseProducts array that is nearest to analyzed data from competitorProducts.',
43         },
44       ],
45     });
46     return thread.id;
47   } catch (err) {
48     console.log('Error: ', err);
49     throw new InternalServerErrorException(
50       'Error while trying create thread',
51     );
52   }
53 }
```

Рис. 2.9. Демонстрація коду createThread

Після створення потоку необхідно запустити процес аналізу даних у його межах. Це виконується за допомогою методу createRun, який ініціює обробку завантаженого файлу. Метод запускає аналіз у створеному потоці (рис. 2.10).

```
55 async createRun(threadId: string) {
56   try {
57     const assistantId = process.env.ASSISTANT_ID ?? '';
58     const run = await this.openai.beta.threads.runs.create(threadId, {
59       assistant_id: assistantId,
60     });
61     return run.id;
62   } catch (err) {
63     console.log('Error: ', err);
64     throw new InternalServerErrorException('Error while trying create run');
65   }
66 }
67 }
```

Рис. 2.10. Демонстрація коду createRun

Оскільки процес аналізу може займати певний час, необхідно виконати невеличку затримку перед тим, як отримувати результати. Це забезпечує завершення обробки. Після затримки викликається метод getListOfMessages, щоб отримати результати аналізу. Метод отримує список повідомлень із результатами аналізу (рис. 2.11).

```

83   async getListOfMessages({ threadId }: { threadId: string }) {
84     try {
85       const response = await this.openai.beta.threads.messages.list(threadId);
86
87       return response.data;
88     } catch (err) {
89       console.log('Error: ', err);
90       throw new InternalServerErrorException('Error while trying create run');
91     }
92   }

```

Рис. 2.11. Демонстрація коду з отриманням результатів

Отримані дані можуть містити зайву інформацію або мати неструктурований вигляд. Для їх обробки використовуються регулярні вирази, які дозволяють виділити необхідні ідентифікатори продуктів.

Усі вищезазначені кроки виконуються в головному методі `analyzeMarket`, який виконується автоматично за допомогою Cron Job у вказаний період часу (рис. 2.12). Основна задача цієї функції – аналіз конкурентних продуктів і підбір відповідних товарів з власної БД для оновлення ринкових даних.

```

94   async analyzeMarket() {
95     const fileId = await this.uploadFile();
96     const threadId = await this.createThread(fileId);
97     // const threadId = process.env.THREAD_ID ?? '';
98     await this.createRun(threadId);
99
100    const delay = () => {
101      return new Promise((resolve) => {
102        setTimeout(() => {
103          resolve(null);
104        }, 20000);
105      });
106    };
107    await delay();
108
109    const response = await this.getListOfMessages({ threadId });
110
111    const result = JSON.parse(
112      // @ts-ignore
113      response[0].content[0].text.value.replace(/```json|```/g, '').trim(),
114    );
115
116    return result;
117  }

```

Рис. 2.12. Демонстрація коду головного методу `analyzeMarket` та форматування результатів

Фінальним кроком є збереження отриманих ідентифікаторів продуктів у базу даних для подальшого використання.

Отже, вибір на користь File Search як основного інструмента для аналізу ринку був результатом ретельного тестування та оцінки доступних можливостей. Переваги полягають у простоті інтеграції, масштабованості та здатності ефективно обробляти складні запити без потреби в заздалегідь підготовлених функціях. Інструмент під назвою Function Calling, хоча й демонструє значну гнучкість і функціональність, але вимагає тривалого процесу підготовки, створення спеціалізованих функцій і їх налаштування. Це значно б ускладнило використання для поставлених задач. Таким чином, інтеграція File Search у проект забезпечила автоматизацію рутинних процесів, значно спростила аналіз ринкових даних і дозволила підвищити ефективність оновлення та управління базою даних товарів.

2.3. Вибір мови програмування для парсеру та їх ключові відмінності

Головним завданням кваліфікаційної магістерської роботи на тему «Аналіз ринкового пошуку на базі веб-сайту електротехніки» є аналіз ринку та конкурентів, що вимагає доступу до великого обсягу актуальних даних. Ефективний підхід до збору інформації. Це дозволяє автоматично отримувати структуровану інформацію з вебресурсів, що суттєво зменшує ручну роботу та підвищує швидкість обробки даних. Вибір інструментів для реалізації такого процесу, включаючи мови програмування та бібліотеки, відіграє ключову роль у забезпеченні продуктивності, надійності та масштабованості рішення. У цьому розділі буде розглянуто дві популярні мови програмування — Python і JavaScript, які зарекомендували себе як провідні технології для парсингу даних завдяки наявності потужних бібліотек, інструментів і широкої спільноти розробників.

Продуктивність включає в себе швидкість виконання коду, ефективність використання ресурсів та здатність обробляти великі обсяги даних чи складні обчислення. JavaScript спочатку був створений як мова для клієнтської частини вебдодатків, тому його продуктивність оптимізована для швидкого виконання в браузері. За допомогою технології Just-In-Time (JIT) компіляції, JavaScript інтерпретується та компілюється безпосередньо перед виконанням, що дозволяє динамічно оптимізувати код під час його виконання. Це сприяє швидшій реакції

додатків на дії користувачів. Node.js, серверна платформа для JavaScript, використовує однопоточну модель з неблокуючим вводом-виводом, що дозволяє ефективно обробляти велику кількість одночасних з'єднань з мінімальними витратами на ресурси. Це особливо корисно для реальних вебдодатків, де важливо швидко обробляти запити від багатьох користувачів. Python, з іншого боку, є інтерпретованою мовою, що зазвичай призводить до повільнішого виконання коду порівняно з компільованими мовами. Python використовує глобальну блокування інтерпретатора (GIL), що може обмежувати продуктивність багатопотокових програм. Однак, це не завжди є проблемою, оскільки багато завдань в Python можна виконувати за допомогою багатопроцесорної обробки або використовуючи асинхронні бібліотеки.

Для обчислювально-інтенсивних задач, Python часто використовує бібліотеки, написані на C чи C++, що значно підвищує продуктивність. Наприклад, NumPy і pandas [33] використовують високопродуктивні обчислювальні ядра для роботи з великими масивами даних. І JavaScript, і Python підтримують асинхронне програмування, що дозволяє ефективно обробляти ввід-вивід та інші повільні операції. JavaScript, завдяки своїй природі, має вбудовані механізми для асинхронного програмування, такі як колбеки, проміси та конструкція `async/await` [34]. Це робить JavaScript дуже ефективним для вебдодатків, які потребують обробки великої кількості запитів та одночасного виконання кількох задач. Python також підтримує асинхронне програмування за допомогою бібліотек, таких як `asyncio`, що дозволяють писати асинхронний код у зрозумілій та читабельній формі. Асинхронне програмування в Python може значно підвищити продуктивність додатків, особливо у випадках, коли потрібно обробляти багато мережових чи файлових операцій. Однією з основних переваг використання JS для парсингу в NestJS проекті є можливість використання єдиної мови програмування для всього стеку додатку. Це спрощує розробку, оскільки всі частини проекту, включаючи бекенд і парсинг, можуть бути написані на JavaScript або TypeScript. Використовуючи JS для парсингу, можна легко інтегрувати парсинг-сервіси в структуру NestJS, що забезпечує краще управління залежностями, конфігурацією та масштабуванням. NestJS також

підтримує модульну архітектуру, що дозволяє організовувати код у вигляді окремих модулів, які легко тестувати та розширювати. JS, завдяки своїй асинхронній природі (через колбеки, проміси та `async/await`), добре підходить для роботи з мережевими запитами, що є ключовим для парсингу вебсторінок. Асинхронність дозволяє ефективно обробляти численні одночасні запити, не блокуючи основний потік виконання. Це особливо важливо при парсингу великої кількості сторінок, де важливою є продуктивність та швидкість.

Також Node.js і NestJS використовують неблокуючу модель введення/виведення, що робить їх дуже ефективними для обробки великої кількості одночасних мережових запитів. Це може бути особливо корисним для парсингу, де часто виникає необхідність здійснювати багато HTTP-запитів одночасно. Застосування неблокуючої моделі дозволяє значно покращити продуктивність і масштабованість парсинг-додатку.

Також налаштування двох мов програмування на сервері може бути менш вигідним порівняно з використанням однієї мови. Кожна мова має свій менеджер пакетів (наприклад, `npm` для JavaScript і `pip` для Python). Це ускладнює управління залежностями, оскільки потрібно відслідковувати оновлення, конфлікти версій та сумісність пакетів для обох мов. Окрім цього підтримка двох мов вимагає більше зусиль для оновлення середовищ, виправлення помилок і забезпечення безпеки. Пакети, написані на цих мовах, потребують регулярного оновлення, щоб залишатися актуальними та безпечними, що може займати більше часу і ресурсів. Запуск двох середовищ виконання одночасно може збільшити споживання ресурсів сервера, таких як пам'ять та процесор. Це може негативно вплинути на загальну продуктивність серверу та вимагати більш потужного обладнання. У ході виконання роботи спочатку скрипт для парсингу даних із вебсайтів конкурентів був розроблений на мові програмування Python. Проте під час розгортання застосунку на сервері виникли значні проблеми, які виявилися складними та нераціональними для вирішення. Зокрема, запуск двох середовищ виконання одночасно спричиняв конфлікти й ускладнював загальну інфраструктуру. З огляду на це, було прийнято рішення переписати скрипт на

мову програмування JavaScript, що дозволило забезпечити єдність технологій у рамках проєкту.

Таким чином, використання JavaScript для парсингу сайтів у проєкті на NestJS має багато переваг, включаючи єдність мови для всього стеку додатку, легку інтеграцію з NestJS, асинхронність, багату екосистему бібліотек та ефективне управління ресурсами. Це робить JS та NestJS ідеальним вибором для створення високопродуктивних та масштабованих парсинг-рішень.

2.4. Інтеграція результатів аналізу в систему популярних товарів

Інтеграція результатів аналізу ринку в систему популярних, сезонних та інших категорій товарів відіграє важливу роль у створенні ефективного користувацького досвіду в інтернет-магазині електротехніки. Якщо правильно розташувати дані товарні секції на різних етапах взаємодії користувача з платформою, то можна значно підвищити ймовірність придбання товарів, що тягне за собою зростання конверсій і продажів.

Головна сторінка магазину структурована таким чином, щоб створити зручний і привабливий інтерфейс. У верхній частині сторінки розташована навігація, яка дозволяє користувачам швидко переходити до ключових розділів магазину, таких як категорії товарів, корзина, профіль користувача тощо. Центральне місце займає динамічний слайдер, що містить головні акції, знижки та пропозиції. Це дозволяє одразу привернути увагу до найбільш вигідних товарів чи послуг. Під банером розташовані секції з картками товарів. Кожна секція має заголовок, наприклад: «Популярні товари» або «Сезонні пропозиції» або ж «Нещодавно переглянуті товари». Кожна секція також містить кнопку «Більше», яка веде на сторінку каталогу (рис 2.13). У каталозі користувач може переглядати повний асортимент товарів, використовувати фільтри та пошукові інструменти для деталізації вибору.

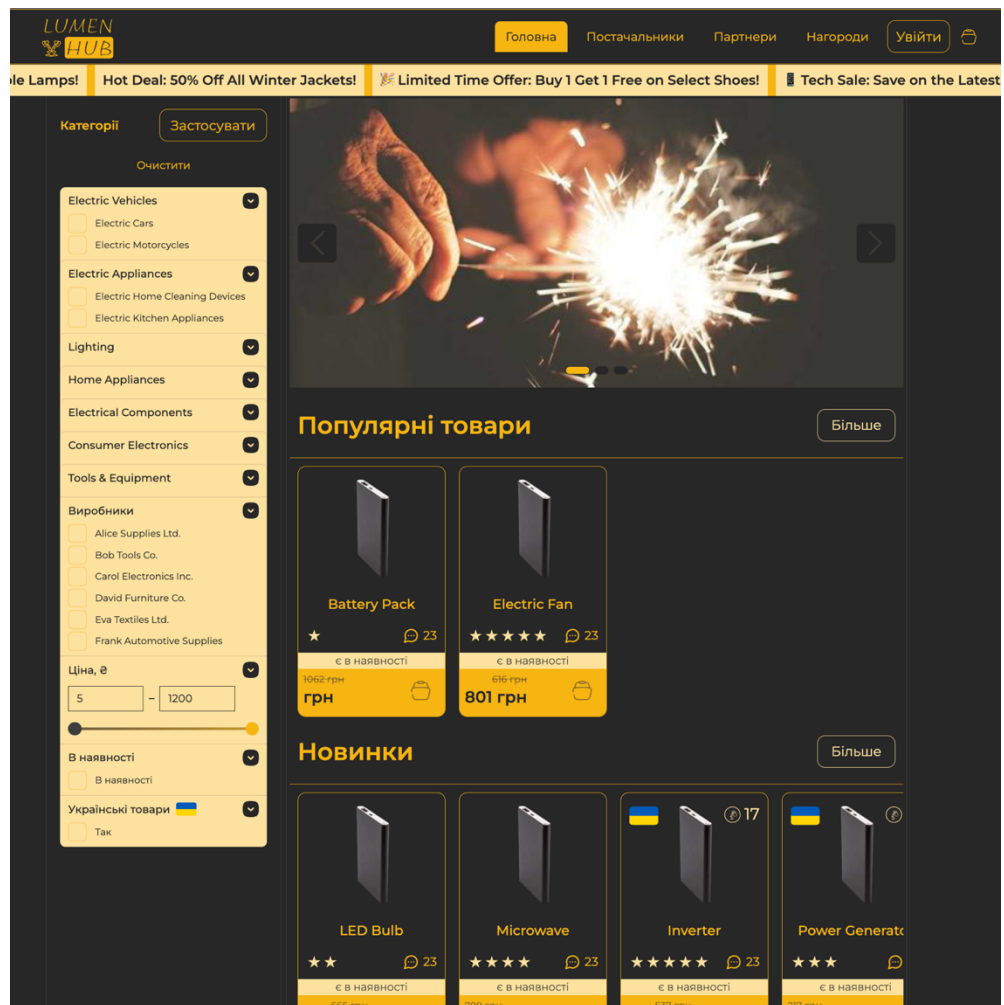


Рис. 2.13. Зображення головної сторінки вебмагазину

Секційний підхід до інтеграції товарів був обраний через його високу адаптивність і зручність. Секції дозволяють розміщувати тематичні списки товарів у багатьох місцях вебмагазину, спонукаючи користувача до покупки. Наприклад, секція «Нещодавно переглянуті» відображається не лише на головній сторінці, але й на сторінці конкретного товару чи під час оформлення замовлення (рис. 2.14). «Популярні товари» можуть з'являтися на головній сторінці, або ж на сторінці оформлення замовлення, пропонуючи клієнту найкращі варіанти з усього асортименту.

Така гнучкість дозволяє забезпечити непомітну, але ефективну присутність товарів протягом усієї взаємодії клієнта з вебсайтом, що значно підвищує ймовірність завершення покупки.

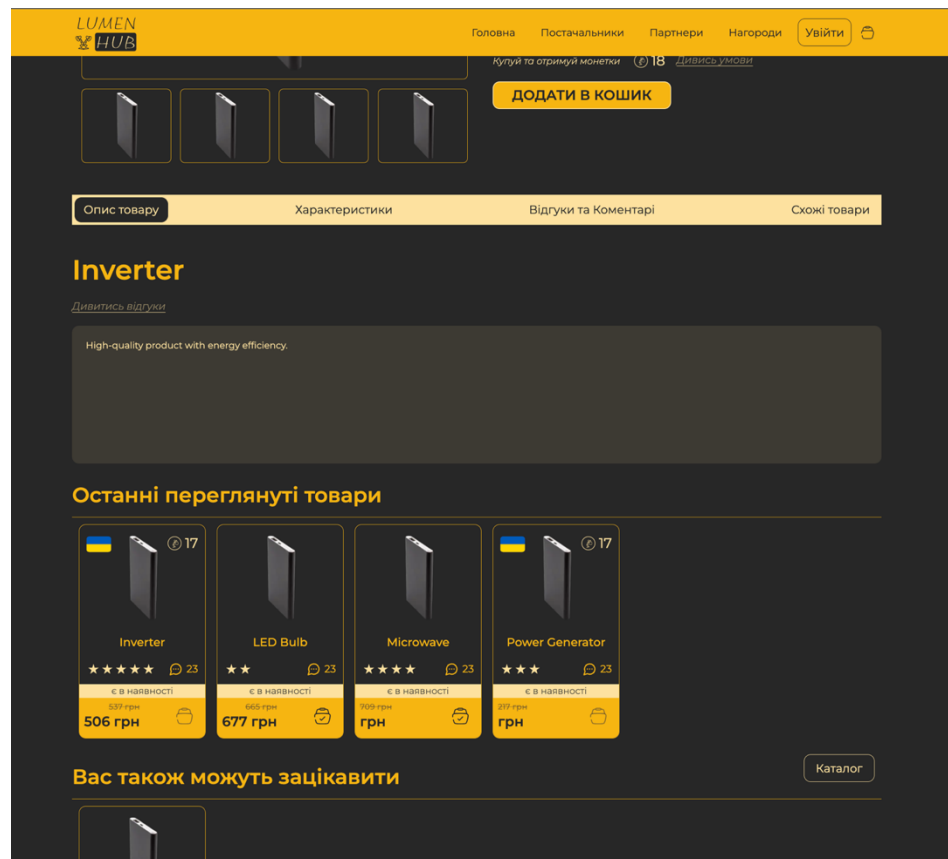
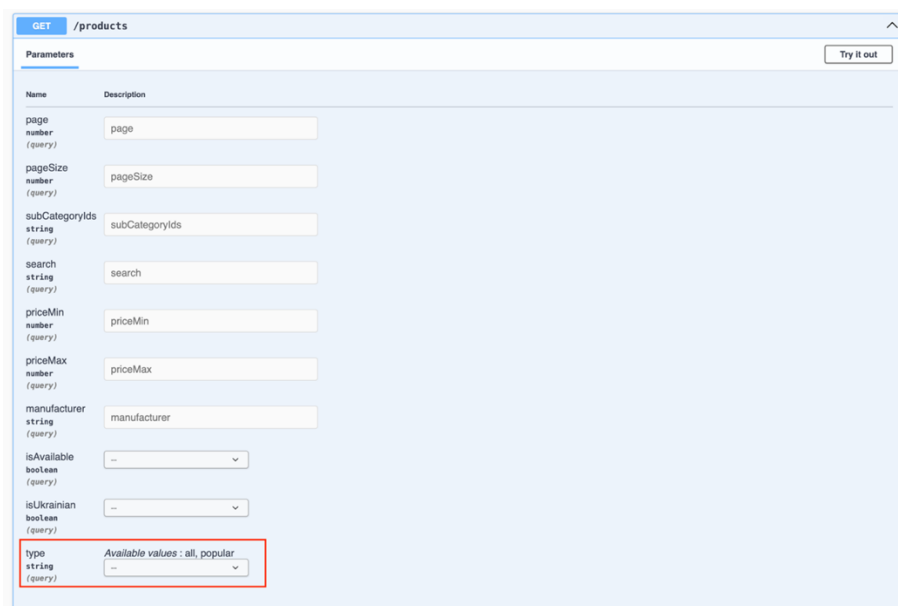


Рис. 2.14. Зображення секцій з пропозиціями на окремій сторінці товару

Процес технічної інтеграції популярних, сезонних та інших категорій товарів у вебмагазині електротехніки базується на синхронізації серверної та клієнтської частин через REST API [35]. Це дозволяє забезпечити динамічне відображення товарних секцій, що адаптуються під потреби користувачів. На серверній стороні результати аналізу ринку та категоризація товарів зберігаються у базі даних MongoDB. Кожна категорія, така як популярні товари, сезонні пропозиції чи нещодавно переглянуті продукти, структурована в окремих колекціях, що дозволяє ефективно управляти даними. Для підтримки актуальності інформації впроваджено механізми автоматичного оновлення, наприклад, при зміні трендів чи додаванні нових товарів ці зміни автоматично враховуються в аналітичних результатах і оновлюються в базі даних.

Для забезпечення доступу до даних створено REST API, який реалізує відповідні ендпоінти, наприклад /products (рис. 2.15) для отримання товарів загалом, або популярних товарів, якщо передати фільтр «type» зі значенням «popular». Кожен ендпоінт підтримує можливість фільтрування, пагінації та сортування, що дозволяє передавати лише необхідні для конкретного запиту

дані. Завдяки цьому клієнтська частина сайту має доступ до релевантної інформації без зайвого навантаження на сервер.



Name	Description
pageNumber (query)	page
pageSize (query)	pageSize
subCategoryIds (query)	subCategoryIds
search (query)	search
priceMin (query)	priceMin
priceMax (query)	priceMax
manufacturer (query)	manufacturer
isAvailable (query)	--
isUkrainian (query)	--
type (query)	Available values : all, popular

Рис. 2.15. Демонстрація документації запиту на отримання товарів

На клієнтській стороні інтеграція з API реалізована за допомогою бібліотек React Query [36] та Axios [37]. React Query виконує роль менеджера стану для асинхронних запитів, забезпечуючи автоматичне кешування, повторне завантаження даних та обробку станів завантаження. Axios, у свою чергу, використовується для налаштування та виконання HTTP-запитів, забезпечуючи зручний синтаксис і розширюваність. Наприклад, функція для отримання популярних товарів виконує запит до ендпоінту /products, повертаючи дані в форматі JSON, які передаються у відповідні React-компоненти.

Щоб полегшити процес розробки та тестування, створено документацію API за допомогою Swagger [38]. Документація надає чіткий опис кожного ендпоінта, включаючи параметри запитів і структуру відповідей (рис. 2.16). Swagger також дозволяє виконувати запити безпосередньо через інтерфейс документації, що значно скорочує час на тестування і інтеграцію нових функцій.

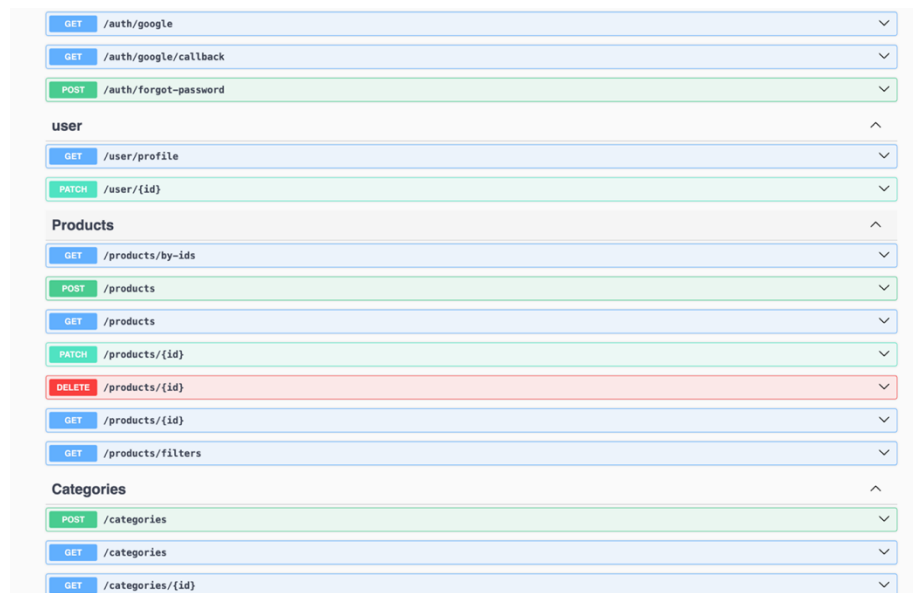


Рис. 2.16. Демонстрація документації Swagger розробленої системи

На етапі інтеграції отримані дані використовуються для динамічного відображення секцій з товарами в інтерфейсі магазину. Наприклад, секції «Популярні товари», «Сезонні пропозиції» та «Нещодавно переглянуті» вбудовані у вигляді карток товарів із заголовком секції та кнопкою «Більше». Реалізація секцій є надзвичайно гнучкою завдяки використанню модульного підходу: компоненти секцій можна легко повторно використовувати у різних частинах магазину, включно з головною сторінкою, сторінкою товару чи навіть сторінкою оформлення замовлення.

Наступним етапом є підготовка взаємодії клієнтської частини із серверною, використовуючи створену документацію (API). Даний процес реалізується за допомогою функцій, що виконують запити до серверу, які використовуються бібліотекою Axios у поєднанні з React Query. Нижче представлено фрагмент коду, що демонструє запит до вхідної точки /products (рис. 2.16).

```

src > api > hooks > product > useProducts.ts > ...
3 import { type ApiErrorResponse } from '@api/types/auth';
4 import ProductService from '@api/services/ProductService';
5 import { type IFilter, type GetProductsResponse } from '@api/types/product';
6
7 function transformQueryParams(filters: Partial<IFilter>) {
8   const params = new URLSearchParams();
9
10  Object.entries(filters).forEach(([key, value]) => {
11    if (value) {
12      params.append(key, value.toString());
13    }
14  });
15
16  return params.toString();
17 }
18
19 type OptionsType = UseQueryOptions<GetProductsResponse, ApiErrorResponse>;
20
21 export default function useProducts(
22   params: Partial<IFilter>,
23   options?: OptionsType,
24 ) {
25   const stringifiedParams = transformQueryParams(params);
26
27   return useQuery({
28     queryKey: queryKeys.getProducts(),
29     queryFn: () => ProductService.getProducts(`/products?${stringifiedParams}`),
30     ...options,
31   });
32 }
33

```

Рис. 2.17. Реакт хук для отримання товарів із використанням React Query

У представленому коді використано ефективний та масштабований підхід до організації роботи з API за допомогою сервісів і React Query (рис. 2.18). Цей підхід сприяє чіткому розподілу відповідальностей між рівнями роботи з даними та забезпечує гнучкість під час внесення змін до логіки взаємодії з сервером.

```

src > api > services > ProductService.ts > ProductService > getProducts
You, 2 weeks ago | 1 author (You)
1 import { axiosClassic } from '@utils/axios';
2 import {
3   type IFiltersResponse,
4   type ICategoriesResponse,
5   type IFilter,
6   type GetProductsResponse,
7   type IProductItem,
8 } from '../types/product';
9
10 You, 2 weeks ago | 1 author (You)
11 export default class ProductService {
12   public static async getProducts(
13     url: string,
14     params?: Partial<IFilter>,
15   ): Promise<GetProductsResponse> {
16     const response = await axiosClassic.get<GetProductsResponse>(url, {
17       params,
18     });
19     return response.data;
20 }
21
22 public static async getCategories() {--
23 }
24
25 public static async getFilters() {--
26 }
27
28 public static async getProductById(id: string): Promise<IProductItem> {--
29 }
30
31 public static async getProductsByIds(ids: string): Promise<IProductItem[]> {--
32 }
33 }
34

```

Рис. 2.18. Сервіс ProductService з методами

Сервіс `ProductService` реалізовано у вигляді класу, який містить методи для взаємодії з API. Усі запити, пов'язані з продуктами, зосереджені в сервісі, що дозволяє забезпечити централізованість та інкапсуляцію логіки. Наприклад, метод `getProducts` виконує запит для отримання списку продуктів, приймаючи як параметри URL і об'єкт із фільтрами. Метод повертає строго типізовані дані за допомогою TypeScript, що гарантує коректність роботи під час розробки. Підхід робить код модульним і зручним у підтримці: якщо потрібно змінити логіку запиту, зміни вносяться лише в одному місці – у відповідному методі сервісу. Окрім того, сервіс забезпечує обробку помилок, наприклад, у методі `getFilters`, де у випадку невдалого запиту повертаються дефолтні значення. Це підвищує стабільність системи, дозволяючи уникнути непередбачених збоїв у роботі інтерфейсу. Інший важливий аспект – використання попередньо налаштованого екземпляра `Axios`, що спрощує конфігурацію HTTP-запитів [39], наприклад, автоматично додаючи базовий URL або токени авторизації.

На клієнтській стороні для роботи з продуктами використовується власноруч написаний хук `useProducts`. Він інтегрує логіку сервісу `ProductService` із можливостями бібліотеки `React Query`, що робить роботу з даними ефективною та зручною. Хук реалізує функцію отримання списку продуктів із переданими параметрами. Перед формуванням запиту об'єкт параметрів обробляється за допомогою функції `transformQueryParams`, яка перетворює його на рядок запиту, забезпечуючи правильне кодування параметрів. Це особливо важливо під час роботи з динамічними фільтрами, оскільки спрощує формування URL для запиту. `React Query`, у свою чергу, додає переваги у вигляді автоматичного кешування даних, повторного використання кешу та обробки станів завантаження. Наприклад, коли дані про продукти вже були отримані, повторний запит до сервера не здійснюється, а використовується кешована версія. Такий підхід дозволяє знижувати навантаження на сервер і підвищувати продуктивність програми. Також `React Query` забезпечує просту обробку помилок і станів завантаження, що значно знижує складність реалізації логіки вручну.

Окремо варто відзначити гнучкість налаштувань хука через параметр `options`, за допомогою якого можна визначати додаткові параметри, такі як час актуальності кешу (`staleTime`), автоматичне оновлення даних під час фокусування вікна (`refetchOnWindowFocus`) тощо. Це дозволяє адаптувати хук до різних сценаріїв використання, залежно від вимог конкретної сторінки або компонента. У підсумку, поєднання сервісу `ProductService` і власного хука `useProducts` створює ефективний і зрозумілий механізм для роботи з даними. Винесення логіки запитів у сервісний шар робить код модульним, полегшує тестування та повторне використання. Використання `React Query` забезпечує автоматизацію багатьох рутинних процесів, таких як кешування, обробка помилок і повторні запити, що значно підвищує якість взаємодії користувача з додатком. Цей підхід є ідеальним вибором для масштабованих додатків, у яких потрібно ефективно працювати з великими обсягами даних і різноманітними параметрами запитів.

Отже, в рамках інтеграції результатів аналізу в систему популярних товарів було проведено всебічне дослідження та розроблено ефективний підхід для адаптації отриманих даних у вебмагазині. На основі аналізу поведінки користувачів, сезонних трендів та ринкових показників були створені динамічні списки популярних і сезонних товарів, які відповідають актуальним потребам клієнтів. З технічної точки зору досягнуто високого рівня структурованості та оптимізації роботи з API за допомогою використання сервісного шару і бібліотеки `React Query`. Як приклад сервісного шару був розглянутий та описаний сервіс для роботи з товарами – `ProductService`. Використання бібліотеки `React Query` дозволило забезпечити: автоматизацію управління станами завантаження, помилок та оновлення даних; ефективне кешування та мінімізацію кількості серверних запитів; зручну масштабованість для інтеграції нових функціональних можливостей у майбутньому.

Реалізований підхід сприяє покращенню користувацького досвіду, забезпечує персоналізованість відображення товарів та підвищує залученість споживачів до взаємодії з платформою. Секційний підхід до відображення популярних і сезонних товарів дозволив інтегрувати ці елементи в різні частини

інтерфейсу, що створює додаткові точки взаємодії з користувачами та стимулює збільшення продажів.

Висновки до другого розділу

В результаті проведеного дослідження та аналізу, а також розробки окремих компонентів системи, було успішно реалізовано архітектуру системи аналізу ринкового пошуку. Реалізована архітектура відповідає вимогам масштабованості, гнучкості та ефективності. Проєктування архітектури системи було спрямоване на забезпечення її здатності адаптуватися до змінюваних умов ринку та швидко інтегрувати нові функціональні можливості без значних затрат на переписування або переробку існуючих компонентів.

Інтеграція технологій штучного інтелекту, зокрема використання OpenAI для автоматизованого аналізу ринкового пошуку, дозволила значно покращити процес обробки інтеграції даних в систему. Вибір OpenAI як інструмента для автоматизації аналізу ринкових трендів дозволив забезпечити високу точність і швидкість обробки інформації, а також надати систему, яка може постійно навчатися і адаптуватися до нових умов ринку, що є критично важливим для сучасних онлайн-платформ.

Паралельно з цим було проведено детальне дослідження різних мов програмування для парсеру даних. Порівнявши такі мови, як Python та JavaScript, було визначено, що JavaScript є найбільш оптимальним вибором для реалізації парсеру у рамках поточної системи, оскільки він інтегрується з рештою компонентів вебдодатка, що значно спрощує процес розробки та підтримки проєкту.

У контексті інтеграції результатів аналізу в систему популярних товарів було розроблено ефективний механізм відображення популярних і сезонних товарів на головній сторінці вебмагазину. Завдяки правильній організації секцій, кожна з яких відображає актуальні товари, зручно і інтуїтивно користувач може ознайомитись з актуальними пропозиціями. Це дозволяє підвищити зацікавленість користувачів і сприяє збільшенню продажів. Вибір розташування цих секцій на головній сторінці та їх поява в інших частинах інтерфейсу

забезпечує постійну взаємодію з користувачем на кожному етапі його шляху по сайту, збільшуючи ймовірність прийняття рішення про покупку.

У підсумку, було досягнуто поставлених цілей щодо створення масштабованої та ефективної системи аналізу ринкового пошуку. Розроблена система інтегрує автоматизовані інструменти на базі штучного інтелекту, а також забезпечує зручну та привабливу для користувачів реалізацію результатів у вигляді секційних блоків. Рішення щодо вибору мови програмування, а також організація структури відображення товарів на сайті стали ключовими елементами, що сприяють успіху проекту в цілому.

РОЗДІЛ 3. ДОСЛІДЖЕННЯ СИСТЕМИ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1. Перевірка працездатності системи аналізу ринкового пошуку

Перевірка працездатності розробленої системи аналізу ринкового пошуку є критично важливим етапом у процесі її створення. Вона дозволяє оцінити якість реалізації функціональних модулів, відповідність заявленим вимогам, стабільність роботи за різних умов експлуатації, а також виявити потенційні недоліки, що можуть вплинути на точність аналізу. Враховуючи комплексність системи, перевірка включала кілька етапів, кожен із яких був спрямований на тестування окремих модулів: парсеру даних, модуля обробки інформації за допомогою штучного інтелекту, а також швидкості та ефективності роботи інтерфейсу популярних продуктів [40]. У цьому підрозділі детально описано результати тестування, які підтверджують коректну роботу системи. Також, розглянуто реалізацію етапів та можливість подальшого вдосконалення системи.

На першому етапі тестування була здійснена перевірка функціональності парсеру даних. Його основне завдання полягає в зборі інформації з вебсайтів конкурентів. Для тестування було обрано сайт Comfy, а саме категорію зарядних станцій («*chargin-stations*»). Основна мета полягала в тому, щоб оцінити, чи здатний парсер виконувати такі завдання, а саме:

1. Коректно відкривати зазначену вебсторінку.
2. Збирати дані про продукти у заданій категорії.
3. Виводити отриманий масив даних у консоль для візуальної перевірки.

Результати тестування підтвердили працездатність парсеру. Після запуску модуля на короткий проміжок часу відкривалася вебсторінка, що демонструє процес ініціалізації збирання даних (рис. 3.1.). Отримані дані були успішно виведені в консоль у вигляді масиву JSON, що свідчить про успішну інтеграцію парсеру з системою (рис. 3.2.).

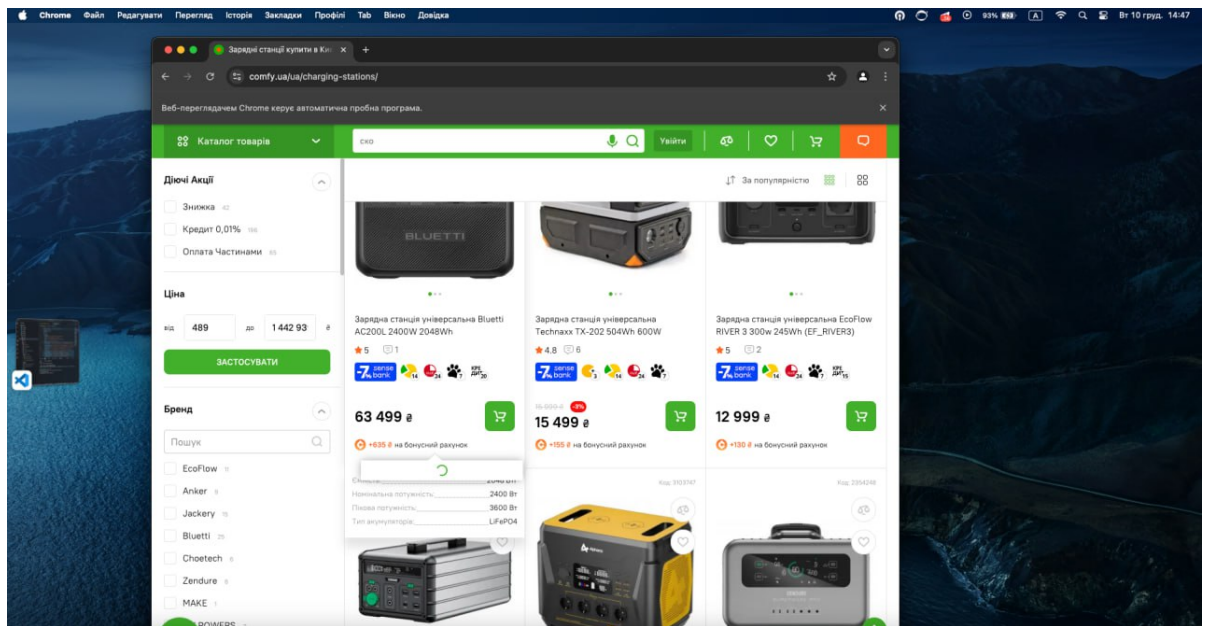


Рис. 3.1. Відображення відкритої сторінки категорії «charging-stations» на сайті Comfy

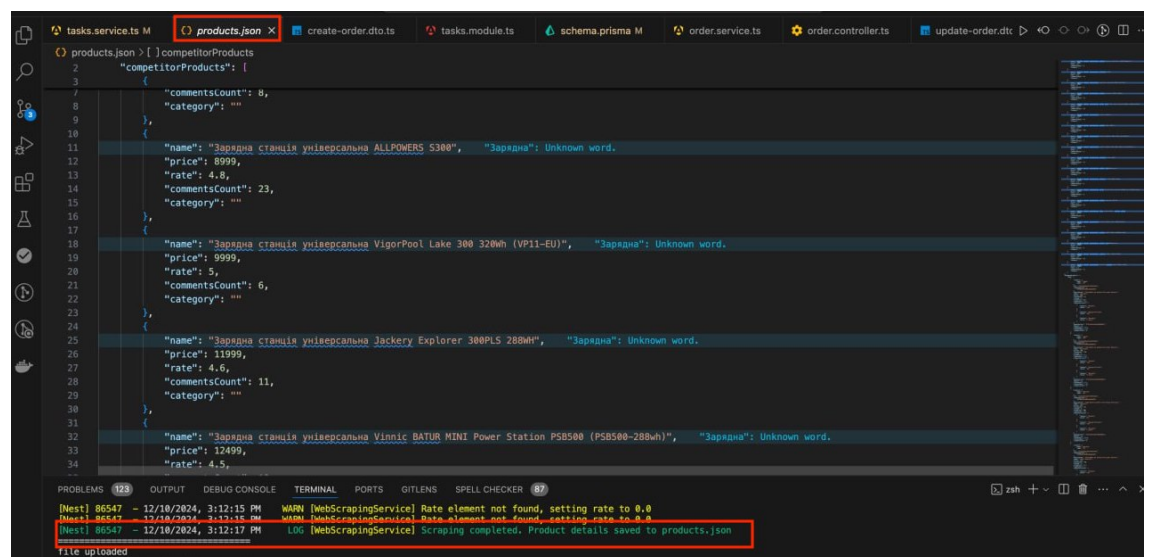


Рис. 3.2. Результати успішного парсингу даних з сайту Comfy та успішний запис в файл

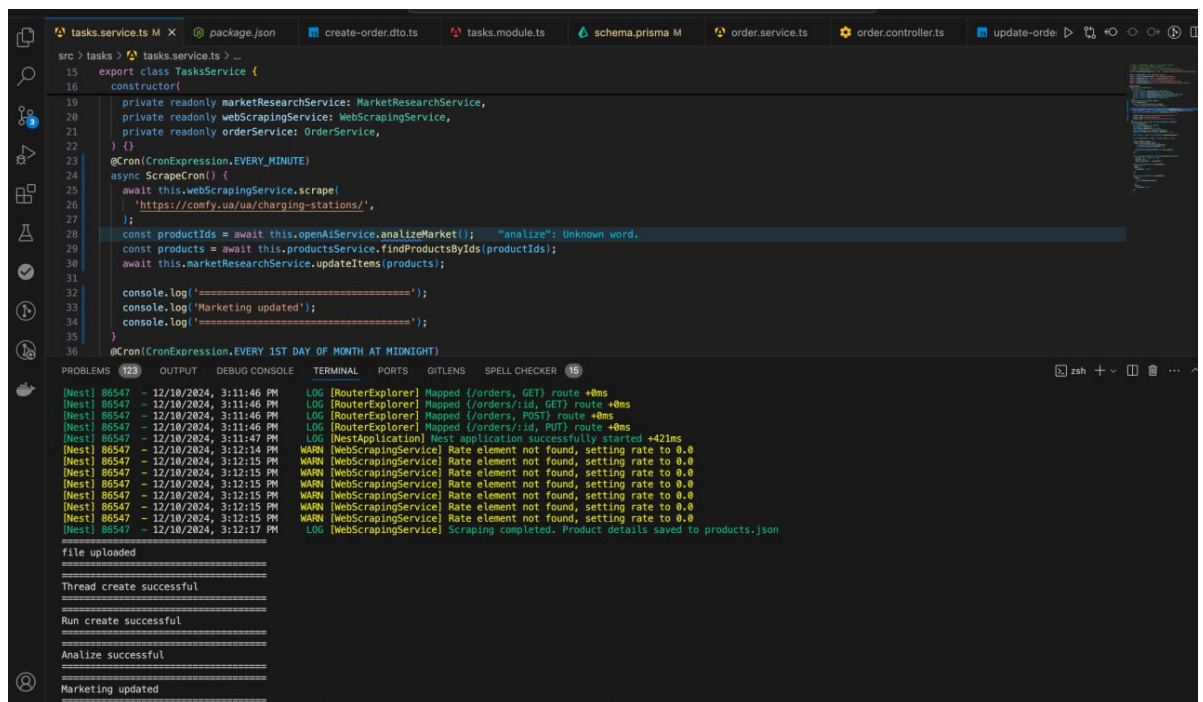
Попри успішну роботу, тестування виявило потенційну необхідність оптимізації швидкості роботи парсеру при обробці великих обсягів даних, а також інтеграції мультипарсингу [41] для охоплення всіх категорій на вебсайті.

Другий етап перевірки був спрямований на оцінку модуля обробки даних за допомогою OpenAI. Даний модуль відіграє ключову роль у системі, оскільки забезпечує автоматизований аналіз зібраної інформації. Тестування включало кілька ключових етапів, а саме: завантаження вхідних даних асистенту,

створення потоку (середовища), його запуск та отримання результатів. Варто зазначити, що отримані товари за допомогою вебскрапінгу успішно записувались у файл, який згодом передавався на аналіз AI-асистенту (рис. 3.3).

Для кожного окремого сеансу аналізу створювався потік, який забезпечував ізоляцію і послідовну обробку даних (рис. 3.4.). Дані успішно передавалися на обробку, після чого виконувалася затримка в 20 секунд для очікування результатів. Система успішно завершувала кожен сеанс, що свідчить про стабільність роботи модуля (див. рис. 3.3.).

Наступним важливим етапом було отримання та форматування результатів роботи OpenAI модулю. Відповіді оброблялися та зберігалися в моделі даних MarketResearch. Така реалізація дозволяє системі накопичувати результати аналізів для подальшого використання (див. рис. 3.3.).



```
src > tasks > tasks.service.ts > ...
15 export class TasksService {
16   constructor() {
17     private readonly marketResearchService: MarketResearchService,
18     private readonly webScrapingService: WebScrapingService,
19     private readonly orderService: OrderService,
20   }
21 }
22 @Cron(CronExpression.EVERY_MINUTE)
23 async scrapeCron() {
24   await this.webScrapingService.scrape(
25     'https://confy.ua/ua/charging-stations/',
26   );
27 }
28 const productIds = await this.openAIService.analyzeMarket(); // "analyze": Unknown word.
29 const products = await this.productsService.findProductsByIds(productIds);
30 await this.marketResearchService.updateItems(products);
31
32 console.log('Marketing updated');
33 console.log('Marketing updated');
34 console.log('Marketing updated');
35 }
36 @Cron(CronExpression.EVERY_1ST_DAY_OF_MONTH_AT_MIDNIGHT)
```

```
[Nest] 86547 - 12/10/2024, 3:11:46 PM LOG [RouterExplorer] Mapped (/orders, GET) route +0ms
[Nest] 86547 - 12/10/2024, 3:11:46 PM LOG [RouterExplorer] Mapped (/orders/:id, GET) route +0ms
[Nest] 86547 - 12/10/2024, 3:11:46 PM LOG [RouterExplorer] Mapped (/orders/:id, PUT) route +0ms
[Nest] 86547 - 12/10/2024, 3:11:47 PM LOG [NestApplication] Nest application successfully started +421ms
[Nest] 86547 - 12/10/2024, 3:12:14 PM WARN [WebScrapingService] Rate element not found, setting rate to 0.0
[Nest] 86547 - 12/10/2024, 3:12:15 PM WARN [WebScrapingService] Rate element not found, setting rate to 0.0
[Nest] 86547 - 12/10/2024, 3:12:15 PM WARN [WebScrapingService] Rate element not found, setting rate to 0.0
[Nest] 86547 - 12/10/2024, 3:12:15 PM WARN [WebScrapingService] Rate element not found, setting rate to 0.0
[Nest] 86547 - 12/10/2024, 3:12:15 PM WARN [WebScrapingService] Rate element not found, setting rate to 0.0
[Nest] 86547 - 12/10/2024, 3:12:15 PM WARN [WebScrapingService] Rate element not found, setting rate to 0.0
[Nest] 86547 - 12/10/2024, 3:12:17 PM LOG [WebScrapingService] Scraping completed. Product details saved to products.json

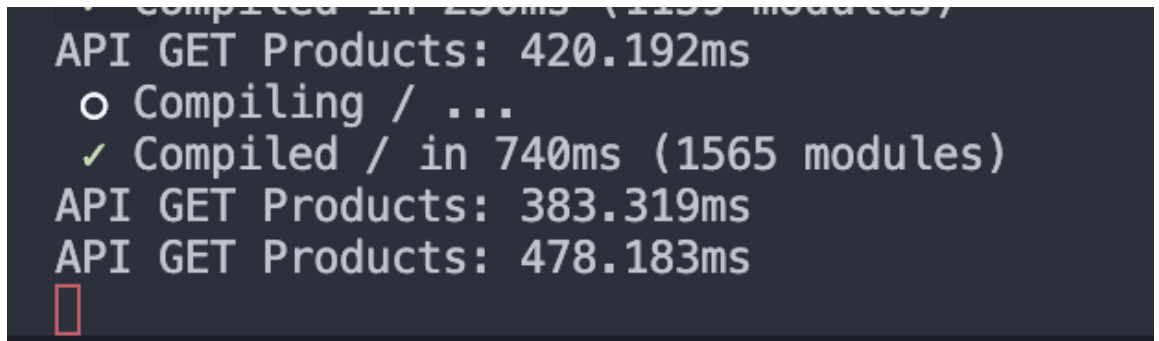
file uploaded
=====
Thread create successful
=====
Run create successful
=====
Analyze successful
=====
Marketing updated
=====
```

Рис. 3.3. Покрокове виконання методів для роботи з AI

Результати етапу підтвердили високу ефективність інтеграції OpenAI, проте було виявлено, що час обробки запиту можна зменшити шляхом оптимізації механізмів передачі даних.

Фінальним етапом тестування стала оцінка швидкості роботи запиту популярних продуктів через REST API (рис. 3.4). Для цього було використано простий метод логування часу виконання за допомогою console.time() та

console.timeEnd()). Тестування виконувалося для ендпоінту /products?type=popular.



```
API GET Products: 420.192ms
o Compiling / ...
✓ Compiled / in 740ms (1565 modules)
API GET Products: 383.319ms
API GET Products: 478.183ms
```

Рис. 3.4. Час виконання запиту за популярними продуктами

Результати показали, що середній час виконання запиту становив **427.231 мілісекунд**, що є прийнятним для забезпечення зручності користувачів у реальних умовах. Проте подальша оптимізація може бути спрямована на зменшення затримки під час обробки великих масивів даних.

Результати перевірки працездатності системи підтверджують її готовність до практичного використання. Усі модулі, а саме: парсер, інтеграція з OpenAI та REST API – продемонстрували стабільну роботу. Водночас тестування дозволило виявити кілька потенційних напрямів для вдосконалення, зокрема оптимізацію швидкості роботи та реалізацію мультипарсингу. Успішне тестування є важливим кроком до реалізації повноцінної системи аналізу ринкового пошуку, що відповідає сучасним бізнес-вимогам.

3.2. Оцінка ефективності застосованих методів і алгоритмів

Ефективність застосованих методів і алгоритмів є ключовим критерієм оцінювання роботи системи аналізу ринкового пошуку, оскільки саме вони визначають якість, швидкість та точність виконання її функцій. У цьому підрозділі проаналізовано результати роботи основних компонентів системи, включаючи модуль збору даних, алгоритми обробки інформації та підсистему інтеграції штучного інтелекту, з метою оцінки їхньої відповідності заявленим вимогам і визначення потенціалу для подальшого вдосконалення.

Парсер, який був розроблений для збору даних із зовнішніх вебресурсів, продемонстрував високу функціональність під час тестування. Він успішно виконував завдання зі збору інформації з вебсайту Comfy, забезпечуючи точність отриманих даних. Критерієм ефективності даного модуля можна виділити точність та швидкість отримання результатів.

Парсер коректно збирав інформацію з визначених категорій без втрати або спотворення даних. Збір даних відбувався в межах кількох секунд, що забезпечує оперативність роботи системи. Попри це, обмеження парсеру в охопленні лише попередньо визначених категорій свідчить про необхідність впровадження мультипарсингу, що значно підвищить його універсальність. Іншою важливою проблемою була помічена зміна структури цільового вебресурсу, а саме сайту Comfy. Впродовж дослідження виявилось, що сайт компанії Comfy змінює назви класів блоків з товарами, що значно ускладнює автоматизацію та стабільність роботи системи. Внаслідок цього необхідно регулярно тестувати систему та у разі зміни структури підлаштовуватись під неї.

Як вже було зазначено в ході роботи, обробка зібраної інформації виконувалася із залученням моделей штучного інтелекту, що дозволяє системі проводити комплексний аналіз даних і генерувати інформативні звіти. Протягом розробки системи була проведена робота щодо оцінки ефективності алгоритмів обробки даних. Насамперед, модуль продемонстрував здатність обробляти великі масиви даних, видаючи точні результати, зокрема рекомендації щодо вибору продуктів або прогнозування ринкових тенденцій. Використання штучного інтелекту через інтеграцію відкритого API (OpenAI) забезпечило можливість адаптації алгоритмів для різних сценаріїв аналізу. В решті решт, навіть за умови передачі значних обсягів інформації система залишалася стабільною, хоча час обробки збільшувався.

Недоліком системи є залежність алгоритмів від часу очікування. При кожному запуску OpenAI модуля проводиться штучна затримка необхідна для успішності отриманих результатів.

Алгоритми, що забезпечують інтеграцію між модулями, виконують важливу роль у координації роботи системи. Зокрема, REST API, відповідальний

за передачу даних між компонентами, демонструє надійність, швидкодію та ефективність роботи із великими обсягами даних. Під час тестування виявлено, що більшість запитів виконуються успішно, без втрати даних, що є гарним результатом. Середній час обробки запиту популярних продуктів становив **427.231 мілісекунд**, що є прийнятним результатом. Незважаючи на наявність великих масивів інформації у БД, фронтенд отримує лише необхідну порцію даних, що усуває будь-які затримки у відображенні результатів.

Даний підхід забезпечує зручність для користувача і оптимізує навантаження на серверні ресурси. Однак для ще більшої продуктивності можна розглянути подальшу оптимізацію запитів до бази даних, наприклад, за допомогою кешування найбільш популярних категорій.

1. Отож, проведений аналіз підтвердив, що застосовані методи і алгоритми відповідають поставленим вимогам щодо збору, обробки та представлення даних, а система успішно виконує свої функції, забезпечуючи точність і надійність результатів. Проте існує потенціал для вдосконалення, наприклад, можна реалізувати мультипарсинг для збільшення охоплення джерел, розробити додаткові модулі візуалізації, які дозволять ефективніше аналізувати результати або ж оптимізувати алгоритми обробки даних для зменшення затримок.

Таким чином, ефективність застосованих методів і алгоритмів підтверджується їхньою функціональністю в реальних умовах експлуатації. Подальше вдосконалення системи дозволить досягти ще вищої продуктивності та задовольнити розширені потреби бізнесу.

3.3. Виявлені обмеження системи та перспективи її вдосконалення

У процесі дослідження та розробки системи аналізу ринкового пошуку було виявлено низку обмежень, які впливають на її функціональність та потенціал. Попри досягнуті результати, існують напрямки, у яких система може бути вдосконалена для забезпечення більш високої ефективності, гнучкості та зручності використання.

Можливим вдосконаленням системи є оптимізація алгоритму щодо збільшення кількості отриманих товарів за короткий час. Це дасть змогу ефективніше аналізувати ринок та значно покращить результати. Також, можна покращити функцію фільтрації отриманих товарів, додавши більшу кількість параметрів для виявлення популярних товарів з конкурентних вебресурсів.

Ще одним напрямом удосконалення може бути впровадження внутрішнього аналізу, який передбачає періодичний перегляд товарів з власної БД, що обиралися як популярні. Це відкриє можливість їх просування в майбутньому або можна буде генерувати нові підбірки товарів, наприклад «Ексклюзив місяця» або «Хіт сезону». Такий підхід сприятиме підвищенню інтересів клієнтів і формуванню актуальних пропозицій.

До того ж, можна покращити розроблену систему шляхом пошуку сторонніх API, які дають можливість проводити аналіз ринкового пошуку, подальше їх підключення та інтеграцію.

Перспективним напрямком розвитку є впровадження мультипарсингу, тобто можливості автоматичного збору даних із великої кількості категорій та підкатегорій на вебсайтах конкурентів. У поточній версії системи реалізовано парсинг лише окремих, заздалегідь підготовлених категорій, що обмежує глибину і масштаб аналізу. Доповненням до цього може бути автоматичне підлаштовування системи парсингу під зміну структури цільового вебресурсу. Наприклад, можна запам'ятовувати назви останніх класів цільових блоків та у разі їх зміни, автоматично переназивати їх. Таким чином можна стабілізувати модуль парсингу даних системи.

Розширення можливостей системи для обробки всіх категорій конкурентних сайтів дозволить отримувати повнішу інформацію про ринок, аналізувати не лише популярні, але й менш очевидні тенденції, виявляти нові ніші та конкурентні переваги.

Запровадження мультипарсингу вимагатиме розробки ефективних алгоритмів для роботи з різними структурами вебсайтів, включно з обробкою динамічного контенту, який генерується за допомогою мови програмування JavaScript. Крім того, знадобиться інтеграція додаткових механізмів для

уникнення блокувань з боку серверів конкурентів, зокрема використання проксі-серверів чи ротації IP-адрес.

Окремо слід зазначити, що масштабування системи вимагатиме оптимізації її швидкодії. Запропоновані удасконалення, а також мультипарсинг, збільшать обсяг даних, що обробляються системою, що безперечно створить виклик у забезпеченні стабільності, швидкості та продуктивності. У цьому контексті може бути доцільним запровадження додаткових механізмів кешування, а також використання хмарних обчислень для обробки великих обсягів даних.

Одні з ключових аспектів – це використання штучного інтелекту для прогнозування. За допомогою обробки історичних даних та алгоритмів машинного навчання можна створити модулі для передбачення майбутніх тенденцій у ринку, попиту на товари та змін у цінах та конкурентоспроможності. Ці прогнози можуть бути корисними для підприємства не лише щодо реагування на зміни ментально, але й у проведенні проактивних заходів по адаптацію стратегій до можливих змін в потребах споживача чи конкурентному оточенню. Це також сприятиме створенню аналітичних звітів на основі не лише поточних даних, але й прогнозів, що зробить систему ще більш корисною для бізнесу. На додачу до використання штучного інтелекту, можна покращити текст запиту до AI-асистента, що може як прискорити відповідь, так і зробити результат точнішим. Це звісно потребує додаткового часу, обсягу токенів (умовних одиниць, які використовуються для обчислення витрат на обробку текстів), які надає AI для та великої кількості запитів для тестування.

Один зі способів полягає у автоматизації рекомендацій на основі отриманих даних. Наприклад, система може автоматично створювати поради для керівників щодо необхідності зміни цін, додавання нових товарів у популярні категорії або запуску маркетингових кампаній для збудження попиту на товари, яким спостерігаються спад популярності або попит за гострий часовий інтервал.

Також важливо враховувати значення підвищення безпеки системи. Оскільки система працюватиме з великим обсягом даних, включаючи комерційно важливу інформацію, необхідно забезпечити захист від витоків даних і кібератак. Можна застосовувати такі методи як шифрування даних під

час передачі та збереження, або ж наприклад, багаторівневе підтвердження особистостей для доступу до адмін-інтерфейсу. Також застосування інструментів для виявлення та запобігання несанкціонованому доступу.

Таким чином, проведене дослідження виявило ключові обмеження існуючої системи аналізу ринкового пошуку. В той же час, були визначені перспективні напрямки для вдосконалення системи. Наприклад, впровадження мультипарсингу для глибшого аналізу конкурентного середовища або оптимізація алгоритму для збільшення кількості отриманих товарів. Також, було запропоновано розглянути інтеграцію прогностичних алгоритмів та автоматизованих рекомендацій. Реалізація цих рішень сприятиме не лише подоланню виявлених недоліків, але й значному розширенню функціональності системи.

Запропоновані вдосконалення дозволять перейти від базового аналізу до інтегрованої та масштабованої платформи, що зможе забезпечити детальний моніторинг ринку, ефективну підтримку прийняття рішень та оптимізацію бізнес-процесів. У результаті система стане більш універсальною, гнучкою та здатною задовольняти потреби як малого, так і великого бізнесу. Таким чином, досягнуто важливих висновків, які формують основу для подальшого розвитку системи та її практичного застосування у сучасних ринкових умовах.

Висновки до третього розділу

У даному розділі було проведено дослідження працездатності системи аналізу ринкового пошуку, зокрема тестування парсеру та процесу аналізу ринкових даних. Результати підтвердили, що система успішно виконує свою основну функцію по збору, обробці та аналізі даних з різноманітних джерел ринку. Усі ключові компоненти продемонстрували стабільну роботу, забезпечили точність і надійність опрацьованих результатів.

За допомогою тестування, вдалося також визначити потенційні напрямки для вдосконалення системи. Одним із важливих аспектів, що потребує оптимізації, є зменшення часу обробки великих масивів даних, а також реалізація мультипарсингу. Крім того, з'явилася потреба у додаткових модулях візуалізації

результатів. Це безперечно дозволить користувачам зручніше аналізувати отриману інформацію і приймати обґрунтовані рішення на основі результатів аналізу.

Система довела свою здатність ефективно обробляти і представляти дані для подальшого аналізу. Пропоновані зміни сприятимуть створенню більш масштабованої та гнучкої платформи для аналізу ринку, що зможе ефективно підтримувати процеси прийняття рішень у реальних умовах.

Таким чином, розглянуті результати дослідження підтверджують досягнення основних цілей і створюють основу для подальшого розвитку системи. Завдяки реалізації запропонованих удосконалень система стане більш універсальною, здатною задовольняти різноманітні потреби бізнесу, а також забезпечить більш глибоке й ефективне прогнозування.

ВИСНОВКИ

У кваліфікаційній магістерській роботі реалізовано автоматизований підхід для аналізу ринку електротехніки, що базується на зборі, обробці та аналізі даних із вебсайтів конкурентів. Система спрямована на підвищення конкурентоспроможності інтернет-магазину шляхом формування актуального асортименту популярних товарів.

Досягнення поставленої мети стало можливим завдяки комплексному вирішенню завдань, а саме: було проведено дослідження сучасних методів збору та аналізу ринкових даних; розроблено архітектуру системи для автоматизації цих процесів; реалізовано алгоритм інтеграції даних із зовнішніх джерел та власної бази даних із використанням технологій штучного інтелекту.

Основу системи склала взаємодія MongoDB як основної бази даних і Prisma ORM, що забезпечило високу продуктивність, гнучкість у збереженні даних та зручність для розробника. Парсинг конкурентних вебсайтів відбувався з використанням спон-завдань, що дозволяє системі функціонувати в автономному режимі. Подальший аналіз отриманих даних із залученням OpenAI гарантував точність відбору популярних товарів для інтеграції в асортимент інтернет-магазину.

Запропонована система показала свою ефективність у виконанні завдань. Популярні товари регулярно оновлюються, що дозволяє оперативно адаптуватися до змін ринку. Збереження історичних даних створює підґрунтя для подальшого аналізу, включаючи побудову трендів, аналітичних звітів і візуалізацій. Такий підхід не лише підвищує зручність для користувачів, але й забезпечує бізнесу конкурентну перевагу.

Робота має потенціал для подальшого вдосконалення. Можливе розширення функціоналу, наприклад, шляхом інтеграції більш складних алгоритмів прогнозування, візуалізації даних та адаптації системи для інших сфер ринку. Результати роботи демонструють, що розроблений автоматизований підхід є не лише ефективним, але й універсальним інструментом для побудови конкурентних стратегій в умовах динамічного ринку.

Таким чином, досягнуто всіх поставлених цілей і завдань, підтверджено актуальність дослідження та практичну цінність отриманих результатів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гулак І.В, Савіцький Р.С. Аналіз ринкового пошуку на базі вебсайту електротехніки. Тези VII Всеукраїнської науково-технічної конференції «Комп'ютерні технології: інновації, проблеми, рішення – 2024», м. Житомир, 02 – 03 грудня 2024 р., 2024.
2. Mayer-Schönberger V., Cukier K. Big Data: A Revolution That Will Transform How We Live, Work, and Think / Viktor Mayer-Schönberger, Kenneth Cukier. – US publishing office located in New York: Eamon Dolan/Houghton Mifflin Harcourt, 2013. – 242 p.
3. Knowles S. Mastering Selenium WebDriver 4.0: Building Robust, Reliable, and Maintainable Test Automation Solutions / Simon Knowles. – Packt Publishing, 2022. – 400 c.
4. Mitchell R. Web Scraping with Python: Collecting More Data from the Modern Web / Ryan Mitchell. – 2-е видання. – O'Reilly Media, 2018. – 416 c.
5. Flanagan D. JavaScript: The Definitive Guide / David Flanagan. – 7-е видання. – O'Reilly Media, 2020. – 706 c.
6. Jurafsky D., Martin J.H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition / Daniel Jurafsky, James H. Martin. – Pearson, 2022. – 1024 p.
7. Sharma H. Data Parsing and Automation: A Practical Guide / Harsh Sharma. – Packt Publishing, 2021. – 312 c.
8. Erol Mrzic, Tarik Zaimovic, “Data Science Methods and Machine Learning Algorithm Implementations for Customized Pratical Usage” TEM Journal. Volume 9, Issue 3, Pages 1179-1185 [Електронний ресурс] – Режим доступу до ресурсу: https://www.temjournal.com/content/93/TEMJournalAugust_1179_1185.pdf
9. DataCamp: A beginners guide to the OpenAI API: Hands on tutorial and best practices. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.datacamp.com/tutorial/guide-to-openai-api-on-tutorial-best-practices>

10. Top 5 best practices for integrating ChatGPT in your app. [Електронний ресурс] – Режим доступу до ресурсу: <https://stormotion.io/blog/top-5-best-practices-integrating-chatgpt-app/>
11. DataCamp: A beginners guide to the OpenAI API: Hands on tutorial and best practices. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.datacamp.com/tutorial/guide-to-openai-api-on-tutorial-best-practices>
12. DataCamp: Openai function calling tutorial [Електронний ресурс] – Режим доступу до ресурсу: <https://www.datacamp.com/tutorial/open-ai-function-calling-tutorial>
13. File Search tool: Efficient Access to Stored Data. [Електронний ресурс] – Режим доступу до ресурсу: <https://platform.openai.com/docs/assistants/tools/file-search>
14. Community AWS. "AI running it is own code: agentic code interpreter. [Електронний ресурс] – Режим доступу до ресурсу: <https://community.aws/https://community.aws/content/2jOQC6L6A4SHH89yxZv84fEvqzD/ai-running-it-s-own-code-agentic-code-interpretercontent/2jOQC6L6A4SHH89yxZv84fEvqzD/ai-running-it-s-own-code-agentic-code-interpreter>
15. Martin R. Clean Architecture: A Craftsman's Guide to Software Structure and Design / Robert C. Martin. – Prentice Hall, 2017. – 432 с.
16. Kroenke D.M., Auer D.J. *Database Concepts* / David M. Kroenke, David J. Auer. – 9th edition. – Pearson, 2020. – 432 p.
17. Chodorow K. MongoDB: The Definitive Guide / Kristina Chodorow. – 3-е видання. – O'Reilly Media, 2019. – 514 с.
18. MongoDB. Офіційна документація [Електронний ресурс] – Режим доступу до ресурсу: <https://www.mongodb.com/docs>
19. Geers T. NestJS: A Progressive Node.js Framework / Thomas Geers. – Manning Publications, 2020. – 350 с.
20. Kropf T. Mastering NestJS: A Modern Node.js Framework / Kropf Thomas. – Packt Publishing, 2021. – 420 с.

21. Eichhorn M. Building Scalable Applications with NestJS / Michael Eichhorn. – Apress, 2022. – 340 с.
22. Segura M. Next.js Cookbook: Modern Web Development with React and Next.js / Manuel Segura. – Packt Publishing, 2022. – 314 с.
23. Snyder J. Learning Next.js: From Zero to Full Stack Development / John Snyder. – Manning Publications, 2021. – 380 с.
24. Fowler M. Patterns of Enterprise Application Architecture / Martin Fowler. – Addison-Wesley, 2002. – 560 с.
25. Date C. Introduction to Database Systems / Christopher Date. – 8th ed. – Pearson, 2003. – 936 p.
26. Taieb H. Prisma in Action: A Guide to Modern Database Access for Node.js and TypeScript / Haider Taieb. – Manning Publications, 2021. – 352 с.
27. Prisma. Офіційна документація Prisma [Електронний ресурс] – Режим доступу до ресурсу: <https://www.prisma.io/docs>
28. A Beginner's Guide To Cron Jobs [Електронний ресурс] – Режим доступу до ресурсу: <https://ostechnix.com/a-beginners-guide-to-cron-jobs/>
29. What is server side rendering? [Електронний ресурс] – Режим доступу до ресурсу: <https://solutionshub.epam.com/blog/post/what-is-server-side-rendering>
30. Next JS docs. Rendering. [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/docs/app/building-your-application/rendering>
31. Postman Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://learning.postman.com/docs/introduction/overview/>
32. Insomnia Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.insomnia.rest/insomnia/get-started>
33. McKinney W. Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter / Wes McKinney. – 2-е видання. – O'Reilly Media, 2022. – 579 с.
34. Wandschneider F. JavaScript: Async Programming and Performance / Fernando Doglio. – Apress, 2017. – 210 p.
35. Richardson L. RESTful Web APIs: Services for a Changing World / Leonard Richardson, Mike Amundsen, Sam Ruby. – O'Reilly Media, 2013. – 448 p.

36. Tanstack Query V5. API Documentation [Электронный ресурс] – Режим доступа до ресурсу: <https://tanstack.com/query/v5/>
37. Axios. Documentation [Электронный ресурс] – Режим доступа до ресурсу: <https://axios-http.com/uk/docs/intro>
38. Swager. Documentation [Электронный ресурс] – Режим доступа до ресурсу: <https://swagger.io/solutions/api-documentation/>
39. Overview HTTP. MDN documentation. [Электронный ресурс] – Режим доступа до ресурсу: <https://developer.mozilla.org/ru/docs/Web/HTTP/Overview>
40. What is software testing. [Электронный ресурс] – Режим доступа до ресурсу: <https://www.geeksforgeeks.org/software-testing-basics/>
41. Multithreading in Node.js. [Электронный ресурс] – Режим доступа до ресурсу: <https://blog.logrocket.com/multithreading-node-js-worker-threads/>

ДОДАТКИ

Лістинг коду файлу web-scraping.service.ts

```
import { Injectable, Logger } from '@nestjs/common';
import { Builder, By, Key, WebElement } from 'selenium-webdriver';
import * as chrome from 'selenium-webdriver/chrome';
import * as fs from 'fs-extra';
import { ProductsService } from '../products/products.service';
```

```
type Product = {
  name: string;
  price: number;
  rate: number;
  commentsCount: number;
  category: string;
};
```

```
@Injectable()
```

```
export class WebScrapingService {
  constructor(private readonly productsService: ProductsService) {}
  private readonly logger = new Logger(WebScrapingService.name);
  private userAgents = [
    'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:103.0) Gecko/20100101
Firefox/103.0',
    'Mozilla/5.0 (Macintosh; Intel Mac OS X 11.6; rv:89.0) Gecko/20100101
Firefox/89.0',
    'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:100.0) Gecko/20100101
Firefox/100.0',
    'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/100.0.4896.127 Safari/537.36',
    'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15
(KHTML, like Gecko) Version/15.1 Safari/605.1.15',
```

'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.61 Safari/537.36',

'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:91.0) Gecko/20100101 Firefox/91.0',

'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.102 Safari/537.36',

'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:89.0) Gecko/20100101 Firefox/89.0',

'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:92.0) Gecko/20100101 Firefox/92.0',

'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0.3 Safari/605.1.15',

'Mozilla/5.0 (X11; Ubuntu; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.93 Safari/537.36',

];

```
async scrape(categoryUrl: string) {  
  const options = new chrome.Options();  
  options.addArguments(  
    `user-agent=${this.userAgents[Math.floor(Math.random()  
this.userAgents.length))}]`,
```

*

```
  );  
  options.addArguments('--disable-blink-features=AutomationControlled');  
  options.addArguments('headless');
```

```
  const driver = await new Builder()  
    .forBrowser('chrome')  
    .setChromeOptions(options)  
    .build();
```

```
  const categoryName = categoryUrl.split('/').slice(-1)[0];
```

```

await driver.get(categoryUrl);

for (let i = 0; i < 5; i++) {
  await driver.findElement(By.tagName('body')).sendKeys(Key.END);
  await this.randomDelay(1, 3);
}

const productElements = await driver.findElements(
  By.css('div.products-catalog-grid__item'),
);
const productsList = await this.constructProductsArray(
  productElements,
  categoryName,
);

await driver.quit();

const sortedProducts = this.sortProductsByPopularity(productsList);
const dbProducts = await this.productsService.findAll({ pageSize: 50 });

await fs.writeFile(
  'products.json',
  JSON.stringify(
    { competitorProducts: sortedProducts, databaseProducts: dbProducts },
    null,
    4,
  ),
  'utf-8',
);
this.logger.log(
  'Scraping completed. Product details saved to products.json',

```

```

);

return sortedProducts;
}

private async randomDelay(minSeconds: number, maxSeconds: number) {
  const delay = Math.random() * (maxSeconds - minSeconds) + minSeconds;
  return new Promise((resolve) => setTimeout(resolve, delay * 1000));
}

private async constructProductsArray(
  elements: WebElement[],
  category: string,
) {
  const productDetailsList = [];
  for (const product of elements) {
    try {
      const name = await product
        .findElement(By.css('a.products-list-item__name'))
        .getText();
      const price = parseFloat(
        (
          await product
            .findElement(
              By.css('div.products-list-item-price__actions-price-current'),
            )
            .getText()
        )
        .slice(0, -1)
        .replace(' ', ''),
      );
    }
  }
}

```

```

let rate = 0.0;
try {
  rate = parseFloat(
    await product
      .findElement(By.css('span.rating-number-box__value'))
      .getText(),
  );
} catch (error) {
  this.logger.warn('Rate element not found, setting rate to 0.0');
}
const commentsCount = parseFloat(
  await product
    .findElement(By.css('div.products-list-item__reviews'))
    .getText(),
);
productDetailsList.push({ name, price, rate, commentsCount, category });
} catch (error) {
  this.logger.error(`An error occurred: ${error}`);
}
}
return productDetailsList;
}

```

```

private sortProductsByPopularity(products: Product[], limit = 20) {
  const calculatePopularity = (product: any) => {
    return product.rate * 2 + product.commentsCount - product.price * 0.1;
  };

  return products
    .sort((a, b) => calculatePopularity(b) - calculatePopularity(a))
    .slice(0, limit);
}

```

```
}  
}
```

ДОДАТОК Б

Лістинг коду файлу openai.service.ts

```
import { Injectable, InternalServerErrorException } from '@nestjs/common';  
import * as fs from 'fs';  
import OpenAIApi from 'openai';
```

```
@Injectable()
```

```
export class OpenAiService {  
  constructor(private readonly openai: OpenAIApi) {  
    this.openai = new OpenAIApi({  
      apiKey: process.env.OPENAI_API_KEY,  
      baseUrl: process.env.OPEN_AI_BASE_URL,  
      dangerouslyAllowBrowser: true,  
    });  
  }  
}
```

```
async createThread(id: string) {  
  try {  
    const fileId = id ?? process.env.UPLOADED_FILE_ID;  
    const thread = await this.openai.beta.threads.create({  
      messages: [  
        {  
          role: 'user',  
          attachments: [  
            {  
              file_id: fileId,  
              tools: [{ type: 'file_search' }],  
            },  
          ],  
        },  
      ],  
    });  
  }  
}
```

```
],
```

```
content:
```

```
'Analyze my uploaded file and your answer must be only array of strings, nothing more. Go through the competitorProducts array, conduct calculations to assess popularity based on factors like product rate, pricing trends, comments count, category and identify the most popular products from this data. No matter what language is in product name, translate this into your own language to continue do comparisons between competitorProducts and databaseProducts. Return me only an array of strings (product ids) from databaseProducts array that is nearest to analyzed data from competitorProducts.'
```

```
},
```

```
],
```

```
});
```

```
return thread.id;
```

```
} catch (err) {
```

```
  console.log('Error: ', err);
```

```
  throw new InternalServerErrorException(
```

```
    'Error while trying create thread',
```

```
  );
```

```
}
```

```
}
```

```
async createRun(threadId: string) {
```

```
  try {
```

```
    const assistantId = process.env.ASSISTANT_ID ?? "";
```

```
    const run = await this.openai.beta.threads.runs.create(threadId, {
```

```
      assistant_id: assistantId,
```

```
    });
```

```
    return run.id;
```

```
  } catch (err) {
```

```

        console.log('Error: ', err);
        throw new InternalServerErrorException('Error while trying create run');
    }
}

async uploadFile() {
    try {
        const file = await this.openai.files.create({
            purpose: 'assistants',
            file: fs.createReadStream('products.json'),
        });

        return file.id;
    } catch (err) {
        console.log('Error: ', err);
        throw new InternalServerErrorException('Error while trying upload file');
    }
}

async getListOfMessages({ threadId }: { threadId: string }) {
    try {
        const response = await this.openai.beta.threads.messages.list(threadId);

        return response.data;
    } catch (err) {
        console.log('Error: ', err);
        throw new InternalServerErrorException('Error while trying create run');
    }
}

async analizeMarket() {

```

```

const fileId = await this.uploadFile();
const threadId = await this.createThread(fileId);
// const threadId = process.env.THREAD_ID ?? "";
await this.createRun(threadId);

const delay = () => {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve(null);
    }, 20000);
  });
};

await delay();

const response = await this.getListOfMessages({ threadId });

const result = JSON.parse(
  // @ts-ignore
  response[0].content[0].text.value.replace(/```json|```/g, "").trim(),
);

return result;
}

async getPopularProducts() {
  const threadId = process.env.THREAD_ID ?? "";

  const response = await this.getListOfMessages({ threadId });
  const result = JSON.parse(
    // @ts-ignore
    response[0].content[0].text.value.replace(/```json|```/g, "").trim(),
  );
}

```

```
);

    return result;
}
}
```

ДОДАТОК В

Лістинг коду файлу tasks.service.ts

```
import { MarketResearchService } from '../market-research/market-re-
search.service';

import { Injectable } from '@nestjs/common';
import { Cron, CronExpression } from '@nestjs/schedule';
import { OpenAiService } from 'src/openai/openai.service';
import { OrderService } from 'src/order/order.service';
import { ProductsService } from 'src/products/products.service';
import { WebScrapingService } from 'src/web-scraping/web-scraping.service';

@Injectable()
export class TasksService {
  constructor(
    private readonly openAiService: OpenAiService,
    private readonly productsService: ProductsService,
    private readonly marketResearchService: MarketResearchService,
    private readonly webScrapingService: WebScrapingService,
    private readonly orderService: OrderService,
  ) {}

  @Cron(CronExpression.EVERY_1ST_DAY_OF_MONTH_AT_MID-
NIGHT)
  async ScrapeCron() {
    await this.webScrapingService.scrape(
```

```

        'https://comfy.ua/ua/charging-stations/',
    );
    const productIds = await this.openAiService.analyzeMarket();
    const products = await this.productsService.findProductsByIds(productIds);
    await this.marketResearchService.updateItems(products);

    console.log('=====');
    console.log('updated');
    console.log('=====');
}

@Cron(CronExpression.EVERY_1ST_DAY_OF_MONTH_AT_MID-
NIGHT)
async handleCron() {
    const startOfMonth = new Date();
    startOfMonth.setDate(1);
    startOfMonth.setHours(0, 0, 0, 0);
    const endOfMonth = new Date(startOfMonth);
    endOfMonth.setMonth(startOfMonth.getMonth() + 1);

    const orders = await this.orderService.getSeasonOrders();

    const productCount: { [key: string]: number } = {};

    orders.forEach((order) => {
        order.items.forEach((item) => {
            if (!productCount[item?.productId]) {
                productCount[item.productId] = 0;
            }
            productCount[item.productId] += item.quantity;
        });
    });
}

```

```
const mostPopularProducts = Object.entries(productCount)
  .sort(([a], [b]) => b - a)
  .slice(0, 10)
  .map(([productId]) => productId);
```

```
await this.productsService.updateMany({
  where: {},
  data: {
    isSeasonal: false,
  },
});
```

```
await this.productsService.updateMany({
  where: {
    id: {
      in: mostPopularProducts,
    },
  },
  data: {
    isSeasonal: true,
  },
});
}
```

ДОДАТОК Г

Лістинг коду файлу page.tsx (головна сторінка вебсайту)

```
import ProductService from '@api/services/ProductService';
import { Container } from '@components/layout/Container';
```

```

import ProductsCatalog from '@components/pages/home/catalog/ProductsCatalog';

import RecentlyViewedProducts from '@components/pages/home/catalog/RecentlyViewedProducts';

import Filters from '@components/pages/home/Filters';
import { CalculatorPresale } from '@components/shared/CalculatorPresale';
import { HotNewsTicker } from '@components/shared/HotNewsTicker';
import { NewsSlider } from '@components/shared/NewsSlider';

export default async function Home() {
  const filters = await ProductService.getFilters();
  const popularProducts = await ProductService.getProducts('/products', {
    type: 'popular',
  });
  const newProducts = await ProductService.getProducts('/products', {
    type: 'all',
  });
  return (
    <>
      <HotNewsTicker />

      <Container>
        <div className="pl-3 grid grid-cols-1 md:grid-cols-12 space-x-2">
          <Filters data={filters} />

          <section className="col-span-1 md:col-span-8 p-1 border border-grey">
            <NewsSlider />

            <ProductsCatalog type="popular" data={popularProducts.data} />
            <ProductsCatalog type="new" data={newProducts.data.slice(0, 6)} />
          </section>
        </div>
      </Container>
    </>
  );
}

```

```
        <RecentlyViewedProducts />
    </section>
</div>
</Container>
<CalculatorPresale />
</>
);
}
```