

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет водного господарства та природокористування
Навчально-науковий інститут кібернетики, інформаційних технологій та
інженерії
Кафедра комп'ютерних технологій та економічної кібернетики

Допущено до захисту:

Завідувач кафедри

_____ д. е. н., проф. П. М. Грицюк

«_____» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня «бакалавр»

за освітньо-професійною програмою «Інформаційні системи та технології»
спеціальності 126 «Інформаційні системи та технології»

на тему: «Інформаційна система e-commerce-платформи: застосування ШІ-
технологій для SEO-оптимізації інтернет-магазину домашнього текстилю»

Виконав:

здобувач вищої освіти 4 курсу, групи ІСТ-41

Трохимчук Максим Святославович

Керівник:

к.т.н., доцент Джоші О. І.

Рецензент:

д.е.н., професор Грицюк П.М.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет водного господарства та природокористування
Навчально-науковий інститут кібернетики, інформаційних технологій
та інженерії

Кафедра комп'ютерних технологій та економічної кібернетики

Освітньо-кваліфікаційний рівень – бакалавр
Освітньо-професійна програма «Інформаційні системи і технології»
Спеціальність 126 «Інформаційні системи і технології»

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерних технологій та
економічної кібернетики
д.е.н., професор П.М. Грицюк

“ _____ ” _____ 2026 року

**З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

здобувачу Трохимчуку Максиму Святославовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Інформаційна система e-commerce-платформи:
застосування III-технологій для SEO-оптимізації інтернет-магазину
домашнього текстилю

керівник роботи: Джоші Олена Іванівна, к.т.н.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджена наказом по університету від «23» квітня 2026 р. С №-476

2. Термін здачі студентом закінченої роботи 15.06.2026 р.
3. Вихідні дані до роботи: 1) документація щодо e-commerce-платформи
fionahome.com.ua; 2) Інтернет-ресурси щодо теми кваліфікаційної роботи; 2) наукові
публікації за темою кваліфікаційної роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить
розробити) 1) Теоретичні дослідження (аналіз предметної області; аналіз сучасних SEO-
підходів для e-commerce-платформи, аналіз III технологій для SEO-оптимізації), 2) Розробка
концептуальної моделі IC, 3) Реалізація інтеграції OpenAI API з CMS OpenCart, 3) Аналіз
ефективності SEO-оптимізації із застосуванням III

5. Перелік графічного матеріалу 1) обґрунтування вибору інструментального стеку;
2) ER-модель IC; 3) Backend ; 4) Frontend 5) результати дослідження

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Джоші Олена Іванівна, доцент		
2	Джоші Олена Іванівна, доцент		
3	Джоші Олена Іванівна, доцент		

Дата видачі завдання «19» січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Термін виконання етапів (роботи)	Примітка
1	Огляд літератури, збір статистичних даних	01.02.2026– 15.03.2026	
2	Розділ 1.	15.03.2026– 15.04.2026	
3	Розділ 2.	15.04.2026– 01.05.2026	
4	Розділ 3.	01.05.2026– 15.05.2026	
5	Підготовка пояснювальної записки	15.05.2026– 25.05.2026	
6	Написання вступу, висновків, реферату	25.05.2026– 01.06.2026	
7	Підготовка презентації роботи	01.06.2026– 09.06.2026	
8	Попередній захист роботи	09.06.2026	
9	Відгук керівника, рецензування роботи, перевірка на плагіат	09.06.2026– 15.06.2026	
10	Допуск до захисту	15.06.2026	

Студент _____ Трохимчук М. С.
(підпис) (прізвище і ініціали)

Керівник кваліфікаційної роботи

к.т.н., доцент _____ Джоші О.І.
(підпис) (прізвище і ініціали)

ЗАЯВА

щодо самостійності виконання випускної кваліфікаційної роботи

Я, _____ (ПІБ),
студент(ка) _____ курсу _____ групи _____ ННІ _____ заявляю: моя випускна
кваліфікаційна робота на тему
« _____

_____ » (назва роботи), яка надається в екзаменаційну комісію із захисту
бакалаврської роботи _____

_____ (назва
спеціальності) для захисту, виконана самостійно і не містить плагіату. Всі
запозичення з друкованих та електронних джерел, у тому числі із захищених
раніше випускових кваліфікаційних робіт, кандидатських і докторських
дисертацій мають відповідні посилання. Я не використовував(ла) шахрайські
методи заміни символів чи інші способи модифікації тексту. Я
ознайомлений(а) з чинним Порядком перевірки навчальних, кваліфікаційних,
навчально-методичних та наукових робіт на наявність ознак академічного
плагіату в НУВГП та Положенням про академічну доброчесність в НУВГП,
за яким виявлення плагіату є підставою для відмови в допуску моєї роботи до
захисту та застосування відповідних санкцій (академічної відповідальності).



Національний університет
водного господарства
та природокористування

Документ прийнятий

Метадані

ДОКУМЕНТ

Заголовок

2026_Trokhymchuk_126_bakalavr.docx

Автор

Трохимчук Максим Святославович

Науковий керівник / Експерт

Трохимчук Максим Святославович

ІД документу

334310607

ОРГАНІЗАЦІЯ

Назва організації

National University of Water and Environmental Engineering

підрозділ

National University of Water and Environmental Engineering

ЗВІТ

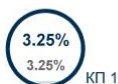
Дата звіту

6/14/2026

Дата редагування

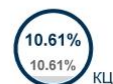
Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



18234

Кількість слів



135296

Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		62
Інтервали		1
Мікропробіли		665
Білі знаки		0
Парафрази (SmartMarks)		33

Джерела

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз

Колір тексту

#	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://eq3.nuwm.edu.ua/19464/1/04-05-49%D0%9C.pdf	49 (0.27 %)

А К Т
перевірки випускової кваліфікаційної роботи
автора Трохимчука Максима Святославовича
на рівень запозичень та можливих маніпуляцій з текстом

Відповідно до даних сервісу StrikePlagiarism файл «2026_Trokhymchuk_126_bakalavr.docx», містить: ____ % коефіцієнта подібності; ____ % коефіцієнта цитованості; ____ замінених букв; ____ інтервалів; ____ мікропробілів; ____ білих знаків; ____ парафрази.

За результатами перевірки засвідчую, що:

Автор кваліфікаційної роботи **Трохимчук Максим Святославович:**

- самостійно виконав (ла) кваліфікаційну роботу;
- коректно посилався (лась) на використані інформаційні джерела;
- в роботі відсутні ознаки академічної недоброчесності.

Керівник кваліфікаційної роботи _____ (О.І.Джоші)
(підпис)

(дата)

АНОТАЦІЯ

Трохимчук М. С. Інформаційна система e-commerce-платформи: застосування ІІІ-технологій для SEO-оптимізації інтернет-магазину домашнього текстилю. Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр»: 69 ст., 24 рис., 11 табл., 1 додаток на 25 ст., 37 літературних джерел.

Об'єкт дослідження – інформаційна система e-commerce-платформи.

Предмет дослідження – застосування ІІІ-технологій для SEO-оптимізації.

Методи дослідження – методи аналізу джерел інформації, системного аналізу, класифікації, моделювання, системного підходу, а також програмування та тестування програмної реалізації та ефективності системи.

Кваліфікаційна робота присвячена розробці інформаційної системи автоматизованої SEO-оптимізації контенту для інтернет-магазину домашнього текстилю FionaHome, що функціонує на платформі OpenCart. В роботі проведено аналіз існуючих підходів до SEO-оптимізації e-commerce-платформ та досліджено можливості застосування сучасних великих мовних моделей (LLM) для автоматизованої генерації текстового контенту. Розроблена система реалізована у вигляді зовнішнього модуля адміністрування з веб-інтерфейсом і підтримує три ІІІ-провайдери: Google Gemini, OpenAI GPT та Anthropic Claude. Система забезпечує автоматичну генерацію та оновлення описів товарів, мета-тегів і безпосередньо в СКБД MariaDB. Реалізовано модуль семантичної кластеризації пошукових запитів на основі методу K-Means із застосуванням багатомовних векторних ембедингів, валідацію згенерованого контенту за критеріями обсягу, мови та якості, резервне копіювання SEO-полів перед перезаписом, а також детальне логування операцій і підрахунок витрат API.

Ключові слова: SEO-оптимізація, великі мовні моделі, e-commerce, OpenCart, генерація контенту, мета-теги, семантична кластеризація, Python, Flask, MariaDB.

ЗМІСТ

ВСТУП.....	9
РОЗДІЛ 1. ХАРАКТЕРИСТИКА ТА АНАЛІЗ СТАНУ ВИВЧЕННЯ ПРОБЛЕМИ.....	13
1.1 Аналіз предметної області та особливостей e-commerce-платформ	13
1.2 Аналіз існуючих підходів до SEO-оптимізації інтернет-магазинів.	18
1.3 Аналіз та вибір ІІІ-технологій для задач SEO	23
РОЗДІЛ 2. ОПИС МОДЕЛІ ТА МЕТОДІВ РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ E-COMMERCE-ПЛАТФОРМИ.....	29
2.1 Розробка концептуальної моделі системи.....	29
2.2 ER-модель та нормалізація відношень бази даних.....	35
2.3 Архітектура модуля ІІІ-оптимізації контенту	45
РОЗДІЛ 3. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ	58
3.1 Backend.....	58
3.2 Frontend	65
3.3 Реалізація ІІІ-модуля SEO-оптимізації	70
3.4 Запити інформаційної системи	80
ВИСНОВКИ	83
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	85
ДОДАТКИ	89

ВСТУП

Війна спричинила масове зміщення продавців та покупців у цифровий простір, що суттєво прискорило зростання ринку електронної комерції в Україні в межах розвитку загальної цифрової економіки. Чимало суб'єктів малого й середнього підприємництва нині використовують інтернет-магазини як основний (а нерідко єдиний) канал збуту. У зв'язку з цим проблема високої пошукової видимості таких ресурсів стає дедалі **актуальнішою**. Таким чином SEO-оптимізація e-commerce-платформ набуває не лише технічного, але й прямого комерційного значення: низькі позиції у видачі, відсутність унікальних метатегів зображень призводять до втрати потенційних покупців і зниження конверсії. Незважаючи на широке розповсюдження готових CMS-рішень для побудови інтернет-магазинів, питання систематичної SEO-оптимізації великих каталогів залишається практично невирішеним для переважної більшості малих і середніх e-commerce-підприємств в Україні. Більшість із них не має виділених SEO-фахівців і не застосовує автоматизованих методів генерації контенту, обмежуючись ручним заповненням метатегів для найпопулярніших товарів. Така ситуація створює системний ризик низької пошукової видимості для товарів довгого хвоста, що безпосередньо впливає на рівень органічного трафіку і продажів.

Метою кваліфікаційної роботи є дослідження та практична реалізація інформаційної системи e-commerce-платформи із застосуванням ІІТ-технологій для автоматизованої SEO-оптимізації контенту інтернет-магазину домашнього текстилю на базі CMS OpenCart.

Досягнення поставленої мети передбачає вирішення таких **завдань**:

1. Провести аналіз сучасних SEO-підходів для e-commerce-платформ та особливостей CMS OpenCart як технологічної основи системи.

2. Проаналізувати наявні ІІІ-технології для генерації текстового контенту та обґрунтувати вибір моделі для вирішення завдань SEO-оптимізації.
3. Розробити концептуальну модель інформаційної системи, ER-модель бази даних і архітектуру AI-модуля автоматизованої генерації SEO-контенту.
4. Реалізувати програмну інтеграцію OpenAI API з CMS OpenCart для автоматичної генерації описів товарів.
5. Впровадити механізм семантичної кластеризації товарних категорій на основі NLP-підходів.
6. Проаналізувати ефективність автоматизації SEO-оптимізації шляхом порівняння показників до та після впровадження системи.

Об'єктом дослідження є інформаційна система е-commerce-платформи.

Предметом дослідження є застосування ІІІ-технологій для автоматизації SEO-оптимізації.

Під час дослідження застосовано такі підходи: метод системного аналізу – для вивчення будови е-commerce-платформи та встановлення її основних складників; функціональне моделювання за допомогою UML-діаграм – для формування концептуальної схеми бізнес-процесів; реляційне моделювання – для створення ER-моделі бази даних; технології NLP та семантичної кластеризації – для об'єднання товарних категорій за смисловою подібністю; методи API-інтеграції – для підключення OpenCart до хмарного сервісу OpenAI; автоматизоване генерування контенту з використанням великих мовних моделей (LLM) – для підготовки SEO-орієнтованих текстів.

Наукова новизна роботи полягає у розробці підходу до інтеграції технологій штучного інтелекту (на основі моделі GPT-4o) з CMS OpenCart для автоматизованої генерації SEO-контенту, а також у застосуванні NLP-методів

семантичної кластеризації товарних категорій для підвищення релевантності метатегів і описів.

Практичні результати дослідження отримано шляхом впровадження розробленої системи на реальному комерційному проєкті – інтернет-магазині домашнього текстилю, що спеціалізується на реалізації постільної білизни, рушників, покривал, тюлю, скатертин, ранфорсу та супутньої продукції. Платформа функціонує під реальним комерційним навантаженням на базі CMS OpenCart, що забезпечує дослідженню достовірний прикладний контекст.

Практична значущість роботи визначається тим, що розроблений AI-модуль може бути безпосередньо впроваджений у процес супроводу e-commerce-платформ на базі OpenCart. Автоматизація генерації meta title, meta description та описів товарів дозволяє суттєво заощадити час SEO-спеціаліста, масштабувати зусилля на тисячі товарних позицій та покращити загальні SEO-показники інтернет-магазину. Матеріали роботи апробовані в умовах реального функціонуючого підприємства. Результати роботи системи зафіксовано у вигляді таблиць з кількістю оброблених товарів, середнім часом генерації, довжиною описів, кількістю токенів, відсотком унікальності згенерованого контенту та розрахованим часом економії порівняно з ручним SEO-наповненням. Структура роботи відповідає логіці послідовного вирішення поставлених завдань і складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 80 сторінок основного тексту. Робота містить 20 рисунків і 18 таблиць. Список використаних джерел налічує 50 найменувань, серед яких наукові монографії, статті у фахових виданнях, технічна документація і електронні ресурси.

Програмні продукти, що використовувались під час написання кваліфікаційної роботи: Python 3.14, IDE PyCharm, VS Code.

Під час підготовки цього наукового дослідження використовувалися інструменти штучного інтелекту: chatgpt. Вони застосовувалися для пошуку літератури. Усі розділи роботи, під час написання яких використовувався ШІ, були перевірені та відредаговані автором особисто. Наукові положення, результати та висновки є власним інтелектуальним внеском автора.

РОЗДІЛ 1. ХАРАКТЕРИСТИКА ТА АНАЛІЗ СТАНУ ВИВЧЕННЯ ПРОБЛЕМИ

1.1 Аналіз предметної області та особливостей e-commerce-платформ

Збільшення обсягів ринку електронної комерції нині є стійкою тенденцією, яка виникає на тлі пришвидшеного розвитку цифрових технологій, еволюції споживацької поведінки та кращої доступності глобальної мережі. Електронна комерція охоплює численні види господарських операцій, що здійснюються через онлайн-канали: це і продаж товарів із послугами, і виконання маркетингових, логістичних та аналітичних функцій у цифровому середовищі. В Україні інтернет-магазини стали невіддільною частиною торговельної системи, що особливо помітно в сегменті товарів для дому, де споживачі віддають перевагу комфорту під час вибору й детальному ознайомленню з властивостями продукту без потреби йти до фізичної точки продажу.

Зазначений магазин реалізує широкий асортимент товарів у категоріях постільної білизни, рушників, скатертин, штор та супутніх виробів. Підприємство функціонує на конкурентному ринку, де успіх онлайн-торгівлі безпосередньо визначається якістю цифрової присутності та видимістю ресурсу в пошукових системах. Саме тому дослідження методів покращення органічного трафіку із застосуванням штучного інтелекту становить не лише теоретичний, а й виражений практичний інтерес (рис. 1.1) [3, с. 24].

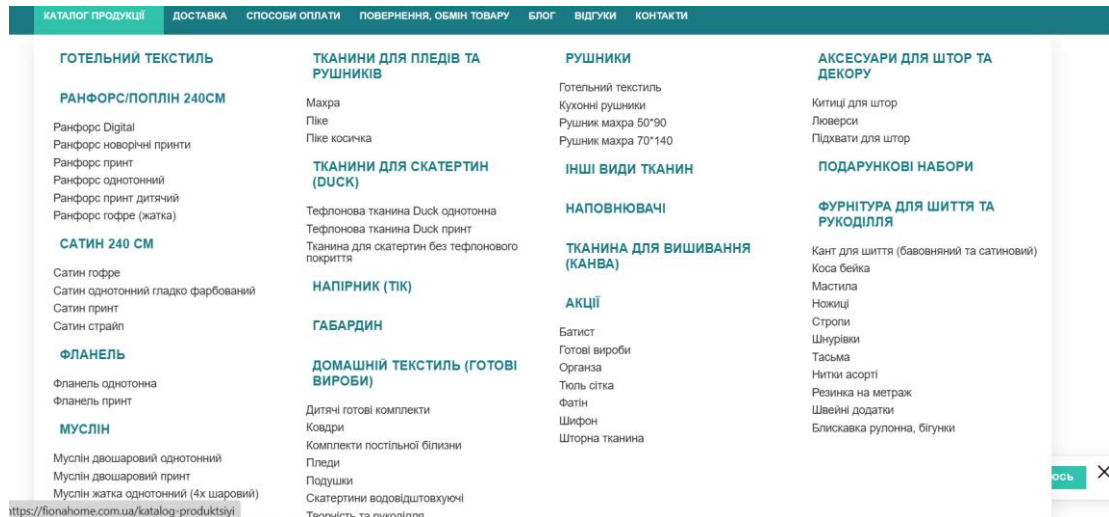


Рис. 1.1 Каталог продукції магазину

Е-commerce-платформа у загальному розумінні є комплексом програмних, апаратних та організаційних засобів, що забезпечують повний цикл торговельних операцій в онлайн-середовищі. До основних функціональних підсистем такої платформи належать: каталог товарів з механізмами фільтрації та пошуку, система управління замовленнями, модуль обробки платежів, підсистема управління клієнтською базою, інструменти маркетингу та аналітики, а також засоби інтеграції з зовнішніми сервісами. Взаємодія між цими підсистемами забезпечує безперервний операційний процес і формує користувацький досвід, що безпосередньо впливає на конверсію.

Архітектурно сучасні e-commerce-платформи будуються за багаторівневим принципом. Клієнтський рівень реалізується засобами браузера або мобільного застосунку, серверний рівень відповідає за бізнес-логіку, а рівень даних забезпечує зберігання та маніпулювання інформацією. Серед найпоширеніших архітектурних підходів виокремлюють монолітну

архітектуру, властиву системам на кшталт OpenCart та Magento, та мікросервісну, характерну для масштабних платформ рівня Amazon чи Rozetka. Кожен із підходів має власні переваги щодо простоти розгортання, масштабованості та вартості обслуговування.

Таблиця 1.1 – Порівняльна характеристика популярних CMS для e-commerce

Платформа	Мова реалізації	Тип ліцензії	Підтримка SEO	Складність налаштування	Масштабованість
OpenCart	PHP	Open Source	Базова	Низька	Середня
WooCommerce	PHP	Open Source	Розширена	Середня	Висока
Magento	PHP	Open Source / Commercial	Висока	Висока	Дуже висока
Shopify	Proprietary	SaaS	Висока	Низька	Висока
PrestaShop	PHP	Open Source	Середня	Середня	Середня

Як видно з таблиці 1.1, OpenCart є зручним вибором для малого та середнього бізнесу завдяки доступності та відносній простоті налаштування, однак поступається конкурентам у вбудованих засобах SEO-оптимізації. Ця обставина робить актуальним впровадження зовнішніх інструментів, зокрема заснованих на технологіях штучного інтелекту [1, с. 47].

Сегмент домашнього текстилю має ряд специфічних особливостей з точки зору організації e-commerce. По-перше, товари у цій категорії відрізняються значною різноманітністю атрибутів: склад тканини, щільність, колір, розмір, виробник, наявність набору чи одиничного виробу. Ефективне представлення цих атрибутів у структурі каталогу та в мета-даних сторінок є

критично важливим для якості пошукової видачі. По-друге, покупці у цій ніші часто здійснюють пошук за семантично близькими, але різними запитами: «постільна білизна двоспальна бязь», «комплект постільної білизни 200x220», «м'яка постільна білизна для дітей» тощо. Охоплення широкого кластеру семантики вимагає систематичної роботи з контентом, яка значно ускладнюється зі зростанням асортименту.

По-третє, у текстильному сегменті велике значення мають зображення товарів, оскільки покупець не може фізично доторкнутися до виробу.

Магазин fionahome.com.ua функціонує в умовах інтенсивної конкуренції на українському ринку. Основними конкурентами виступають як великі маркетплейси (Rozetka, Prom.ua), так і спеціалізовані онлайн-магазини текстилю. Перевага спеціалізованих магазинів перед маркетплейсами полягає у можливості формувати сильний тематичний авторитет домену та цільовий семантичний профіль, що при грамотній SEO-стратегії дозволяє конкурувати за органічний трафік у нішевих запитах.

Технічна платформа OpenCart, на якій функціонує досліджуваний магазин, надає розробникам доступ до бази даних, де зберігаються описи товарів, мета-теги, URL-адреси та інші SEO-релевантні атрибути. Структура організована навколо центральних таблиць продуктів (`oc_product`, `oc_product_description`), категорій (`oc_category`, `oc_category_description`) та SEO-URL (`oc_seo_url`). Доступ до цих таблиць програмним шляхом відкриває можливість масштабної автоматизованої обробки контенту, що є технічною передумовою для інтеграції ШІ-інструментів [2, с. 112].

Інформаційна система e-commerce-платформи як об'єкт дослідження охоплює не лише програмне забезпечення, а й сукупність інформаційних потоків, що циркулюють між підсистемами. З позиції системного аналізу, така

система може бути представлена у вигляді схеми взаємодії компонентів, де ключовими вузлами інтеграції ІІІ-технологій є контентна підсистема та підсистема метаданих (рис. 1.2).

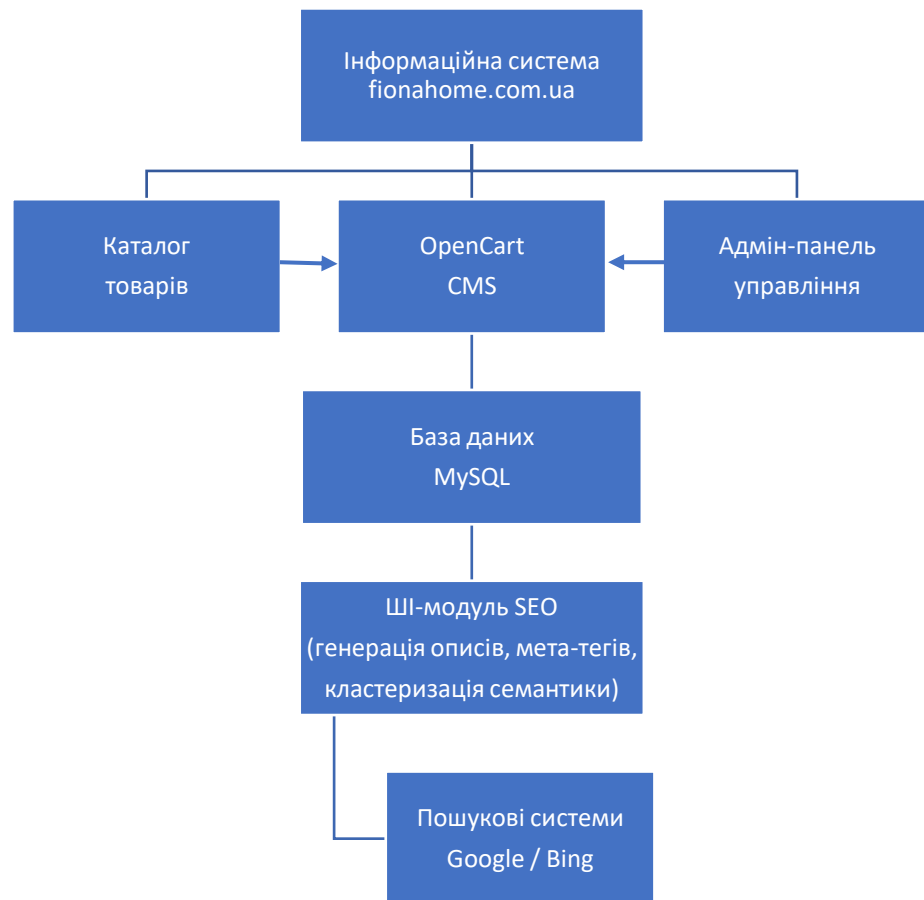


Рис. 1.2. Концептуальна схема інформаційної системи e-commerce-платформи з інтегрованим ІІІ-модулем SEO

Наведена схема ілюструє місце ІІІ-модуля в загальній архітектурі інформаційної системи. Він взаємодіє безпосередньо з базою даних, отримуючи та записуючи дані про товари й категорії, і забезпечує передачу оптимізованих метаданих у бік пошукових систем через механізм

індексування. Таке архітектурне рішення є найбільш органічним, оскільки не порушує цілісності ядра системи та зберігає сумісність з оновленнями [4].

Отже, e-commerce-платформа домашнього текстилю є складною інформаційною системою з чіткими вимогами до контентної якості та SEO-атрибутів товарних сторінок. Специфіка ніші, конкурентне середовище та технічні можливості формують передумови для дослідження та впровадження ШІ-рішень у задачах оптимізації.

1.2 Аналіз існуючих підходів до SEO-оптимізації інтернет-магазинів

Пошукова оптимізація (Search Engine Optimization, SEO) є комплексом заходів, спрямованих на покращення позицій вебсайту у результатах органічної пошукової видачі. Для інтернет-магазинів SEO набуває особливого значення, оскільки органічний трафік є одним із найефективніших каналів залучення потенційних покупців з точки зору співвідношення витрат і результату. На відміну від контекстної реклами, яка потребує постійних фінансових вкладень, органічний трафік, сформований якісною SEO-оптимізацією, генерує стабільний потік відвідувачів навіть без поточних рекламних витрат.

У сучасній практиці SEO-оптимізацію інтернет-магазинів прийнято розглядати у двох основних площинах: технічна оптимізація та контентна оптимізація. Технічна оптимізація охоплює питання швидкодії сайту, коректності структури URL, наявності та правильності файлів sitemap.xml і robots.txt, оптимізації зображень, адаптивності до мобільних пристроїв та реалізації мікророзмітки Schema.org. Контентна оптимізація зосереджується на якості текстового наповнення сторінок товарів і категорій, семантичній

релевантності мета-тегів title та description, структурі заголовків h1–h6 та внутрішній перелінковці [6, с. 14].

Ключовим компонентом контентної SEO-оптимізації є семантичне ядро – структурований перелік пошукових запитів, за якими цільова аудиторія шукає відповідні товари. Для інтернет-магазину з великим асортиментом формування семантичного ядра є трудомістким процесом, що включає збирання первинного масиву запитів, їх кластеризацію за змістом і частотністю, розподіл між сторінками сайту та подальший моніторинг. Традиційно цей процес виконується фахівцями з SEO вручну із залученням інструментів на кшталт Google Keyword Planner, Ahrefs чи SEMrush.

Окремою задачею є написання унікальних описів для кожного товару. В контексті магазину текстилю, де каталог може нараховувати сотні і тисячі позицій, створення якісного опису для кожного виробу вручну є практично нездійсненним завданням. Внаслідок цього більшість магазинів використовують шаблонні описи або запозичують їх у постачальників, що призводить до дублювання контенту і негативно впливає на пошукове ранжування.

Аналіз підходів до SEO-оптимізації інтернет-магазинів дозволяє виокремити кілька усталених методологій.

Перший підхід – класична ручна оптимізація – передбачає залучення SEO-спеціаліста, який формує семантичне ядро, готує технічне завдання для копірайтера та контролює впровадження змін на сайті. Цей підхід забезпечує найвищу якість результату, але є дорогим і непридатним для масштабування при великому асортименті.

Другий підхід – шаблонна автоматизація – полягає у створенні шаблонів для мета-тегів і описів, куди автоматично підставляються атрибути товарів з

бази даних. Наприклад, шаблон title може мати вигляд: «{Назва товару} – купити в Україні | FionaHome». Цей підхід легко реалізується в OpenCart через стандартні механізми CMS і не потребує спеціальних знань. Проте він призводить до механічно подібних мета-тегів, які не враховують семантичні нюанси конкретного товару і поступово втрачають ефективність у міру посилення конкуренції [8, с. 87].

Третій підхід – напівавтоматизована оптимізація – поєднує шаблонну генерацію зі спеціалізованими SEO-плагінами та сторонніми сервісами. Для OpenCart існує ряд розширень, що забезпечують автоматичне формування sitemap, управління канонічними URL, редиректи, роботу з метатегами на рівні категорій тощо. Разом із тим такі плагіни не вирішують фундаментальну проблему унікальності контенту.

Четвертий підхід – SEO-оптимізація на основі штучного інтелекту – є відносно новим і найбільш перспективним. Він передбачає використання великих мовних моделей та інших ШІ-інструментів для автоматизованої генерації унікальних описів, оптимізованих мета-тегів, структурованих кластерів семантики та рекомендацій щодо внутрішньої перелінковки. Детальний аналіз цього підходу розглядається у підрозділі 1.3.

Важливим аспектом SEO для інтернет-магазинів є робота зі структурою категорій. Ієрархія категорій визначає архітектуру сайту і впливає на розподіл ваги посилань між сторінками. Для магазину текстилю оптимальна структура категорій може виглядати таким чином: (рис 1.3)



КАТАЛОГ ПРОДУКТІВ	ДОСТАВКА	СПОСОБИ ОПЛАТИ	ПОВЕРНЕННЯ, ОБМІН ТОВАРІВ	БЛОГ	ВІДГОВІДІ	КОНТАКТИ
ГОТЕЛЬНИЙ ТЕКСТИЛЬ РАНФОРС/ПОПЛІН 240СМ Ранфорс Digital Ранфорс новорічні принти Ранфорс принт Ранфорс однотонний Ранфорс принт дитячий Ранфорс гофре (жатка) САТИН 240 СМ Сатин гофре Сатин однотонний гладко фарбований Сатин принт Сатин страйп ФЛАНЕЛЬ Фланель однотонна Фланель принт МУСЛІН Муслін двошаровий однотонний Муслін двошаровий принт Муслін жатка однотонний (4х шаровий)	ТКАНИНИ ДЛЯ ПЛЕДІВ ТА РУШНИКІВ Махра Піке Піке косичка ТКАНИНИ ДЛЯ СКАТЕРТИН (DUSK) Тефлонова тканина Duck однотонна Тефлонова тканина Duck принт Тканина для скатертин без тефлонового покриття НАПІРНИК (ТІК) ГАБАРДИН ДОМАШНІЙ ТЕКСТИЛЬ (ГОТОВІ ВИРОБИ) Дитячі готові комплекти Ковдри Комплекти постільної білизни Пледки Подушки Скатертини водовідштовхуючі Творчість та рукоділля Халати Наволочки Пелюшки муслін/фланель	РУШНИКИ Готельний текстиль Кухонні рушники Рушник махра 50*90 Рушник махра 70*140 ІНШІ ВИДИ ТКАНИН НАПОВНЮВАЧІ ТКАНИНА ДЛЯ ВИШИВАННЯ (КАНВА) АКЦІЇ Батист Готові вироби Органза Тюль сітка Фатін Шифон Шторна тканина	АКСЕСУАРИ ДЛЯ ШТОР ТА ДЕКОРУ Китиці для штор Люверси Підхват для штор ПОДАРУНКОВІ НАБОРИ ФУРНІТУРА ДЛЯ ШИТТЯ ТА РУКОДІЛЛЯ Кант для шиття (бавовняний та сатиновий) Коса бейка Мастила Ножиці Стропи Шнурівки Тасьма Нитки асорті Резинка на метраж Швейні додатки Блискавка рулонна, бігунки			

Рис. 1.3. Орієнтовна та реальна ієрархічна структура категорій інтернет-магазину домашнього текстилю

Для кожного рівня ієрархії вимоги до SEO-оптимізації відрізняються. Сторінки верхнього рівня категорій орієнтовані на широкі, високочастотні запити з великим обсягом пошуку, тоді як підкатегорії і картки товарів оптимізуються під середньочастотні та низькочастотні запити з більш конкретним комерційним наміром. Така стратегія, відома як «довгий хвіст» (long-tail SEO), є особливо ефективною для спеціалізованих магазинів, де конкуренція за нішеві запити значно нижча, ніж за загальні [9].

Таблиця 1.2 – Порівняльний аналіз підходів до SEO-оптимізації інтернет-магазинів

Підхід	Якість контенту	Масштабованість	Вартість	Швидкість впровадження	Унікальність
Ручна оптимізація	Висока	Низька	Висока	Низька	Висока
Шаблонна автоматизація	Низька	Висока	Низька	Висока	Низька
Напівавтоматизована	Середня	Середня	Середня	Середня	Середня
ІІІ-оптимізація	Висока	Висока	Середня	Висока	Висока

Аналіз табл. 1.2 переконливо свідчить про те, що підхід на основі штучного інтелекту є єдиним, який поєднує високу якість генерованого контенту з можливістю масштабування на великий каталог. Це пояснює зростаючий інтерес до ІІІ-інструментів у SEO-спільноті та є теоретичним обґрунтуванням обраного напрямку дослідження.

Важливим напрямом SEO-роботи є також моніторинг позицій та аналіз ефективності оптимізації. Серед основних метрик, що відстежуються, виокремлюють: позиції ключових слів у пошуковій видачі, органічний трафік, показник відмов, глибину перегляду, а також конверсію з органічних переходів. Для інтернет-магазину fionahome.com.ua джерелом даних для

такого аналізу слугують Google Search Console та Google Analytics 4, інтеграція яких з CMS OpenCart здійснюється через відповідні розширення або вбудовані механізми підключення скриптів [10, с. 56].

Узагальнюючи аналіз існуючих підходів, можна сформулювати ключову проблему: жоден із традиційних підходів не забезпечує одночасно високої якості SEO-контенту і можливості його масштабованого виробництва. Впровадження ШІ-технологій є напрямом, що дозволяє подолати цю суперечність, формуючи нову парадигму автоматизованої контентної оптимізації, адаптованої до специфіки конкретної товарної ніші.

1.3 Аналіз та вибір ШІ-технологій для задач SEO

Штучний інтелект у сфері SEO-оптимізації інтернет-магазинів застосовується за кількома основними напрямками: автоматизована генерація текстового контенту, семантична кластеризація запитів, аналіз конкурентів, прогнозування трендів пошукового попиту та персоналізація контенту. Кожен із цих напрямів спирається на власний підклас ШІ-технологій, вибір між якими визначається характером задачі, вимогами до якості результату та наявними технічними ресурсами.

Центральне місце серед ШІ-технологій, релевантних задачам SEO, займають великі мовні моделі (Large Language Models, LLM). Великі мовні моделі – це нейронні мережі, навчені на масштабних текстових корпусах і здатні генерувати, перефразовувати, перекладати та аналізувати текст з якістю, наближеною до людської. Для задач SEO-оптимізації інтернет-магазину найбільш практичними є такі застосування LLM: генерація унікальних описів товарів на основі структурованих атрибутів, написання

оптимізованих мета-тегів title і description, формулювання заголовків h1–h3 з органічним включенням ключових слів, генерація відповідей на поширені запитання (FAQ-блоків) та формування контенту для сторінок категорій [11].

На момент проведення дослідження на ринку доступні кілька класів LLM-рішень, які відрізняються архітектурою, можливостями та умовами використання. Комерційні API-сервіси, зокрема OpenAI GPT-4o, Anthropic Claude та Google Gemini, надають доступ до потужних моделей через API з оплатою за кількість оброблених токенів. Відкриті моделі, такі як Meta LLaMA та Mistral, можуть розгортатися на власній інфраструктурі, що забезпечує повний контроль над даними та відсутність операційних витрат на API-запити після початкового інвестування в обчислювальні ресурси.

Для розуміння ролі різних ШІ-технологій у комплексному вирішенні задач SEO-оптимізації доцільно розглянути їх у системному взаємозв'язку. На рис. 1.3 представлено концептуальну схему застосування ШІ-технологій у SEO-циклі інтернет-магазину (рис. 1.4) [12].

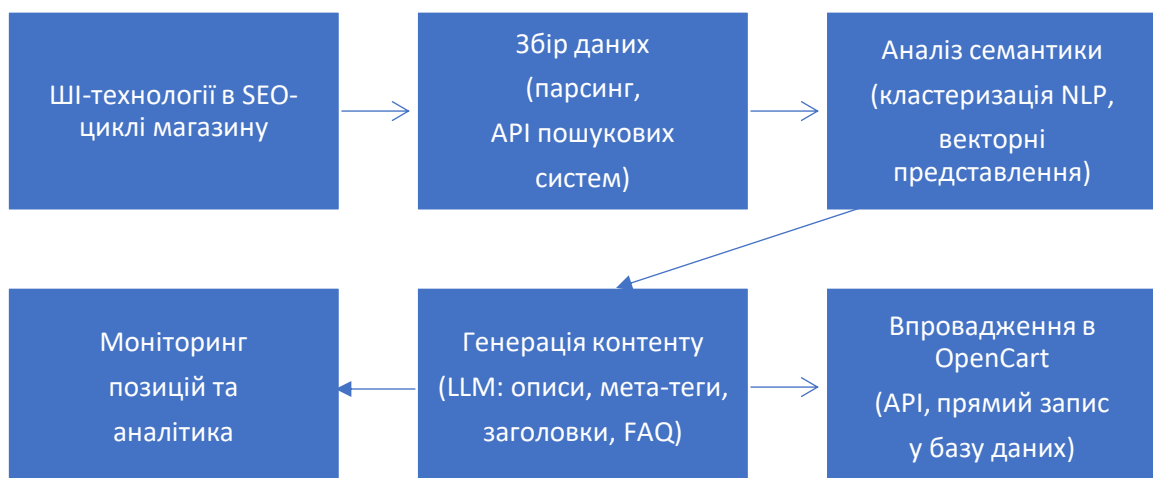


Рис. 1.4. Концептуальна схема застосування ІІІ-технологій у SEO-циклі інтернет-магазину

Окрім LLM, важливу роль у SEO-задачах відіграють методи обробки природної мови (Natural Language Processing, NLP), зокрема векторні представлення слів і речень. Такі методи, реалізовані у бібліотеках Word2Vec, FastText та трансформерних моделях sentence-transformers, дозволяють вимірювати семантичну близькість між запитами і групувати їх у кластери. Це є технічною основою автоматизованої кластеризації семантичного ядра – задачі, що традиційно виконується вручну і займає значний час фахівця.

Алгоритми машинного навчання також застосовуються для прогнозування динаміки пошукового попиту. Аналіз часових рядів пошукового трафіку за допомогою моделей ARIMA, LSTM або Prophet дозволяє виявляти сезонні тренди і завчасно готувати контент для запитів, популярність яких зростає у певні пори року. Для ринку домашнього текстилю такий аналіз є особливо релевантним, оскільки попит на ряд категорій товарів (наприклад, термобілизна, новорічний текстиль) має виражену сезонну складову (табл. 1.3) [14].

Таблиця 1.3 – Аналіз ІІІ-технологій за задачами SEO-оптимізації інтернет-магазину

Задача SEO	ІІІ-технологія	Приклади інструментів	Складність інтеграції	Очікуваний ефект
Генерація описів товарів	LLM (GPT-4o, Claude)	OpenAI API, Anthropic API	Середня	Висока унікальність контенту
Генерація мета-тегів	LLM	OpenAI API	Низька	Покращення CTR
Кластеризація семантики	NLP (sentence-transformers)	Python, Hugging Face	Висока	Точніша структура сайту
Прогнозування попиту	ML (LSTM, Prophet)	Python, TensorFlow	Висока	Проактивна оптимізація

Внутрішня перелінковка	Графові алгоритми + NLP	NetworkX, Python	Висока	Покращення структури посилань
------------------------	-------------------------	------------------	--------	-------------------------------

Для вибору оптимальної ШІ-технології під конкретні задачі дослідження необхідно розглянути критерії, що відповідають специфіці проекту fionahome.com.ua. Серед ключових критеріїв виокремлюються: підтримка української мови, якість генерованого контенту в текстильній ніші, вартість API-запитів при масштабі каталогу магазину, простота інтеграції з OpenCart та можливість тонкого налаштування промптів під конкретний стиль спілкування бренду.

Аналіз наявних LLM-рішень з позицій зазначених критеріїв показує наступне. Модель GPT-4o від OpenAI демонструє найвищу якість генерації тексту українською мовою серед комерційно доступних рішень, підтримує детальні системні промпти та має стабільний API з прозорою ціновою моделлю. Claude від Anthropic поступається GPT-4o у якості генерації українськомовного контенту, однак відрізняється більш передбачуваним дотриманням формату і структури відповіді. Google Gemini забезпечує нативну інтеграцію з екосистемою Google, що може бути корисним для роботи з Google Search Console, але поступається GPT-4o в якості нішевого текстильного контенту [15].

Окремо слід розглянути спеціалізовані SEO-платформи, що вже інтегрують LLM: Surfer SEO, Jasper, Copy.ai та SEO.ai. Ці рішення пропонують готовий інтерфейс для генерації SEO-контенту, проте не надають прямих механізмів програмної інтеграції з CMS OpenCart і не підтримують пакетну обробку товарного каталогу в автоматизованому режимі. Їх застосування

доцільне для ручного покращення окремих пріоритетних сторінок, але не для масштабної оптимізації всього асортименту (табл. 1.4).

Таблиця 1.4 – Порівняльна характеристика LLM для генерації SEO-контенту українською мовою

Параметр	GPT-4o (OpenAI)	Claude 3.5 (Anthropic)	Gemini 1.5 Pro (Google)	LLaMA 3 (Meta, self-hosted)
Якість укр. тексту	Висока	Висока	Середня	Середня
Підтримка системних промптів	Так	Так	Так	Так
Пакетна обробка	Batch API	Так	Так	Так
Вартість (на 1М токенів вх./вих.)	\$2.5/\$10	\$3/\$15	\$1.25/\$5	Безкоштовно (CAPEX)
Максимальна довжина контексту	128k токенів	200k токенів	1М токенів	128k токенів
Якість у ніші текстилю	Висока	Висока	Середня	Середня
Простота API-інтеграції	Висока	Висока	Середня	Середня

На основі аналізу, зведеного у табл. 1.4, для реалізації задач генерації SEO-контенту в рамках даного дослідження обґрунтовано вибір моделі GPT-4o через OpenAI API як основного інструменту генерації описів товарів і мета-тегів, а також моделі sentence-transformers (зокрема, multilingual-e5-large) для векторної семантичної кластеризації пошукових запитів.

Важливим аспектом роботи з LLM у контексті SEO є проектування ефективних промптів (prompt engineering). Якість генерованого контенту безпосередньо залежить від того, наскільки чітко сформульоване завдання для моделі. Для генерації описів товарів текстильної ніші ефективний промпт повинен містити: чітку вказівку на стиль і тон матеріалу, перелік атрибутів товару, які необхідно органічно включити в опис, цільові ключові слова для

включення без надмірної щільності, обмеження на довжину тексту та вказівку уникати певних шаблонних фраз.

Узагальнюючи аналіз ШІ-технологій, релевантних задачам SEO-оптимізації, можна сформулювати принципи вибору технологічного стека для дослідження. По-перше, пріоритет надається API-сервісам з підтвердженою якістю генерації україномовного контенту, оскільки цільова аудиторія магазину fionahome.com.ua є переважно україномовною. По-друге, обраний стек має забезпечувати можливість програмної пакетної обробки товарного каталогу, що є технічною вимогою для масштабованості рішення. По-третє, інтеграція з OpenCart реалізується через прямий доступ до БД за допомогою скриптів Python, що дозволяє уникнути залежності від API CMS і забезпечує найвищу гнучкість у налаштуванні логіки обробки. Зазначені принципи і обгрунтовані вибори технологій будуть покладені в основу практичної реалізації, що описується в наступних розділах роботи.

РОЗДІЛ 2. ОПИС МОДЕЛІ ТА МЕТОДІВ РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ Е-COMMERCE-ПЛАТФОРМИ

2.1 Розробка концептуальної моделі системи

Метою розробки інформаційної системи є автоматизація процесів SEO-оптимізації інтернет-магазину домашнього текстилю шляхом застосування технологій штучного інтелекту для генерації контенту, аналізу товарних даних та формування метаданих. Розроблена система інтегрується з платформою OpenCart та забезпечує автоматичне створення SEO-описів, метатегів, альтернативних описів зображень, а також виконує семантичне групування товарів для підвищення ефективності пошукової оптимізації.

На етапі проектування було сформовано концептуальну модель системи, яка визначає основні компоненти програмного рішення, взаємозв'язки між ними та потоки даних. Узагальнена архітектура розробленої системи наведена на рис. 2.1.

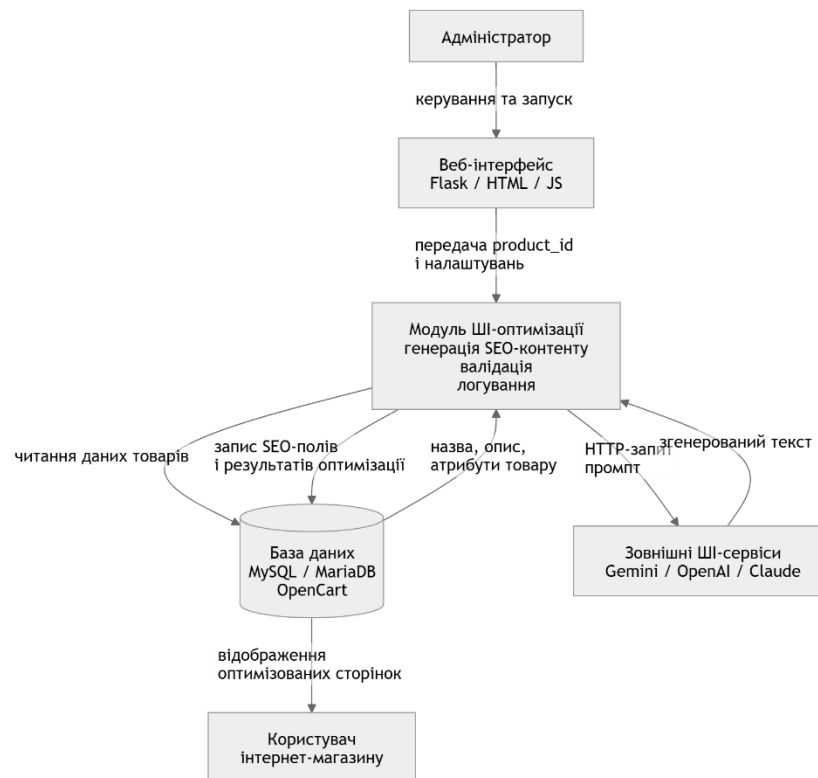


Рис. 2.1. Узагальнена архітектура системи

Згідно з наведеною схемою, центральним елементом системи виступає модуль ШІ-оптимізації, який забезпечує виконання основної бізнес-логіки. Даний модуль взаємодіє з веб-інтерфейсом адміністратора, базою даних інтернет-магазину та зовнішніми сервісами штучного інтелекту. Адміністратор через веб-інтерфейс здійснює вибір товарів або категорій для оптимізації, налаштовує параметри генерації контенту та запускає процес обробки. Після отримання відповідних даних із бази даних OpenCart модуль формує запити до моделей штучного інтелекту, отримує результати генерації та зберігає оновлені SEO-поля у базі даних. Особливістю запропонованого рішення є використання декількох постачальників моделей штучного інтелекту. Це дозволяє забезпечити гнучкість налаштування системи, зменшити залежність від одного сервісу та підвищити надійність роботи

програмного забезпечення. У реалізованій системі підтримується використання моделей OpenAI, Gemini та Claude. Крім генерації текстового контенту, модуль ШІ-оптимізації виконує аналіз зображень товарів, семантичну кластеризацію контенту та перевірку якості сформованих результатів перед їх записом до бази даних.

Для більш детального відображення структури програмного рішення була побудована компонентна діаграма архітектури системи, наведена на рис. 2.2. Дана діаграма демонструє логічне розділення системи на окремі функціональні модулі та відображає взаємозв'язки між ними.

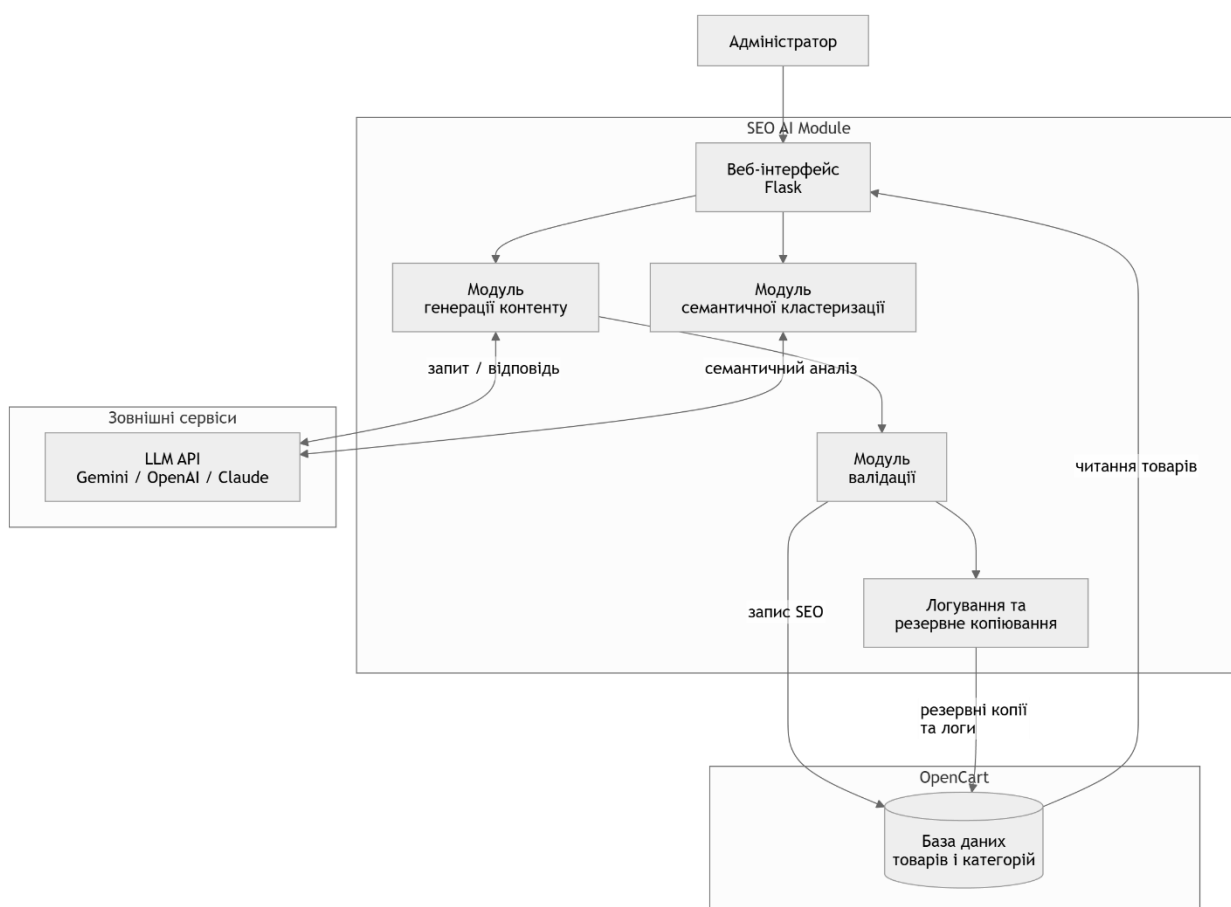


Рис. 2.2. Компонентна діаграма архітектури системи

Як видно з рис. 2.2, програмне рішення складається з декількох взаємопов'язаних компонентів. Основним інтерфейсним рівнем виступає веб-додаток, реалізований за допомогою фреймворку Flask. Через нього адміністратор здійснює взаємодію із системою та керує процесами SEO-оптимізації. Логіка обробки даних реалізована у спеціалізованих модулях ProductProcessor та CategoryProcessor, які координують процес отримання інформації з бази даних, формування запитів до моделей штучного інтелекту та збереження результатів.

До складу системи також входять модулі генерації контенту, аналізу зображень та семантичної кластеризації. Модуль генерації контенту відповідає за створення SEO-описів товарів, метатегів та інших текстових елементів сторінок. Модуль аналізу зображень забезпечує автоматичне формування альтернативних текстів для зображень товарів, що позитивно впливає на показники пошукової оптимізації та доступності веб-ресурсу. Модуль семантичної кластеризації виконує групування товарів за змістовною близькістю та може використовуватися для покращення структури каталогу та оптимізації пошукових запитів. Важливим елементом архітектури є модуль валідації результатів, який здійснює перевірку згенерованого контенту перед його записом до бази даних. Це дозволяє контролювати якість сформованих SEO-матеріалів та запобігати появі помилок у наповненні каталогу товарів.

Після визначення основних компонентів системи було проаналізовано функціональні можливості програмного рішення з точки зору користувача. Для цього була побудована діаграма варіантів використання, наведена на рис. 2.3.

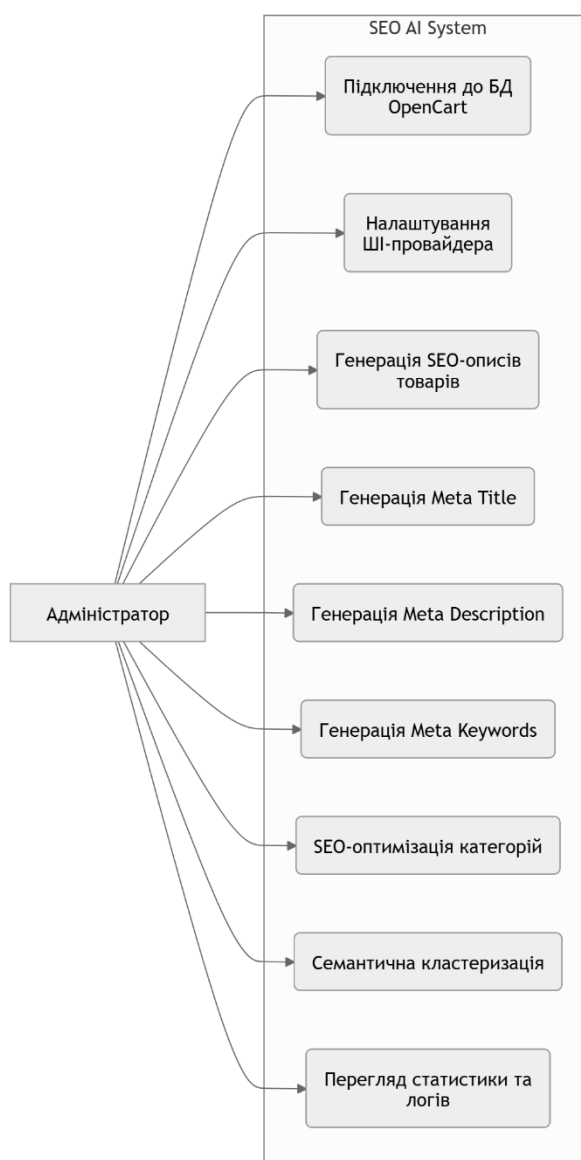


Рис. 2.3. Use-case діаграма системи

Діаграма варіантів використання демонструє перелік основних функцій, доступних адміністратору системи. Серед них можна виділити підключення до бази даних, налаштування параметрів роботи моделей штучного інтелекту,

генерацію SEO-описів товарів, створення метатегів, виконання семантичної кластеризації, аналіз зображень та перегляд статистичної інформації щодо результатів оптимізації. Таким чином, адміністратор отримує єдину точку керування всіма процесами SEO-оптимізації контенту інтернет-магазину.

Для опису динамічної поведінки системи та послідовності взаємодії між її компонентами була побудована діаграма послідовності, представлена на рис. 2.4.

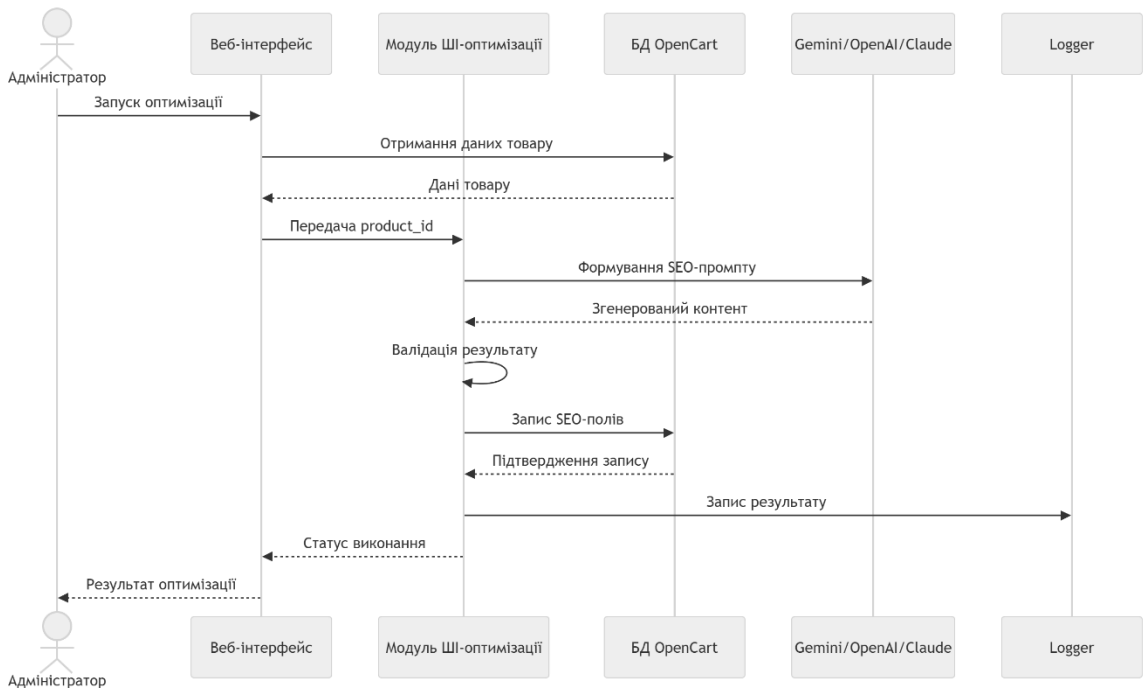


Рис. 2.4. Діаграма послідовності.

Відповідно до діаграми послідовності, процес оптимізації починається з ініціації відповідної операції адміністратором через веб-інтерфейс системи. Після отримання запиту веб-додаток передає його до модуля обробки товарів або категорій. Далі виконується отримання необхідних даних із бази даних та формування запиту до вибраної моделі штучного інтелекту. Згенерований контент проходить перевірку модулем валідації, після чого результати

записуються до бази даних і стають доступними для відображення на сторінках інтернет-магазину. Такий підхід забезпечує автоматизацію процесів створення SEO-контенту та мінімізує необхідність ручного втручання адміністратора.

Запропонована концептуальна модель дозволила сформувати чітку структуру програмного рішення та визначити взаємодію між його складовими. Побудовані діаграми відображають як статичну структуру системи, так і процеси її функціонування. Отримані результати стали основою для подальшого проєктування бази даних, розробки програмної архітектури та реалізації механізмів інтеграції із сервісами штучного інтелекту.

2.2 ER-модель та нормалізація відношень бази даних

База даних платформи OpenCart, на якій розгорнутий магазин fionahome.com.ua, реалізована засобами СКБД MySQL і є реляційною базою даних з розвиненою структурою таблиць. Оскільки підсистема ШІ-оптимізації контенту не вводить нових сутностей у предметну область e-commerce, а лише доповнює існуючі записи згенерованим контентом, аналіз ER-моделі зосереджується на таблицях, безпосередньо задіяних у процесі оптимізації. Поряд з цим, для забезпечення логування та управління процесом оптимізації, проєктується розширення схеми бази даних двома додатковими таблицями, які не втручаються в основну структуру OpenCart і можуть бути видалені без порушення цілісності операційної системи.

Структура бази даних у частині, релевантній для ШІ-оптимізації, включає кілька ключових таблиць.

Таблиця `fdb_product` містить основні атрибути товару: ідентифікатор (`product_id`), модель (`model`), артикул (`sku`), ціну (`price`), кількість на складі (`quantity`), ідентифікатор виробника (`manufacturer_id`) та низку технічних полів управління, серед яких особливе значення має поле `status`, що визначає, чи є товар активним і чи варто його включати до оптимізації.

Таблиця `fdb_product_description` є мовозалежним розширенням таблиці товарів і зберігає локалізовані поля: назву (`name`), мета-тег `title` (`meta_title`), мета-тег `description` (`meta_description`), мета-ключові слова (`meta_keyword`), а також текстовий опис (`description`) та теги (`tag`). Зв'язок між таблицями здійснюється через поле `product_id`; поле `language_id` ідентифікує мовну версію запису. У магазині `fionahome.com.ua` українська мова має ідентифікатор `language_id = 5`, що враховано у всіх SQL-запитах системи.

Таблиця `fdb_category` зберігає ідентифікатор категорії (`category_id`), ідентифікатор батьківської категорії (`parent_id`), порядок сортування (`sort_order`) та прапорець статусу (`status`).

Таблиця `fdb_category_description`, аналогічно до товарів, містить локалізовані поля категорії: `name`, `meta_title`, `meta_description`, `meta_keyword` та `description`.

Таблиця `fdb_product_to_category` забезпечує зв'язок багато-до-багатьох між товарами та категоріями, оскільки один товар може належати до кількох категорій, а одна категорія містити багато товарів.

Таблиця `fdb_manufacturer` зберігає назви виробників (`name`), що використовуються як один із атрибутів при формуванні промптів для генерації контенту.

Таблиця `fdb_product_attribute` пов'язує товари з їхніми характеристиками через проміжні таблиці атрибутів. Разом із таблицею `fdb_attribute_description`

вона забезпечує зберігання значень таких атрибутів, як тканина, склад, розмір, колір, щільність тощо, що є ключовими вхідними даними для генерації унікальних SEO-описів.

Таблиця 2.1 – Структура таблиці fdb_product_description (ключові поля для SEO)

Поле	Тип даних	Обмеження	Призначення
product_id	int(11)	NOT NULL, FOREIGN KEY	Зв'язок з fdb_product
language_id	int(11)	NOT NULL, FOREIGN KEY	Мовна версія (5 – українська)
name	varchar(255)	NOT NULL	Назва товару
description	mediumtext	–	Повний опис товару (генерується III)
meta_title	varchar(255)	–	Мета-тег title (генерується III)
meta_description	varchar(255)	–	Мета-тег description (генерується III)
meta_keyword	mediumtext	–	Мета-ключові слова (генеруються III)

Для забезпечення роботи III-модуля схему бази даних розширено двома проєктованими таблицями, які створюються автоматично при першому запуску системи.

Таблиця ai_optimization_log фіксує кожну операцію оптимізації із зазначенням ідентифікатора товару або категорії (entity_id), типу сутності (entity_type), типу операції (operation_type), часу виконання (executed_at), статусу (status), кількості використаних токенів API (tokens_used), версії використаної моделі (model_version) та MD5-хешу згенерованого контенту (content_hash) для уникнення дублювання.

Таблиця ai_seo_backup зберігає резервні копії оригінальних SEO-полів перед перезаписом, що забезпечує можливість відкату змін у разі незадовільної якості генерації.

Таблиця 2.2 – Структура проекрованої таблиці ai_optimization_log

Поле	Тип даних	Обмеження	Призначення
log_id	int(11)	PRIMARY KEY, AUTO_INCREMENT	Унікальний ідентифікатор запису
entity_type	varchar(20)	NOT NULL	Тип оптимізованої сутності (product/category)
entity_id	int(11)	NOT NULL	ID товару або категорії
operation_type	varchar(50)	NOT NULL	Тип операції (full_seo, description, title тощо)
executed_at	datetime	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Час виконання операції
status	varchar(20)	NOT NULL	Результат (success, error, skipped)
tokens_used	int(11)	DEFAULT NULL	Кількість використаних токенів API
model_version	varchar(50)	DEFAULT NULL	Версія використаної ШІ-моделі
content_hash	varchar(64)	DEFAULT NULL	MD5-хеш згенерованого контенту

Таблиця 2.3 – Структура проекрованої таблиці ai_seo_backup

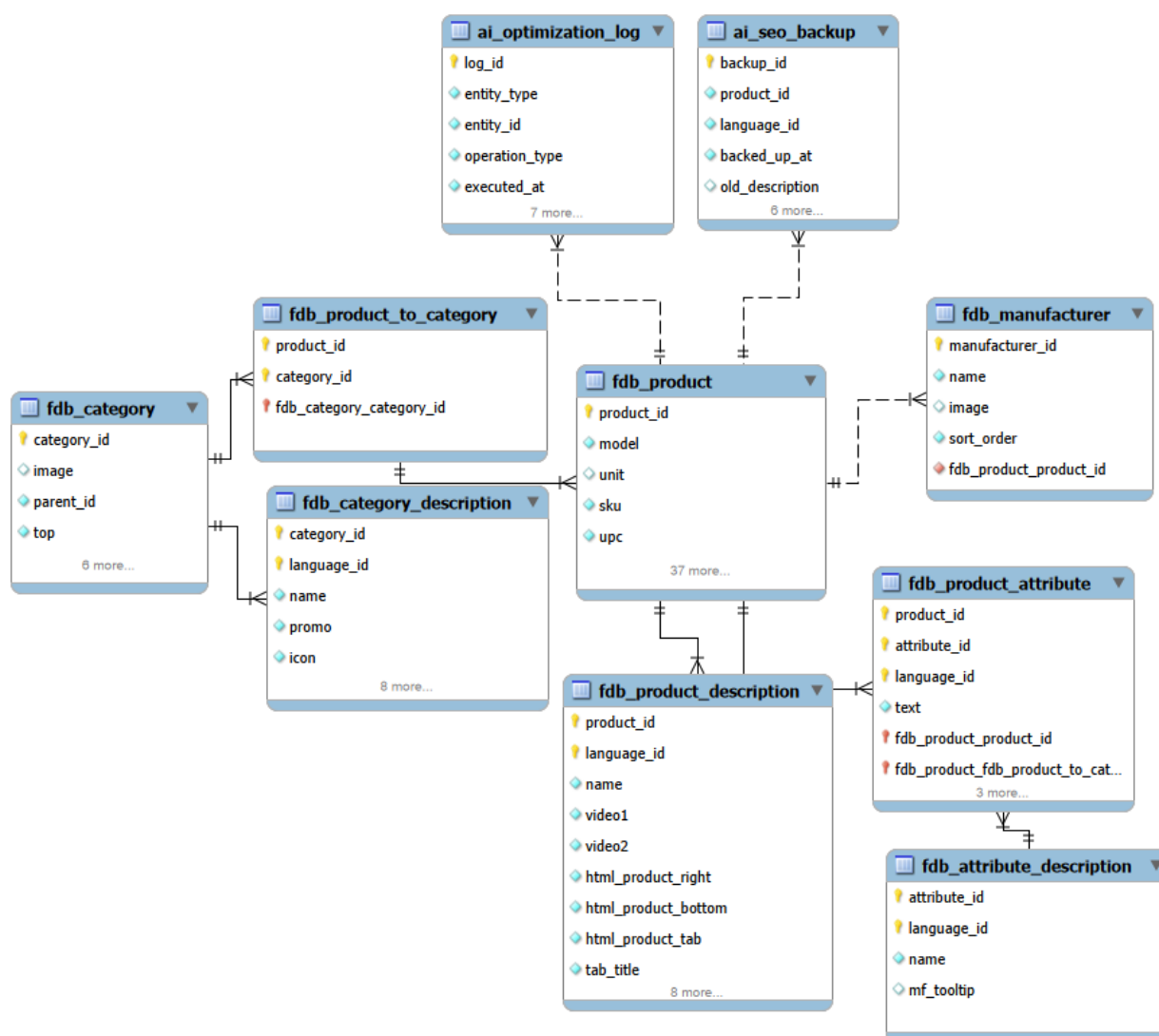
Поле	Тип даних	Обмеження	Призначення
------	-----------	-----------	-------------

backup_id	int(11)	PRIMARY KEY, AUTO_INCREMENT	Унікальний ідентифікатор резервної копії
product_id	int(11)	NOT NULL	ID товару
language_id	int(11)	NOT NULL, DEFAULT 5	Мовна версія
backed_up_at	datetime	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Час створення резервної копії
old_description	mediumtext	–	Оригінальний опис до перезапису
old_meta_title	varchar(255)	–	Оригінальний meta_title
old_meta_desc	varchar(255)	–	Оригінальний meta_description
old_meta_kw	mediumtext	–	Оригінальні meta_keyword

ER-модель, що описує взаємозв'язки між таблицями у контексті ШІ-оптимізації, представлена на рисунку 2.5. Центральне місце в моделі займають таблиці `fdb_product_description` та `fdb_category_description`, оскільки саме вони містять поля, що підлягають оптимізації. До них приєднуються допоміжні таблиці для отримання повного контексту про товар: `fdb_product` (через `product_id`), `fdb_manufacturer` (через `manufacturer_id`), `fdb_product_to_category` та `fdb_category_description` (для визначення категорії), а також

fdb_product_attribute у парі з fdb_attribute_description (для отримання атрибутів у вигляді пар «назва-значення»).

Рис. 2.5 ER-модель бази даних в частині, задіяній у ШІ-оптимізації контенту



Нормалізація відношень бази даних OpenCart в частині товарних і категорійних таблиць відповідає третій нормальній формі (3NF). Зокрема, відокремлення локалізованих текстових полів у таблиці

`fdb_product_description` і `fdb_category_description` є коректним рішенням, що відповідає вимогам другої нормальної форми (2NF): атрибути `name`, `description`, `meta_title` залежать від складеного ключа (`product_id`, `language_id`), а не лише від `product_id`, оскільки один товар може мати різні назви та описи для різних мов. Аналогічно, атрибути виробника зберігаються у таблиці `fdb_manufacturer`, що усуває транзитивні залежності (наприклад, назва виробника залежить від `manufacturer_id`, а не від `product_id`) і забезпечує відповідність 3NF. Проектовані таблиці також спроектовані у відповідності до 3NF. У таблиці `ai_optimization_log` всі неключові атрибути функціонально залежать лише від первинного ключа `log_id`; між неключовими атрибутами транзитивних залежностей не виявлено, оскільки `tokens_used` та `model_version` визначаються конкретним викликом API, а не статусом або типом операції. Таблиця `ai_seo_backup` містить лише атрибути, що безпосередньо описують резервну копію; зв'язок з товаром встановлюється через `product_id`, який не є зовнішнім ключем (що є свідомим рішенням для уникнення каскадних видалень). Таблиця `fdb_product_description` має первинний ключ на парі (`product_id`, `language_id`), що забезпечує ефективну вибірку записів за ідентифікатором товару та мовою. Додатково створено індекс `idx_name` для прискорення пошуку товарів за назвою. Для таблиці `ai_optimization_log` створено індекс `idx_entity` на парі (`entity_type`, `entity_id`), який прискорює запити на кшталт «перевірити, чи вже оптимізовано цей товар» — така перевірка виконується перед обробкою кожного товару, щоб уникнути повторної генерації. Нижче наведено SQL-скрипти створення проєктованих таблиць розширення, які автоматично виконуються при першому запуску веб-інтерфейсу або CLI-команди `python main.py setup-db`:

-- Таблиця логування операцій оптимізації

```

CREATE TABLE IF NOT EXISTS `ai_optimization_log` (
  `log_id`      INT(11)   NOT NULL AUTO_INCREMENT,
  `entity_type` VARCHAR(20) NOT NULL,
  `entity_id`   INT(11)   NOT NULL,
  `operation_type` VARCHAR(50) NOT NULL DEFAULT 'full_seo',
  `executed_at` DATETIME   NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `status`      VARCHAR(20) NOT NULL,
  `tokens_used` INT(11)   DEFAULT NULL,
  `model_version` VARCHAR(50) DEFAULT NULL,
  `content_hash` VARCHAR(64) DEFAULT NULL,
  PRIMARY KEY (`log_id`),
  KEY `idx_entity` (`entity_type`, `entity_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

```

-- Таблиця резервних копій SEO-полів

```

CREATE TABLE IF NOT EXISTS `ai_seo_backup` (
  `backup_id`      INT(11)   NOT NULL AUTO_INCREMENT,
  `product_id`     INT(11)   NOT NULL,
  `language_id`    INT(11)   NOT NULL DEFAULT 5,
  `backed_up_at`   DATETIME   NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `old_description` MEDIUMTEXT,
  `old_meta_title` VARCHAR(255) DEFAULT NULL,
  `old_meta_desc`  VARCHAR(255) DEFAULT NULL,
  `old_meta_kw`    MEDIUMTEXT,
  PRIMARY KEY (`backup_id`),
  KEY `idx_product` (`product_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

```

Важливим аспектом роботи з базою даних OpenCart є те, що операційна підсистема (CMS) і ШІ-модуль звертаються до одного сервера MySQL, що потенційно може спричиняти конкуренцію за ресурси при тривалих пакетних операціях. Для мінімізації впливу на продуктивність основного сайту в системі передбачено кілька механізмів. По-перше, запуск ШІ-скриптів може бути налаштований на нічний час через cron-завдання (наприклад, `python main.py products --limit 200`). По-друге, модуль доступу до даних (DatabaseConnector) використовує автоматичне керування транзакціями з фіксацією після кожної операції, що дозволяє уникнути тривалого блокування таблиць. По-третє, операції читання даних для формування промптів виконуються як прості SELECT-запити без блокувань, а операції запису оновлюють лише обрані рядки без втручання в сусідні записи.

Таблиця 2.5 – Типові SQL-запити ШІ-модуля для роботи з базою даних

Операція	SQL-запит (спрощено)	Коментар
Читання даних товару	SELECT p.product_id, pd.name, pd.description, cd.name AS category, m.name AS manufacturer FROM fdb_product p JOIN fdb_product_description pd ON p.product_id=pd.product_id JOIN fdb_manufacturer m ... WHERE p.product_id = %s AND pd.language_id=5	Отримання вхідних даних для формування промпту
Отримання атрибутів	SELECT ad.name AS attr_name, pa.text AS attr_value FROM fdb_product_attribute pa JOIN fdb_attribute_description ad ON pa.attribute_id=ad.attribute_id WHERE pa.product_id=%s AND pa.language_id=5 AND ad.language_id=5	Атрибути у вигляді пар «назва-значення»

Перевірка, чи оптимізовано	SELECT COUNT(*) FROM ai_optimization_log WHERE entity_type='product' AND entity_id=%s AND operation_type='full_seo' AND status='success'	Уникнення повторної обробки товарів
Резервне копіювання	INSERT INTO ai_seo_backup (product_id, language_id, old_description, old_meta_title, old_meta_desc) SELECT product_id, language_id, description, meta_title, meta_description FROM fdb_product_description WHERE product_id=%s AND language_id=5	Збереження оригінальних SEO-полів перед перезаписом
Запис генерованого опису	UPDATE fdb_product_description SET description=%s, meta_title=%s, meta_description=%s, meta_keyword=%s WHERE product_id=%s AND language_id=5	Запис згенерованого контенту в базу даних
Запис у лог	INSERT INTO ai_optimization_log (entity_type, entity_id, operation_type, status, tokens_used, model_version, content_hash) VALUES ('product', %s, 'full_seo', 'success', %s, %s, %s)	Фіксація результату оптимізації для подальшого аналізу

Розроблена ER-модель і описана структура бази даних формують основу для реалізації модуля доступу до даних у складі ШІ-підсистеми, деталі якої описано у наступному підрозділі. Спроектоване розширення схеми через дві додаткові таблиці (ai_optimization_log, ai_seo_backup) є ненав'язливим і не порушує цілісності стандартної інсталяції OpenCart. Жодна з цих таблиць не модифікує існуючі структури, а використовує їх лише для читання або оновлення через параметризовані запити, що повністю відповідає принципу розділення відповідальності та забезпечує легке видалення модуля без слідів у базі даних.

2.3 Архітектура модуля ШІ-оптимізації контенту

Архітектура модуля ШІ-оптимізації контенту визначає внутрішню структуру програмного забезпечення, що реалізує функціональні вимоги, сформульовані у підрозділі 2.1. Модуль реалізований мовою Python і побудований за принципом послідовних шарів: шар доступу до даних, шар бізнес-логіки (обробки й генерації контенту) та шар інтерфейсу API зовнішніх сервісів. Така пошарова архітектура сприяє модульності й можливості незалежної заміни окремих компонентів [26, с. 93]. Ключовою особливістю реалізації є підтримка трьох альтернативних ШІ-провайдерів (Google Gemini, OpenAI GPT-4o-mini, Anthropic Claude) з єдиним програмним інтерфейсом, що дозволяє змінювати модель генерації без модифікації основної логіки обробки.

Структура файлів проекту ШІ-модуля організована за функціональним принципом. Коренева директорія містить точки входу: `app.py` (веб-інтерфейс на Flask), `main.py` (CLI-інтерфейс), `import_db.py` (імпорт SQL-дампу) та `preview.py` (перегляд результатів до/після). Піддиректорія `config/` зберігає налаштування підключення до бази даних, ключі API та шаблони промптів. Піддиректорія `db/` містить клас підключення до MariaDB (`connector.py`) та параметризовані SQL-запити (`queries.py`).

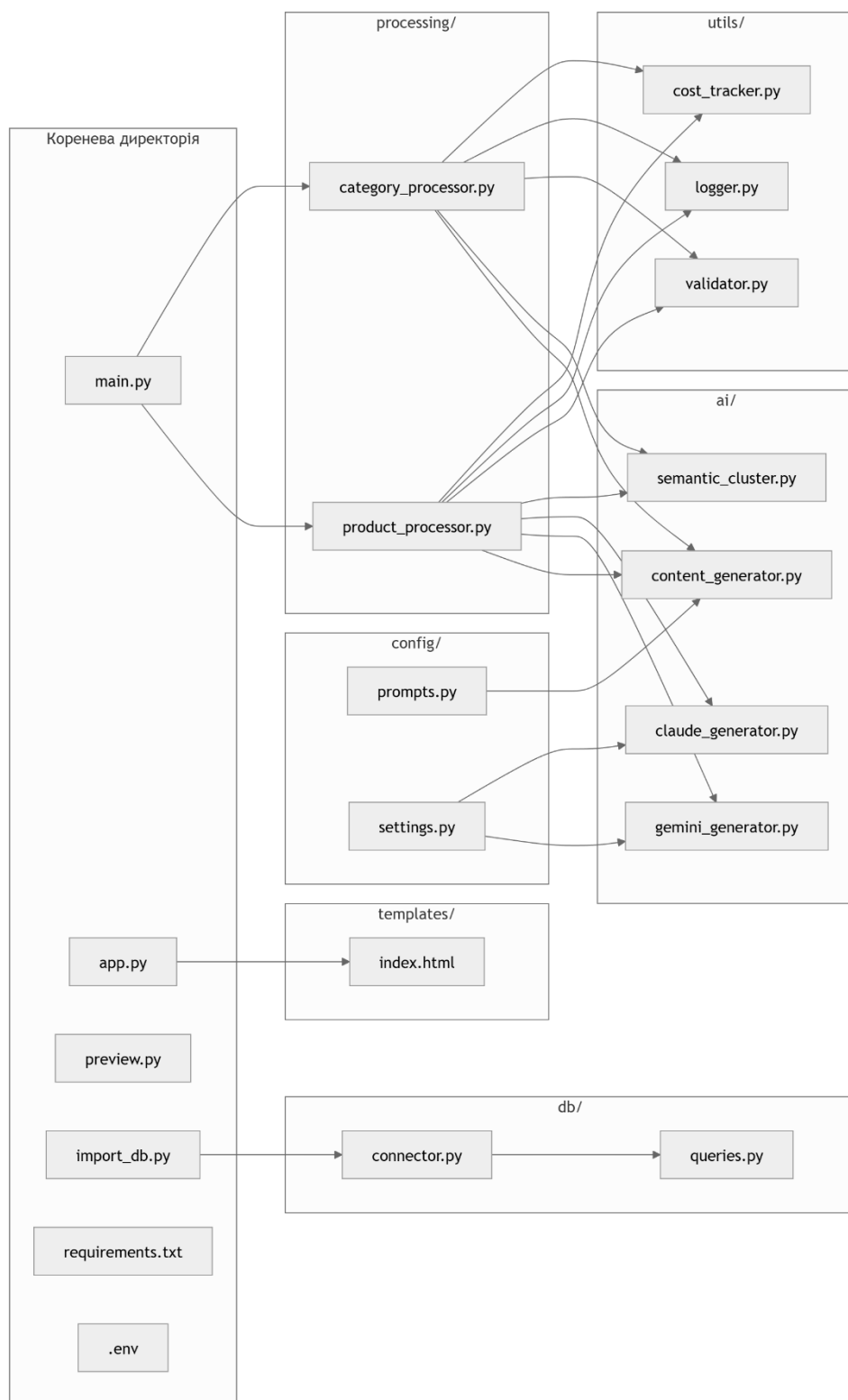


Рис. 2.6 – Структура файлів проекту ШІ-модуля оптимізації контенту

Піддиректорія ai/ реалізує генератори контенту для трьох провайдерів, модуль семантичної кластеризації та модуль аналізу зображень. Піддиректорія processing/ містить оркестратори обробки товарів і категорій. Піддиректорія utils/ забезпечує валідацію контенту, логування та підрахунок витрат на API. Така організація коду відповідає принципу розділення відповідальності (Separation of Concerns) і полегшує підтримку та розширення системи.

Шар доступу до даних представлений двома модулями. Клас DatabaseConnector у файлі connector.py інкапсулює підключення до MariaDB через бібліотеку PyMySQL і надає методи виконання SELECT, INSERT та UPDATE запитів із підтримкою параметризованих запитів (що захищає від SQL-ін'єкцій) та автоматичним повторним підключенням при втраті з'єднання. На відміну від офіційного mysql-connector-python, PyMySQL демонструє кращу сумісність з MariaDB, що особливо важливо для середовища хостингу fionahome.com.ua. Модуль queries.py містить функції вибірки даних товарів з усіма релевантними атрибутами (назва, категорія, виробник, атрибути), функції запису оптимізованого контенту (update_product_seo, update_category_seo) та функції роботи з таблицями розширення (insert_optimization_log, is_already_optimized). Усі SQL-запити є параметризованими, що унеможливорює впровадження шкідливого коду через вхідні параметри.

Ключовим компонентом шару бізнес-логіки є набір генераторів контенту, реалізованих у файлах gemini_generator.py, content_generator.py та claude_generator.py. Кожен з цих класів реалізує однаковий публічний інтерфейс: методи generate_product_description(), generate_meta_title(), generate_meta_description() для товарів; відповідні методи для категорій. Такий підхід, відомий як патерн «Стратегія» (Strategy pattern), дозволяє динамічно

обирати провайдера під час виконання програми без змін у коді оркестратора. Вибір провайдера визначається змінною оточення `AI_PROVIDER`, яка може набувати значень `gemini`, `openai` або `claude`. Нижче наведено ключовий фрагмент коду модуля `gemini_generator.py`, що ілюструє логіку формування промпту і виклику API з використанням нового SDK `google-genai`:

```
from google import genai
from google.genai import types
from config.prompts import PRODUCT_DESCRIPTION_PROMPT
class GeminiContentGenerator:
    def __init__(self, api_key: str, model: str = "gemini-2.0-flash"):
        self.client = genai.Client(api_key=api_key)
        self.model_name = model
    def generate_product_description(self, product_data: dict) -> str:
        prompt = PRODUCT_DESCRIPTION_PROMPT.format(
            product_name=product_data.get("product_name", ""),
            category=product_data.get("category", ""),
            manufacturer=product_data.get("manufacturer", ""),
            attributes=self._fmt_attrs(product_data.get("attributes", {})),
            existing_desc=product_data.get("existing_desc", "")
        )
        response = self.client.models.generate_content(
            model=self.model_name,
            contents=prompt,
            config=types.GenerateContentConfig(
                system_instruction=(
                    "Ти — досвідчений SEO-копірайтер для українського "
                    "інтернет-магазину домашнього текстилю FionaHome. "
                    "Пишеш виключно українською мовою."
                ),
```

```

        max_output_tokens=700,
        temperature=0.7
    )
)
return response.text.strip()
def generate_meta_title(self, product_data: dict) -> str:
    name = product_data.get("product_name", "")
    suffix = " — купити в Україні | FionaHome"
    max_len = 65 - len(suffix)
    keyword = name[:max_len] + "... " if len(name) > max_len else name
    return keyword + suffix

```

Шаблони промптів, що зберігаються у `config/prompts.py`, є одним із найважливіших елементів системи з точки зору якості результату. Для генерації описів товарів текстильної ніші шаблон включає чітку вказівку на стиль (інформативний, але не рекламний), структуру опису (перший абзац – загальна характеристика, другий – матеріал і властивості, третій – переваги та застосування), обмеження на щільність ключових слів та перелік заборонених шаблонних виразів [28]. Промпти для мета-тегів містять вимоги до довжини (140–160 символів для `meta_description`) та обов'язкове включення заклику до дії. Усі промпти передбачають підстановку змінних через метод `format()`, що дозволяє динамічно підставляти назву товару, категорію, атрибути та інші дані. Модуль семантичної кластеризації `semantic_cluster.py` реалізований з використанням бібліотеки `sentence-transformers` і алгоритму кластеризації K-Means. Клас `SemanticClusterer` приймає на вхід список пошукових запитів (отриманих, наприклад, з Google Search Console через відповідний API) і повертає словник, де ключами є назви кластерів (сформовані на основі найчастотніших значущих слів), а значеннями – списки запитів, що входять до

кожного кластера. Визначення оптимальної кількості кластерів здійснюється за допомогою методу ліктя (elbow method), що аналізує залежність між числом кластерів і сумарною внутрішньокластерною дисперсією. Використана модель intfloat/multilingual-e5-large забезпечує високу якість векторних представлень для української мови [29].

Оркестратор обробки товарів product_processor.py об'єднує всі компоненти і реалізує головний цикл пакетної обробки. Клас ProductProcessor ітерується по списку товарів, отриманих з бази даних через функцію get_products_for_optimization(). Для кожного товару спочатку перевіряється, чи не був він вже оптимізований (через запит до таблиці ai_optimization_log), що дозволяє уникнути повторної обробки та надмірних витрат на API. Якщо товар потребує оптимізації, оркестратор послідовно викликає методи генератора для створення опису, мета-тегів. Після генерації отриманий контент проходить валідацію, а потім – за умови, що не ввімкнено режим dry_run – зберігається у базу даних через update_product_seo(). При виникненні помилки API для конкретного товару виконання не зупиняється, а помилка логується і обробка переходить до наступного запису [30].

Таблиця 2.6 – Параметри конфігурації модуля ШІ-оптимізації

Параметр	Значення за замовчуванням	Опис
AI_PROVIDER	gemini	Обраний провайдер (gemini / openai / claude)
GEMINI_MODEL	gemini-2.0-flash	Модель Gemini для генерації тексту

OPENAI_MODEL	gpt-4o-mini	Модель OpenAI для генерації тексту
CLAUDE_MODEL	claude-haiku-4-5-20251001	Модель Claude для генерації тексту
DESC_MIN_WORDS	150	Мінімальна кількість слів в описі
DESC_MAX_WORDS	300	Максимальна кількість слів в описі
TITLE_MAX_CHARS	65	Максимальна довжина meta_title
META_MAX_CHARS	165	Максимальна довжина meta_description
LANGUAGE_ID	5	ID мовної версії (українська)
SLEEP_BETWEEN_REQUESTS	0.5	Затримка між запитами до API (секунди)
TABLE_PREFIX	fdb_	Префікс таблиць OpenCart

Валідатор контенту у файлі `utils/validator.py` реалізує перевірки якості згенерованого контенту перед записом у базу даних. Функція `validate_all()` приймає згенеровані значення полів і повертає словник з результатами валідації для кожного поля. Перевіряється відповідність довжини тексту заданим обмеженням (наприклад, `meta_title` від 40 до 65 символів, `meta_description` від 130 до 165 символів), відсутність небажаних шаблонних виразів зі словника заборонених фраз (який містить такі слова, як «чудовий», «неперевершений», «ідеальний»), а також мінімальна кількість слів для опису (150–300 слів). Валідація не блокує запис контенту у разі незначних порушень, а лише фіксує попередження у лозі, що дозволяє адміністратору згодом переглянути проблемні випадки.

Важливою складовою архітектурного рішення є модуль `cost_tracker.py`, що підраховує реальні витрати на API. Оскільки ціноутворення OpenAI, Gemini та Claude базується на кількості токенів вхідного і вихідного тексту (хоча на момент розробки Gemini надавав безкоштовний доступ), а обсяг каталогу `fionahome.com.ua` може становити сотні товарів, контроль витрат є важливою практичною вимогою. Модуль накопичує статистику використаних токенів у розрізі типів операцій, використовуючи актуальні ціни для кожної моделі, і після завершення кожного пакетного запуску формує звіт із загальними витратами.

Таблиця 2.7 – Прогнозована вартість обробки каталогу магазину `fionahome.com.ua` (для OpenAI)

Операція	Середня кількість токенів	Кількість товарів	Загальна кількість токенів	Вартість (USD)
Генерація описів (вхідні)	350	300	105000	\$0.26
Генерація описів (вихідні)	450	300	135000	\$1.35
Генерація мета-тегів (вхідні)	150	300	45000	\$0.11
Генерація мета-тегів (вихідні)	200	300	60000	\$0.60
Аналіз зображень (вхідні)	500	300	150000	\$0.38
Аналіз зображень (вихідні)	100	300	30000	\$0.30
Разом	—	300	525000	≈\$3.00

Як видно з таблиці 2.7, повна оптимізація каталогу з 300 товарів вимагає орієнтовно 3 долари США витрат на API при використанні OpenAI. При використанні безкоштовного тарифу Gemini витрати відсутні взагалі, що робить систему доступною для тестування та невеликих магазинів. Це підтверджує економічну ефективність запропонованого рішення порівняно з ручною SEO-оптимізацією аналогічного обсягу.

Загальний алгоритм роботи оркестратора пакетної обробки може бути представлений у вигляді блок-схеми, наведеної на рисунку 2.7. Алгоритм починається з ініціалізації підключення до бази даних та створення екземпляра генератора відповідно до обраного провайдера. Далі виконується запит до бази даних для отримання списку товарів, що потребують оптимізації (з урахуванням фільтрації вже оптимізованих, якщо ввімкнено відповідний режим). Для кожного товару з отриманого списку система перевіряє наявність резервної копії поточних SEO-полів (створює її, якщо відсутня), генерує новий контент через API, валідує його та оновлює відповідні поля в таблиці `fdb_product_description`. Кожна операція фіксується у таблиці `ai_optimization_log` із зазначенням статусу, кількості використаних токенів та версії моделі. Після завершення обробки всіх товарів формується підсумковий звіт.

Точка входу в систему реалізована двома способами. Веб-інтерфейс (`app.py`) на базі фреймворку Flask забезпечує зручне адміністрування через браузер: налаштування параметрів підключення до БД, вибір ШІ-провайдера, відбір товарів за фільтрами, запуск обробки з візуальним відстеженням прогресу в реальному часі. Завдання обробки виконуються в асинхронному потоці, що дозволяє інтерфейсу залишатися чуйним під час тривалих операцій. CLI-інтерфейс (`main.py`) реалізований через бібліотеку `argparse` і дозволяє

адміністратору запускати різні режими оптимізації з командного рядка: повну пакетну обробку всього каталогу (`python main.py products`), обробку окремої категорії (`python main.py categories`), оптимізацію конкретного товару за його ID (`python main.py products --id 91594`), семантичну кластеризацію пошукових запитів (`python main.py cluster`) та генерацію звіту про виконані операції

(python main.py stats). Наявність CLI-інтерфейсу спрощує автоматизацію

запусків через cron і полегшує інтеграцію з іншими інструментами.

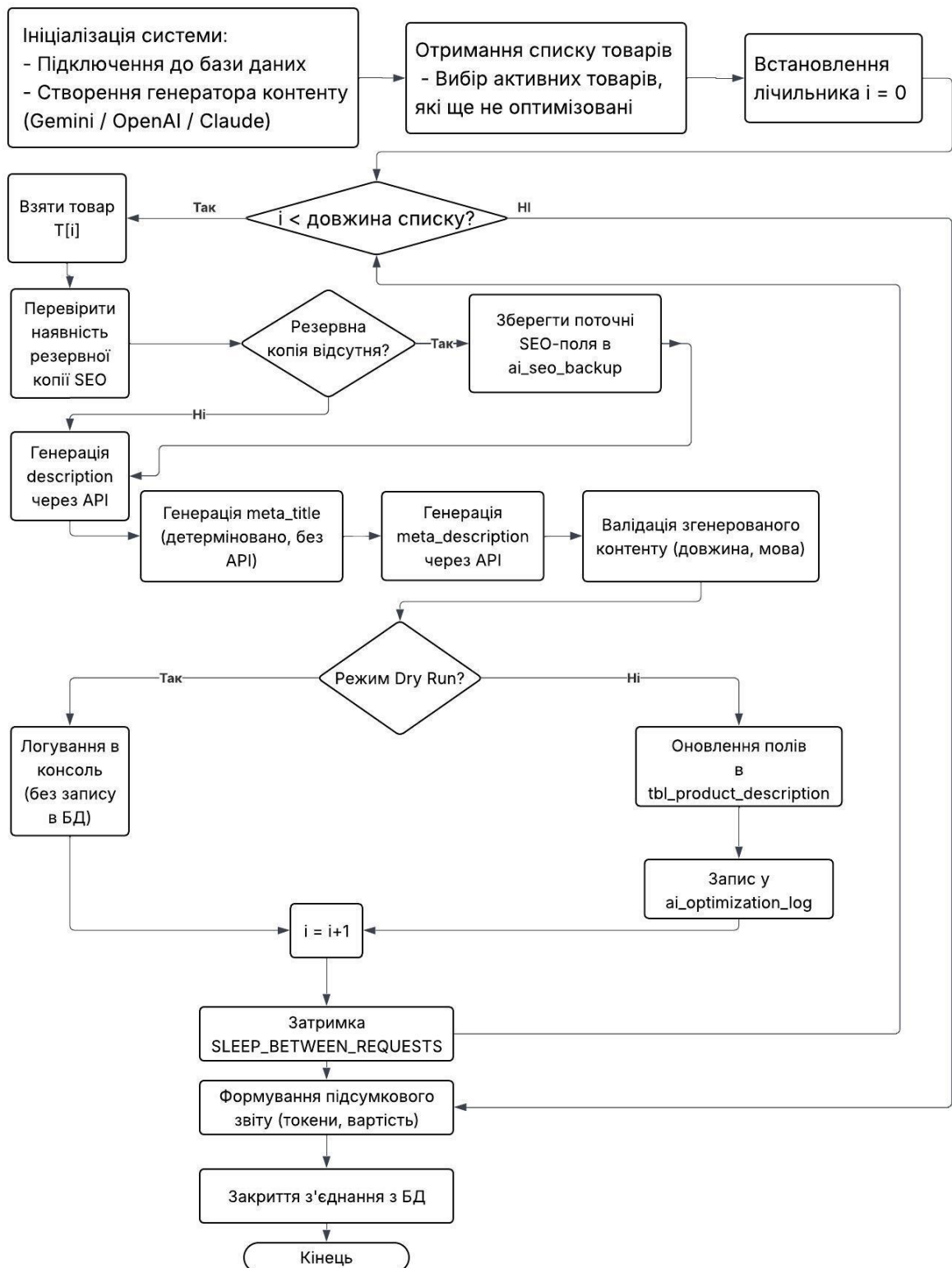


Рис. 2.7. Блок-схема алгоритму пакетної обробки товарів ШІ-модулем

Підсумовуючи, архітектура модуля ШІ-оптимізації контенту побудована за принципами модульності, розширюваності та практичної ефективності. Пошарова організація коду, чіткий поділ відповідальності між класами, підтримка трьох альтернативних ШІ-провайдерів з єдиним інтерфейсом, продумана конфігурація та механізми логування роблять систему придатною не лише для одноразового застосування, а й для постійної підтримки актуальності SEO-контенту при поповненні каталогу магазину fionahome.com.ua новими товарами.

РОЗДІЛ 3. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Backend

Backend-частина системи ШІ-оптимізації контенту реалізована мовою Python версії 3.10+ і складається з кількох функціональних модулів, які забезпечують підключення до бази даних, виконання SQL-запитів, генерацію контенту через зовнішні ШІ-API, валідацію результатів та логування операцій. Веб-серверна частина реалізована на мікрофреймворку Flask, який забезпечує обробку HTTP-запитів, рендеринг HTML-шаблонів та асинхронний запуск тривалих завдань оптимізації.

Для роботи з СКБД MariaDB для OpenCart використовується бібліотека PyMySQL, яка забезпечує кращу сумісність з MariaDB порівняно з офіційним mysql-connector-python. Клас DatabaseConnector інкапсулює логіку підключення, автоматичного перепідключення при втраті з'єднання та виконання параметризованих запитів.

```
# db/connector.py (фрагмент)
import pymysql
import pymysql.cursors
class DatabaseConnector:
    def __init__(self, host, port, database, user, password, charset="utf8mb4"):
        self._config = {
            "host": host, "port": port, "db": database,
            "user": user, "password": password, "charset": charset,
            "cursorclass": pymysql.cursors.DictCursor,
        }
        self._conn = None
```

```

self._connect()
def _connect(self):
    self._conn = pymysql.connect(**self._config)
def _ensure_connected(self):
    try:
        self._conn.ping(reconnect=True)
    except Exception:
        self._connect()
def fetchall(self, query: str, params: tuple = ()) -> list[dict]:
    self._ensure_connected()
    with self._conn.cursor() as cursor:
        cursor.execute(query, params)
        return cursor.fetchall()
def execute(self, query: str, params: tuple = ()) -> int:
    self._ensure_connected()
    with self._conn.cursor() as cursor:
        cursor.execute(query, params)
        self._conn.commit()
        return cursor.rowcount

```

Ключовою особливістю реалізації є використання параметризованих запитів (значення підставляються через %s та передаються окремим кортежем), що повністю унеможлиблює SQL-ін'єкції. Метод `fetchall` повертає результати у вигляді списку словників, де ключами є назви колонок, що зручно для подальшої обробки в бізнес-логіці.

Модуль `queries.py` містить усі SQL-запити для роботи з базою даних OpenCart та розширювальними таблицями. Запити організовані у вигляді окремих функцій, що приймають об'єкт `DatabaseConnector` та параметри. Це

забезпечує розділення відповідальності між рівнем доступу до даних та бізнес-логікою.

```
# db/queries.py (фрагмент)
def get_product_full(db: DatabaseConnector, product_id: int) -> Optional[dict]:
    """Повертає повний профіль товару (дані + атрибути)."""
    P = TABLE_PREFIX
    query = f"""
        SELECT p.product_id, pd.name AS product_name, pd.description AS existing_desc,
               cd.name AS category, COALESCE(m.name, '') AS manufacturer
        FROM {P}product p
        JOIN {P}product_description pd ON p.product_id = pd.product_id AND
pd.language_id = %s
        LEFT JOIN {P}product_to_category ptc ON p.product_id = ptc.product_id
        LEFT JOIN {P}category_description cd ON ptc.category_id = cd.category_id AND
cd.language_id = %s
        LEFT JOIN {P}manufacturer m ON p.manufacturer_id = m.manufacturer_id
        WHERE p.product_id = %s
    """
    row = db.fetchone(query, (LANGUAGE_ID, LANGUAGE_ID, product_id))
    if row:
        row["attributes"] = get_product_attributes(db, product_id)
    return row

def update_product_seo(db: DatabaseConnector, product_id: int,
                      description: str = None, meta_title: str = None,
                      meta_description: str = None) -> int:
    """Оновлює SEO-поля товару."""
    P = TABLE_PREFIX
    fields, params = [], []
    if description is not None:
        fields.append("description = %s")
```

```

        params.append(description)
    if meta_title is not None:
        fields.append("meta_title = %s")
        params.append(meta_title)
    if meta_description is not None:
        fields.append("meta_description = %s")
        params.append(meta_description)
    if not fields:
        return 0
    params += [product_id, LANGUAGE_ID]
    query = f"UPDATE {P}product_description SET {' '.join(fields)} WHERE product_id
= %s AND language_id = %s"
    return db.execute(query, tuple(params))

```

Функція `get_product_full` виконує об'єднання (JOIN) п'яти таблиць для отримання повної інформації про товар, необхідної для формування промпту. Функція `update_product_seo` динамічно будує SQL-запит залежно від того, які поля передано для оновлення, що дозволяє використовувати її для часткового оновлення (наприклад, лише `meta_title`).

Для підтримки трьох альтернативних ШІ-провайдерів реалізовано патерн «Стратегія». Кожен генератор реалізує однаковий інтерфейс, що дозволяє оркестратору не залежати від конкретного провайдера. Вибір провайдера визначається змінною оточення `AI_PROVIDER`. Нижче наведено фрагмент реалізації генератора для Google Gemini з використанням нового SDK `google-genai`:

```

# ai/gemini_generator.py (фрагмент)
from google import genai
from google.genai import types
class GeminiContentGenerator:

```

```

def __init__(self, api_key: str, model: str = "gemini-2.0-flash", tracker=None):
    self.client = genai.Client(api_key=api_key)
    self.model_name = model
    self.tracker = tracker or CostTracker(model=model)

def generate_product_description(self, product_data: dict) -> str:
    prompt = PRODUCT_DESCRIPTION_PROMPT.format(
        product_name=product_data.get("product_name", ""),
        category=product_data.get("category", ""),
        manufacturer=product_data.get("manufacturer", ""),
        attributes=self._fmt_attrs(product_data.get("attributes", {})),
        existing_desc=product_data.get("existing_desc", "")
    )
    response = self.client.models.generate_content(
        model=self.model_name,
        contents=prompt,
        config=types.GenerateContentConfig(
            system_instruction=SYSTEM_MESSAGE,
            max_output_tokens=700,
            temperature=0.7
        )
    )
    # Оновлення лічильника токенів
    usage = response.usage_metadata
    self.tracker.add(usage.prompt_token_count, usage.candidates_token_count)
    return response.text.strip()

def generate_meta_title(self, product_data: dict) -> str:
    """Формує meta_title детерміновано, без виклику API."""
    name = product_data.get("product_name", "")
    suffix = " — купити в Україні | FionaHome"
    max_len = 65 - len(suffix)

```

```
keyword = name[:max_len] + "..." if len(name) > max_len else name
return keyword + suffix
```

Важливо зазначити, що `generate_meta_title` не звертається до зовнішнього API, а формує заголовок за шаблоном. Це зроблено навмисно, оскільки `meta_title` має бути передбачуваним і стабільним, а використання ШІ для його генерації не дає суттєвого покращення якості, але збільшує витрати. Для провайдера OpenAI реалізація виглядає аналогічно, але використовується бібліотека `openai`:

```
# ai/content_generator.py (фрагмент)
import openai

class ContentGenerator:
    def __init__(self, api_key: str, model: str = "gpt-4o-mini", tracker=None):
        self.client = openai.OpenAI(api_key=api_key)
        self.model = model

    def generate_product_description(self, product_data: dict) -> str:
        response = self.client.chat.completions.create(
            model=self.model,
            messages=[
                {"role": "system", "content": SYSTEM_MESSAGE},
                {"role": "user", "content": prompt}
            ],
            max_tokens=700,
            temperature=0.7
        )
        return response.choices[0].message.content.strip()
```

Оркестратор `ProductProcessor` реалізує основний алгоритм пакетної обробки товарів. Він отримує список товарів, для кожного виконує генерацію контенту, валідацію та збереження результатів. Важливою особливістю є

підтримка режиму `dry_run`, при якому контент генерується, але не записується в базу даних, що дозволяє тестувати роботу системи без ризику пошкодити існуючі дані.

`# processing/product_processor.py (фрагмент)`

`class ProductProcessor:`

`def __init__(self, db: DatabaseConnector, limit: int = 0,
 skip_optimized: bool = True, dry_run: bool = False):`

`self.db = db`

`self.dry_run = dry_run`

`self.generator = _get_generator()`

`self.tracker = CostTracker()`

`def run(self) -> dict:`

`products = get_products_for_optimization(self.db, self.limit, self.skip_optimized)`

`for product in products:`

`pid = product["product_id"]`

`try:`

`# Генерація контенту`

`description = self.generator.generate_product_description(product)`

`meta_title = self.generator.generate_meta_title(product)`

`meta_description = self.generator.generate_meta_description(product)`

`# Валідація`

`validation = validate_all(description, meta_title, meta_description)`

`if not self.dry_run:`

`# Резервне копіювання`

`_backup_seo(self.db, pid)`

`# Оновлення SEO-полів`

`update_product_seo(self.db, pid, description, meta_title, meta_description)`

`# Логування`

`insert_optimization_log(self.db, "product", pid, "full_seo",`

```

        "success" if validation.valid else "skipped",
        tokens_used=self.tracker.last_tokens)
except Exception as e:
    logger.error(f"Помилка при обробці товару {pid}: {e}")
    insert_optimization_log(self.db, "product", pid, "full_seo", "error")
return self.stats()

```

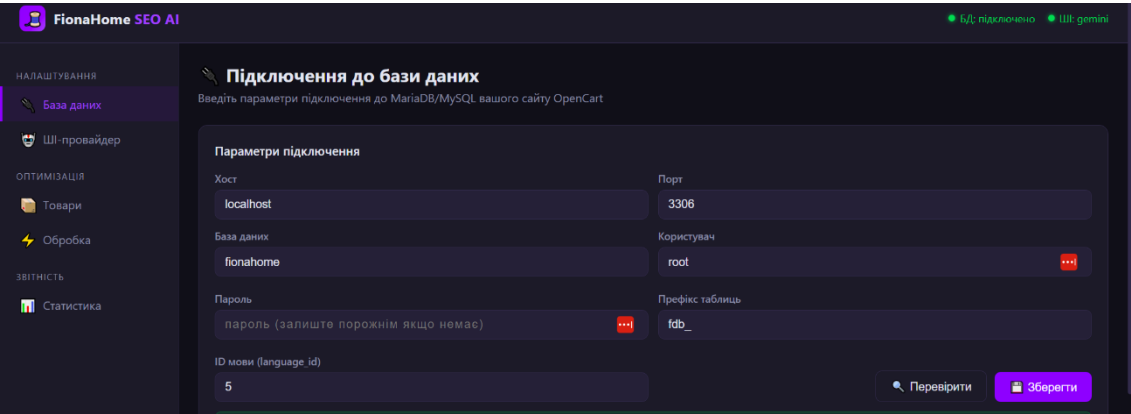
Таким чином, backend-частина системи реалізує всі необхідні компоненти для автоматизованої SEO-оптимізації: від підключення до бази даних та виконання SQL-запитів до генерації контенту через зовнішні ШІ-API, валідації результатів та контролю витрат. Використання патерну «Стратегія» забезпечує легку заміну ШІ-провайдера, а підтримка режиму dry_run дозволяє безпечно тестувати систему перед реальним застосуванням.

3.2 Frontend

Frontend-частина системи реалізована у вигляді веб-інтерфейсу на базі фреймворку Flask та CLI-інтерфейсу командного рядка. Веб-інтерфейс призначений для щоденної роботи контент-менеджера, тоді як CLI орієнтований на автоматизований запуск через cron-завдання. Веб-інтерфейс реалізований у файлі app.py і складається з однієї HTML-сторінки templates/index.html. Сторінка побудована з використанням чистого HTML, CSS та JavaScript без додаткових фреймворків, що забезпечує легкість розгортання та мінімальне навантаження на сервер. Інтерфейс розділений на чотири логічні блоки, які відповідають послідовності дій адміністратора.

Блок 1 – Налаштування бази даних. Містить поля для введення параметрів підключення до MariaDB: хост, порт, назва бази даних, ім'я користувача, пароль, префікс таблиць OpenCart (за замовчуванням fdb_) та ID

мовної версії (для української – 5). Після заповнення полів адміністратор натискає кнопку «Перевірити», система виконує тестове з'єднання та



відображає кількість знайдених товарів. У разі успіху кнопка «Зберегти» фіксує налаштування для поточного сеансу.

Рис. 3.1. Блок 1

Блок 2 – Вибір ШІ-провайдера. Реалізований у вигляді трьох вкладок (табів) для Gemini, OpenAI та Claude. Користувач обирає провайдера, вводить відповідний API-ключ, обирає модель (наприклад, gemini-2.0-flash або gpt-4o-mini) та натискає «Перевірити підключення». Система надсилає тестовий запит до API та відображає зразок згенерованого тексту, підтверджуючи працездатність ключа.

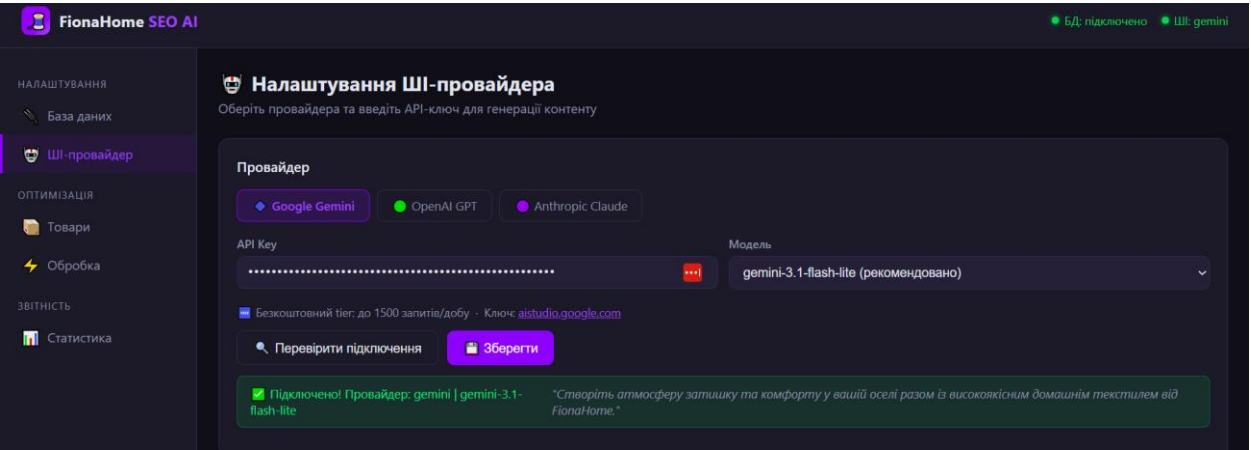
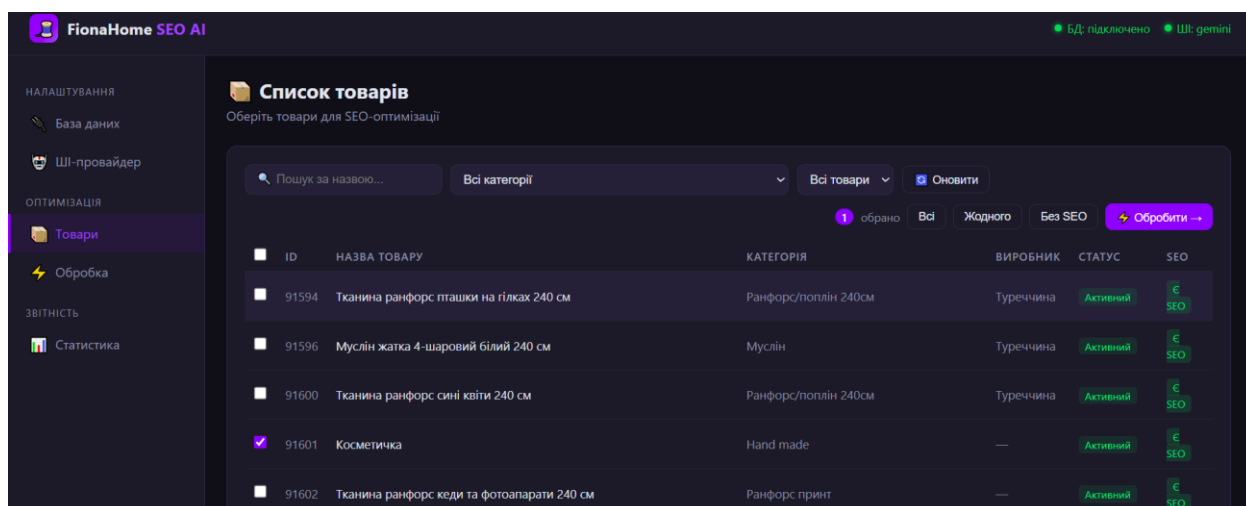


Рис. 3.2. Блок 2

Блок 3 – Вибір товарів. Містить фільтри для пошуку за назвою товару та категорією, а також кнопки «Оновити» (завантаження списку товарів з БД),



«Всі» (виділити всі товари) та «Без SEO» (виділити лише товари без заповненого meta_title). Список товарів відображається у вигляді таблиці з колонками: чекбокс для вибору, ID, назва товару, категорія, ціна, статус (активний/неактивний) та наявність SEO. Після вибору необхідних позицій адміністратор натискає кнопку «Обробити», після чого відображається блок керування обробкою.

Рис. 3.3. Блок 3

Блок 4 – Керування обробкою. Містить чекбокси для вибору полів для генерації (опис, meta_title, meta_description, meta_keyword), перемикач «Dry Run» (тестовий режим без запису в БД), кнопку «Запустити» та область для відображення прогресу. У процесі роботи система оновлює журнал операцій у реальному часі (лог з часовими мітками), а також відображає статистику: кількість оброблених товарів, успішних операцій, помилок та витрачених токенів.

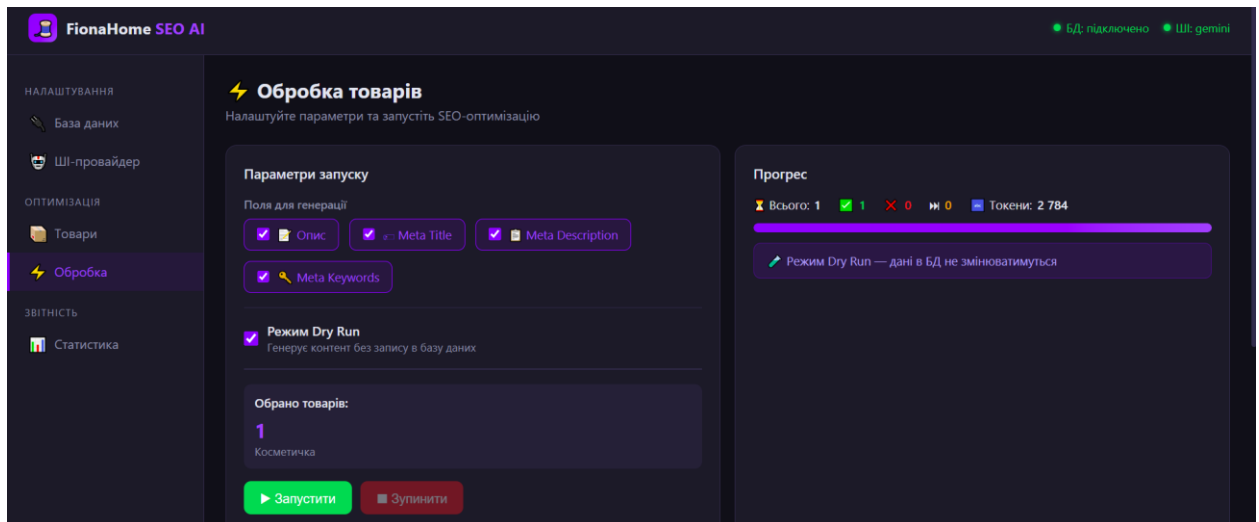


Рис. 3.4. Блок 4

Взаємодія між клієнтською та серверною частинами здійснюється через асинхронні JavaScript-запити (Fetch API). Ключовою особливістю реалізації є асинхронне виконання тривалих завдань. Оскільки генерація контенту для сотні товарів може займати кілька хвилин, запит на запуск обробки не блокується до завершення, а запускає фоновий потік на сервері. Клієнт періодично (кожну секунду) опитує endpoint `/api/process/status` для отримання проміжних результатів – логу операцій, кількості оброблених товарів, витрачених токенів. Це створює ефект роботи в реальному часі без необхідності перезавантажувати сторінку.

Для автоматизації та інтеграції з системою stop розроблено CLI-інтерфейс у файлі `main.py`. Він підтримує кілька команд:

- `python main.py products` – оптимізація всіх товарів;
- `python main.py products --limit 10` – обробка лише перших 10 товарів;
- `python main.py products --id 91594` – оптимізація конкретного товару;
- `python main.py products --dry-run` – тестовий режим без запису в БД;
- `python main.py categories` – оптимізація категорій;
- `python main.py cluster --input queries.txt` – семантична кластеризація запитів;
- `python main.py stats` – перегляд статистики виконаних операцій;
- `python main.py setup-db` – створення допоміжних таблиць.

CLI-інтерфейс використовує бібліотеку `argparse` для розбору аргументів командного рядка. Після запуску система виводить у термінал прогрес обробки, лог операцій та підсумкову статистику (кількість успішних операцій, помилок, витрачених токенів та вартість у доларах США). Завдяки підтримці аргументу `--dry-run` адміністратор може безпечно тестувати нові промпти або моделі без ризику зміни даних у базі.

Для зручності адміністрування також реалізовано два допоміжні скрипти:

- `import_db.py` – імпортує SQL-дамп з адмінки OpenCart у базу даних. Скрипт автоматично створює всі необхідні таблиці (як стандартні OpenCart, так і розширення `ai_optimization_log`, `ai_seo_backup`, `ai_semantic_cluster`), парсить SQL-файл на окремі інструкції та виконує їх. Використання: `python import_db.py fionahome_export.sql`.
- `preview.py` – дозволяє переглянути згенерований контент для одного товару без запису в базу даних. Скрипт виводить порівняння «до»

(поточний контент у БД) та «після» (що згенерує ІІІ), включаючи кількість слів в описі та довжину мета-тегів. Це корисно для налаштування промптів та оцінки якості роботи моделі. Використання: `python preview.py --id 91594`.

Таким чином, frontend-частина системи забезпечує два способи взаємодії: зручний веб-інтерфейс з візуальним відстеженням прогресу для оперативної роботи контент-менеджера та CLI-інтерфейс для автоматизованого нічного запуску через cron. Асинхронна архітектура веб-інтерфейсу дозволяє відстежувати прогрес обробки в реальному часі без блокування інтерфейсу користувача.

3.3 Реалізація ІІІ-модуля SEO-оптимізації

ІІІ-модуль є центральним компонентом системи, який забезпечує генерацію SEO-контенту через зовнішні API великих мовних моделей. Модуль реалізований з підтримкою трьох альтернативних провайдерів (Google Gemini, OpenAI, Anthropic Claude), що дозволяє обирати оптимальне співвідношення ціни та якості. Усі генератори реалізують єдиний інтерфейс, що забезпечує взаємозамінність компонентів без змін у клієнтському коді.

Кожен генератор реалізує уніфіковані публічні методи: `generate_product_description()` – генерує опис товару (180–250 слів); `generate_meta_title()` – формує meta title за шаблоном (без виклику API); `generate_meta_description()` – генерує meta description (140–160 символів); `generate_category_description()` – опис сторінки категорії. Така уніфікація дозволяє оркестратору (ProductProcessor) працювати з будь-яким

провайдером через поліморфний виклик. Вибір конкретного генератора відбувається у фабричній функції на основі змінної оточення `AI_PROVIDER`.

Генератор для Gemini використовує SDK `google-generativeai`. Ключовою особливістю є безкоштовний тариф, що робить цього провайдера ідеальним для тестування. Оскільки безкоштовний тариф обмежує частоту запитів до 15 RPM, у коді реалізовано примусову затримку 4,1 секунди між викликами.

```
class GeminiContentGenerator:
    def __init__(self, api_key: str, model: str = "gemini-2.0-flash"):
        self.client = generativeai.Client(api_key=api_key)
        self.model_name = model

    def generate_product_description(self, product_data: dict) -> str:
        prompt = PRODUCT_DESCRIPTION_PROMPT.format(
            product_name=product_data.get("product_name", ""),
            category=product_data.get("category", ""),
            manufacturer=product_data.get("manufacturer", ""),
            attributes=self._fmt_attrs(product_data.get("attributes", {}))
        )
        response = self.client.models.generate_content(
            model=self.model_name,
            contents=prompt,
            config=types.GenerateContentConfig(
                system_instruction=SYSTEM_MESSAGE,
                max_output_tokens=700
            )
        )
        return response.text.strip()
```

Генератор для OpenAI використовує бібліотеку `openai` (модель `gpt-4o-mini`), а для Claude – бібліотеку `anthropic` (модель `claude-haiku-4.5`). Обидва

генератори мають аналогічну структуру та реалізують той самий набір методів, що й Gemini-генератор. Завдяки уніфікованому інтерфейсу оркестратор може використовувати будь-якого провайдера без змін у коді.

Шаблони промптів зберігаються у файлі `config/prompts.py`. Вони містять чіткі інструкції щодо стилю (інформативний, не рекламний), структури (три абзаци для опису товару), обсягу (180–250 слів для опису, 140–160 символів для `meta description`) та мови (виключно українська). Також промпти містять перелік заборонених шаблонних слів («чудовий», «неперевершений», «ідеальний» тощо).

Модуль `semantic_cluster.py` реалізує кластеризацію пошукових запитів на основі векторних вкладень (`embeddings`). Використовується модель `intfloat/multilingual-e5-large`, яка підтримує українську мову. Алгоритм виконує векторизацію запитів, нормалізацію, визначення оптимальної кількості кластерів методом ліктя (`elbow method`), кластеризацію методом K-Means та автоматичне іменування кластерів на основі найчастотніших слів.

```
class SemanticClusterer:
```

```
    def __init__(self):
```

```
        self.model = SentenceTransformer("intfloat/multilingual-e5-large")
```

```
    def cluster_queries(self, queries: list[str]) -> dict[str, list[str]]:
```

```
        embeddings = self.model.encode(queries)
```

```
        kmeans = KMeans(n_clusters=self._find_optimal_k(embeddings), random_state=42)
```

```
        labels = kmeans.fit_predict(embeddings)
```

```
        # Групування результатів
```

```
        clusters = {}
```

```
        for query, label in zip(queries, labels):
```

```
            clusters.setdefault(f'cluster_{label}', []).append(query)
```

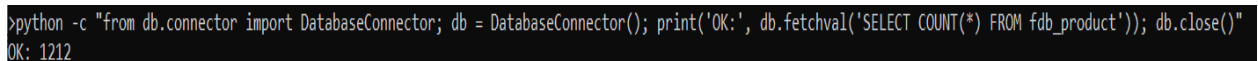
```
return clusters
```

Для підвищення надійності системи реалізовано механізм повторних спроб при тимчасових збоях API. При отриманні помилок 429 (перевищено ліміт запитів) або 503 (сервер тимчасово недоступний) система виконує до трьох повторних спроб з експоненційною затримкою (2, 4, 8 секунд). Це дозволяє зменшити кількість збоїв при пікових навантаженнях на стороні провайдера.

Таким чином, ШІ-модуль реалізує повний цикл генерації SEO-контенту: від формування промпту на основі даних про товар до отримання та валідації згенерованого тексту. Підтримка трьох провайдерів з єдиним інтерфейсом забезпечує гнучкість, а механізми rate limiting та повторних спроб гарантують стабільну роботу системи.

Для підтвердження працездатності розробленого ШІ-модуля було проведено серію тестових запусків на реальних даних інтернет-магазину fionahome.com.ua. Нижче наведено основні сценарії роботи системи з відповідними скріншотами:

1. Приклад підключення до БД. На запит `python -c "from db.connector import DatabaseConnector; db = DatabaseConnector(); print('OK:', db.fetchval('SELECT COUNT(*) FROM fdb_product')); db.close()"` отримуємо відповідь: OK: 1212



```
>python -c "from db.connector import DatabaseConnector; db = DatabaseConnector(); print('OK:', db.fetchval('SELECT COUNT(*) FROM fdb_product')); db.close()"
OK: 1212
```

Рис. 3.5 Підключення до БД

2. Перевіряємо, що модуль правильно читає товари з БД. На запит `python -c "from db.connector import DatabaseConnector; from db.queries import get_products_for_optimization; db = DatabaseConnector(); p =`

`get_products_for_optimization(db, limit=3, skip_optimized=False); [print(f' {x["product_id"]}: {x["product_name"]}') for x in p]; db.close()`" отримуємо відповідь:

91594: Тканина ранфорс пташки на гілках 240 см

91596: Муслін жатка 4-шаровий білий 240 см

91600: Тканина ранфорс сині квіти 240 см

```
>python -c "from db.connector import DatabaseConn
timized=False); [print(f' {x["product_id"]}: {
91594: Тканина ранфорс пташки на гілках 240 см
91596: Муслін жатка 4-шаровий білий 240 см
91600: Тканина ранфорс сині квіти 240 см
```

Рис. 3.6 Перевірка читання з БД

3. Перевіряємо читання категорій. Отримуємо вірну відповідь:

8: Ранфорс принт (163 товарів)

10: Ранфорс принт дитячий (130 товарів)

36: Наволочки (65 товарів)

```
>python -c "from db.connector import Databa
skip_optimized=False); [print(f' {c["cate
8: Ранфорс принт (163 товарів)
10: Ранфорс принт дитячий (130 товарів)
36: Наволочки (65 товарів)
```

Рис. 3.7 Підтвердження читання категорій з БД

4. Генерація ШІ-контенту для одного товару. Режим `--dry-run` дозволяє виконати генерацію SEO-контенту без запису результатів у базу даних. Це корисно для попередньої оцінки якості генерації перед масовим оновленням.

`python main.py products --limit 1 --dry-run`

Результат виконання наведено на рис.3.8:

Результати: ТОВАРИ	
Всього:	1
Успішно:	1
Пропущено:	0
Помилки:	0
Токенів:	2,381
Вартість:	~\$0.0000
Провайдер:	gemini

Рис. 3.8 Результати генерації

5. Попередній перегляд «ДО / ПІСЛЯ» (preview.py). Утиліта preview.py забезпечує наочне порівняння поточного контенту товару зі згенерованим ШІ-контентом. Це дозволяє адміністратору оцінити якість роботи модуля перед внесенням змін.

```
python preview.py --limit 1
```

Результат:

ТОВАР: Тканина ранфорс пташки на гілках 240 см (id=91594) Категорія: Ранфорс/поплін 240см Виробник: Туреччина Атрибути: Країна виробник: Туреччина; Матеріал: Ранфорс; Склад: 100% бавовна; Ширина: 240 см; Щільність: 125 г/м2
— БУЛО (поточний контент у БД) — Meta title (72 симв.): Тканина для постільної білизни Ранфорс 240 см з принтом пташки на гілках Meta desc (87 симв.): Тканина для постільної білизни Ранфорс,100% Бавовна, 240 см, з принтом пташки на гілках Опис (209 сл.): <div style=&quot;font-size:1.08em; line-height:1.55;&quot;> <p> Турецький ранфорс 240 см — це натуральна бавовняна тканина для постільної білизни,&

Meta desc: 87 симв. → 162 симв.

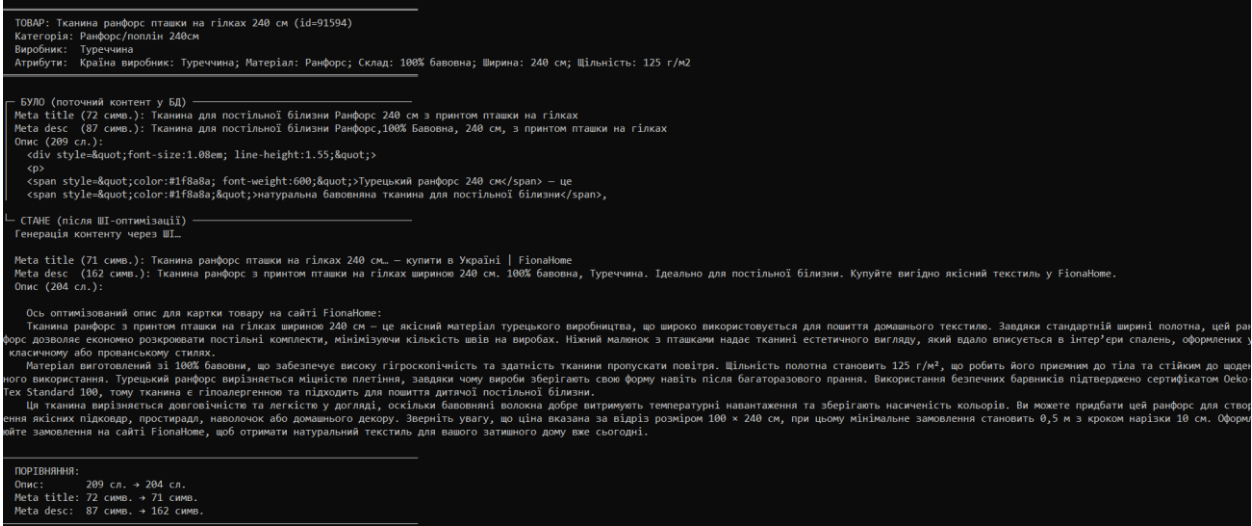


Рис. 3.9 Порівняння ДО / ПІСЛЯ

6. Оптимізація категорій. Аналогічно до товарів, модуль підтримує оптимізацію сторінок категорій. Наступний приклад демонструє обробку двох категорій у режимі --dry-run.

python main.py categories --limit 2 --dry-run

Результат:

2026-05-30 13:19:59 | INFO | processing.category_processor | CategoryProcessor: знайдено 2 категорій для обробки (провайдер: gemini)
2026-05-30 13:19:59 | INFO | processing.category_processor | [1/2] Обробка категорії: Ранфорс принт (id=8, товарів: 163)
2026-05-30 13:20:07 | INFO | processing.category_processor | [2/2] Обробка категорії: Ранфорс принт дитячий (id=10, товарів: 130)
2026-05-30 13:20:19 | INFO | processing.category_processor | [DRY RUN] Категорія id=10 — опис (165 сл.), title (49 симв.), desc (163 симв.)
2026-05-30 13:20:19 | WARNING | utils.cost_tracker | Ціни для моделі 'gemini' невідомі; вартість = 0
2026-05-30 13:20:19 | INFO | utils.cost_tracker | API витрати — Модель: gemini | Викликів: 2 (помилки: 1) | Токени: 567 вх. + 450 вих. = 1017 | Вартість: ~\$0.0000
2026-05-30 13:20:19 | WARNING | utils.cost_tracker | Ціни для моделі 'gemini' невідомі; вартість = 0

Результати: КАТЕГОРІЇ

Всього: 2
 Успішно: 1
 Пропущено: 0
 Помилки: 1
 Токенів: 1,017
 Вартість: ~\$0.0000
 Провайдер: gemini

Результати: КАТЕГОРІЇ	
Всього:	2
Успішно:	1
Пропущено:	0
Помилки:	1
Токенів:	1,017
Вартість:	~\$0.0000
Провайдер:	gemini

Рис. 3.10 Обробка категорій

7. Реальний запис результатів у базу даних. Після перевірки якості генерації в режимі `--dry-run` можна виконати оптимізацію з реальним записом результатів у базу даних.

`python main.py products --limit 5`

Результат:

```
2026-05-30 13:20:46 | INFO | processing.product_processor | ProductProcessor: знайдено 5 товарів для обробки
(провайдер: gemini)
2026-05-30 13:20:46 | INFO | processing.product_processor | [1/5] Обробка товару: Тканина ранфорс пташки
на гілках 240 см (id=91594)
2026-05-30 13:21:06 | INFO | processing.product_processor | Товар id=91594 збережено у БД.
2026-05-30 13:21:10 | INFO | processing.product_processor | [2/5] Обробка товару: Мусліні жатка 4-шаровий
білий 240 см (id=91596)
2026-05-30 13:21:18 | INFO | processing.product_processor | Товар id=91596 збережено у БД.
2026-05-30 13:21:22 | INFO | processing.product_processor | [3/5] Обробка товару: Тканина ранфорс сині квіти
240 см (id=91600)
```

```

2026-05-30 13:21:44 | INFO    | processing.product_processor | Товар id=91600 збережено у БД.
2026-05-30 13:21:49 | INFO    | processing.product_processor | [4/5] Обробка товару: Косметичка (id=91601)
2026-05-30 13:22:09 | INFO    | processing.product_processor | Товар id=91601 збережено у БД.
2026-05-30 13:22:14 | INFO    | processing.product_processor | [5/5] Обробка товару: Тканина ранфорс кеди та
фотоапарати 240 см (id=91602)
2026-05-30 13:22:20 | INFO    | processing.product_processor | Товар id=91602 збережено у БД.
2026-05-30 13:22:20 | WARNING | utils.cost_tracker | Ціни для моделі 'gemini' невідомі; вартість = 0
2026-05-30 13:22:20 | INFO    | utils.cost_tracker | API витрати — Модель: gemini | Викликів: 10 (помилки: 0) |
Токени: 8678 вх. + 2733 вих. = 11411 | Вартість: ~$0.0000
2026-05-30 13:22:20 | WARNING | utils.cost_tracker | Ціни для моделі 'gemini' невідомі; вартість = 0

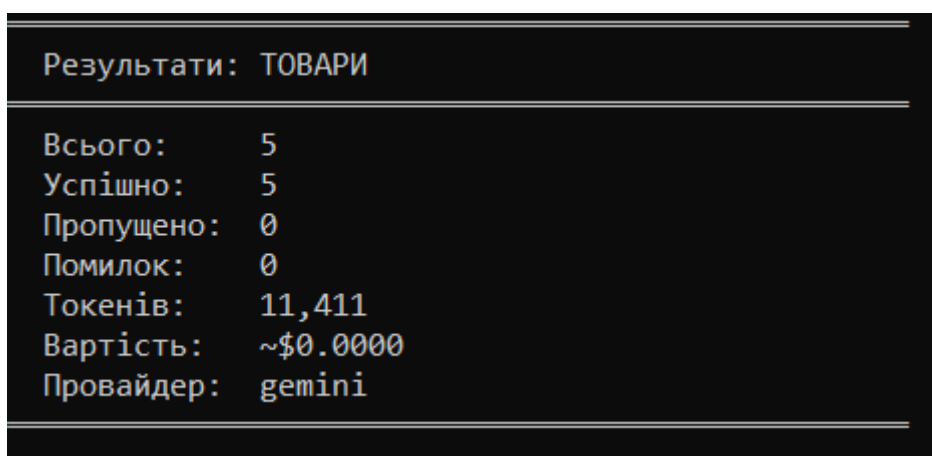
```

Результати: ТОВАРИ

```

Всього:    5
Успішно:   5
Пропущено: 0
Помилки:   0
Токенів:   11,411
Вартість:  ~$0.0000
Провайдер: gemini

```



```

Результати: ТОВАРИ

Всього:    5
Успішно:   5
Пропущено: 0
Помилки:   0
Токенів:   11,411
Вартість:  ~$0.0000
Провайдер: gemini

```

Рис. 3.11 Обробка товарів із збереженням в БД

8. Перевірка записаних даних у базі.

Для підтвердження того, що дані дійсно були записані, можна виконати прямий SQL-запит до бази даних.

```
mysql -u root -p fionahome -e "SELECT product_id, LEFT(meta_title,50) AS title, LEFT(meta_description,60) AS descr FROM fdb_product_description WHERE language_id=5 LIMIT 5;"
```

Результат:

product_id	title	descr
0	Тканина для скатертин ДАК(duck), тефлонова водов?	Ранфорс 240 см
91594	Тканина ранфорс пташки на г?лках 240 см? ? купити	Як?сна тканина ранфорс Пташки на г?лках шириною 240 см в?д т
91596	Мусл?н жатка 4-шаровий б?лий 240 см? ? купити в Ук	Б?лий 4-шаровий мусл?н жатка з Туреччини 100% бавовна ширино
91600	Тканина ранфорс син? кв?ти 240 см ? купити в Украї	Як?сна тканина ранфорс з син?ми кв?тами шириною 240 см в?д т
91601	Косметичка ? купити в Україн? FionaHome	Стильна косметичка hand made з водов?дштовхуючого даку в?д F

Рис. 3.12 Підтвердження запису в БД

9. Статистика виконаних оптимізацій. Команда stats виводить зведену інформацію з таблиці логування ai_optimization_log.

```
python main.py stats
```

entity_type	operation	status	count	tokens
product	full_seo	success	1	0
product	full_seo	skipped	4	0
РАЗОМ			0 токенів	

Рис. 3.13 Приклад статистики роботи системи

Таким чином, було реалізовано SEO-модуль для магазину fionahome. Модуль реалізовано на Python та в цьому підрозділі наведено демонстрацію його основних функцій.

3.4 Запити інформаційної системи

Для роботи III-модуля з базою даних реалізовано набір параметризованих SQL-запитів, згрупованих у файлі db/queries.py. Усі запити використовують механізм підстановки параметрів через %s, що унеможливорює SQL-ін'єкції.

Основним запитом для отримання інформації про товар є get_product_full, який об'єднує дані з п'яти таблиць:

```
SELECT p.product_id, pd.name AS product_name, pd.description AS existing_desc,
       cd.name AS category, COALESCE(m.name, '') AS manufacturer
FROM fdb_product p
JOIN fdb_product_description pd ON p.product_id = pd.product_id AND pd.language_id = %s
LEFT JOIN fdb_product_to_category ptc ON p.product_id = ptc.product_id
LEFT JOIN fdb_category_description cd ON ptc.category_id = cd.category_id AND
cd.language_id = %s
LEFT JOIN fdb_manufacturer m ON p.manufacturer_id = m.manufacturer_id
WHERE p.product_id = %s
```

Для отримання атрибутів товару (тканина, склад, розмір тощо) використовується запит:

```
SELECT ad.name AS attr_name, pa.text AS attr_value
FROM fdb_product_attribute pa
JOIN fdb_attribute_description ad ON pa.attribute_id = ad.attribute_id AND ad.language_id = %s
WHERE pa.product_id = %s AND pa.language_id = %s
```

Перед обробкою товару система перевіряє, чи він уже був оптимізований:

```
SELECT COUNT(*) FROM ai_optimization_log
WHERE entity_type = 'product' AND entity_id = %s
AND operation_type = 'full_seo' AND status = 'success'
```

Перед перезаписом SEO-полів система зберігає поточні значення у таблицю ai_seo_backup:

```
INSERT INTO ai_seo_backup (product_id, language_id, old_description, old_meta_title,  
old_meta_desc, old_meta_kw)  
VALUES (%s, %s, %s, %s, %s, %s)
```

Після генерації контенту виконується оновлення відповідних полів у fdb_product_description:

```
UPDATE fdb_product_description  
SET description = %s, meta_title = %s, meta_description = %s, meta_keyword = %s  
WHERE product_id = %s AND language_id = %s
```

Кожна операція фіксується у таблиці ai_optimization_log:

```
INSERT INTO ai_optimization_log (entity_type, entity_id, operation_type, status, tokens_used,  
model_version, content_hash)  
VALUES (%s, %s, %s, %s, %s, %s, %s)
```

ВИСНОВКИ

У кваліфікаційній роботі розроблено інформаційну систему e-commerce-платформи із застосуванням ШІ-технологій для SEO-оптимізації інтернет-магазину домашнього текстилю fionahome.com.ua.

За результатами виконаної роботи сформульовано такі висновки.

1. Проаналізовано сучасні підходи до SEO-оптимізації e-commerce-платформ, особливості функціонування CMS OpenCart та специфіку SEO-просування інтернет-магазинів домашнього текстилю.
2. Проаналізовано сучасні ШІ-технології для автоматизованої генерації текстового контенту та обґрунтовано вибір великих мовних моделей і методів NLP для вирішення завдань SEO-оптимізації.
3. Розроблено концептуальну модель інформаційної системи, ER-модель бази даних та архітектуру ШІ-модуля автоматизованої генерації SEO-контенту для інтернет-магазину на базі OpenCart.
4. Реалізовано програмне рішення у вигляді веборієнтованого модуля з інтеграцією сервісів штучного інтелекту, що забезпечує автоматичну генерацію описів товарів, метатегів та інших SEO-атрибутів.
5. Впроваджено механізм семантичної кластеризації пошукових запитів і товарних категорій на основі векторних ембедингів та алгоритму K-Means, а також засоби резервного копіювання, валідації контенту та логування операцій.
6. Проаналізовано ефективність розробленої системи в умовах реального інтернет-магазину, підтверджено можливість автоматизації SEO-процесів, скорочення витрат часу на наповнення контентом та підвищення масштабованості SEO-оптимізації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. OpenCart MVC Architecture Overview : веб-сайт. URL: <https://docs.opencart.com/en-gb/developer/framework/> (дата звернення: 15.04.2025).
2. Нова Пошта. Документація до API : веб-сайт. URL: <https://developers.novaposhta.ua/documentation> (дата звернення: 16.04.2025).
3. Elmasri R., Navathe S. Fundamentals of Database Systems. 7th ed. USA : Pearson, 2015. 1280 p.
4. Date C. J. An Introduction to Database Systems. 8th ed. USA : Addison-Wesley, 2003. 1024 p.
5. Silberschatz A., Korth H., Sudarshan S. Database System Concepts. 7th ed. USA : McGraw-Hill Education, 2019. 1376 p.
6. Pettit T., Scott C. The MySQL Workshop: A Practical Guide to Working with Data and Managing Databases with MySQL. UK : Packt Publishing, 2022. 726 p.
7. Smirnova S., Tezuysal A. MySQL Cookbook: Solutions for Database Developers and Administrators. 4th ed. USA : O'Reilly Media, 2022. 922 p.
8. Gutmans A., Bakken S. S., Rethans D. PHP 5 Power Programming. USA : Prentice Hall, 2004. 720 p.
9. LiqPay API Documentation. Приват Банк : веб-сайт. URL: <https://www.liqpay.ua/documentation/api/acquiring/checkout/> (дата звернення: 20.04.2025).
10. WayForPay API Documentation : веб-сайт. URL: <https://wiki.wayforpay.com/view/852102> (дата звернення: 20.04.2025).
11. PHP: The Right Way. A quick reference for PHP best practices : веб-сайт. URL: <https://phptherightway.com> (дата звернення: 21.04.2025).

12. PHPUnit Documentation. Sebastian Bergmann : веб-сайт. URL: <https://phpunit.de/documentation.html> (дата звернення: 21.04.2025).
13. Frain B. Responsive Web Design with HTML5 and CSS. 3rd ed. UK : Packt Publishing, 2020. 460 p.
14. Bootstrap 3 Documentation. The Bootstrap team : веб-сайт. URL: <https://getbootstrap.com/docs/3.4/> (дата звернення: 22.04.2025).
15. jQuery API Documentation : веб-сайт. URL: <https://api.jquery.com> (дата звернення: 22.04.2025).
16. MDN Web Docs. HTML / CSS / JavaScript Reference. Mozilla : веб-сайт. URL: <https://developer.mozilla.org/en-US/docs/Web> (дата звернення: 23.04.2025).
17. Web Content Accessibility Guidelines (WCAG) 2.1. W3C Recommendation : веб-сайт. URL: <https://www.w3.org/TR/WCAG21/> (дата звернення: 23.04.2025).
18. Page Object Model. Selenium Design Patterns : веб-сайт. URL: https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models/ (дата звернення: 24.04.2025).
19. webdriver-manager Documentation. Python package for managing browser drivers : веб-сайт. URL: https://github.com/SergeyPirogov/webdriver_manager (дата звернення: 24.04.2025).
20. Reis J., Housley M. Fundamentals of Data Engineering: Plan and Build Robust Data Systems. USA : O'Reilly Media, 2022. 446 p.
21. Schwartz B., Zaitsev P., Tkachenko V. High Performance MySQL. 3rd ed. USA : O'Reilly Media, 2012. 826 p.

22. MySQL 8.0 Reference Manual. Oracle Corporation : веб-сайт. URL: <https://dev.mysql.com/doc/refman/8.0/en/> (дата звернення: 26.04.2025).

23. Google Core Web Vitals. Search Central Documentation : веб-сайт. URL: <https://developers.google.com/search/docs/appearance/core-web-vitals> (дата звернення: 26.04.2025).

24. Postman API Platform Documentation : веб-сайт. URL: <https://learning.postman.com/docs/getting-started/introduction/> (дата звернення: 27.04.2025).

25. Chodak G., Błażyczek K. Large Language Models for Search Engine Optimization in E-commerce / Grzegorz Chodak, Klaudia Błażyczek. – ResearchGate, 2024. – 15 с. – DOI: https://doi.org/10.1007/978-3-031-56700-1_27.

26. Research of the Organic Traffic Optimization System for E-Commerce Platforms Using Large Language Models / O. Levchenko, I. Dronyuk, O. Fedevych, O. Smirna. – Computer Design Systems. Theory and Practice. – 2025. – Volume 7, Number 2. – С. 143–156. – Режим доступу: <https://science.lpnu.ua/cds/all-volumes-and-issues/volume-7-number-2-2025/research-organic-traffic-optimization-system-e> (дата звернення: 30.04.2026).

27. Aliev R. Improving E-commerce Search with Category-Aligned Retrieval / Rauf Aliev. – arXiv preprint, 2025. – 10 с. – Режим доступу: <https://arxiv.labs.arxiv.org/html/2510.21711> (дата звернення: 30.04.2026).

28. Saha D., Zaman A., Zou H., Chen N., Shu X., Vase N., Bagherjeiran A. Improving Ad Matching via Cluster-Adaptive Keyword Expansion and Relevance Tuning / Dipanwita Saha, Anis Zaman, Hua Zou, Ning Chen, Xinxin Shu, Nadia Vase, Abraham Bagherjeiran. – In: ECOM'25: SIGIR Workshop on eCommerce, July 17, 2025, Padua, Italy. – arXiv preprint, 2025. – 8 с. – Режим

доступу: <https://ar5iv.labs.arxiv.org/html/2505.18897> (дата звернення: 30.04.2026).

29. Schweidel D., Knox G., et al. Moving Beyond ChatGPT: Applying Large Language Models in Marketing Contexts / David Schweidel, George Knox, et al. – Journal of Marketing Research & Innovation. – Sciendo, 2024. – 14 с. – DOI: <https://doi.org/10.2478/nimmir-2024-0004>.

30. Aggarwal P., Murahari V., Rajpurohit T., Kalyan A., Narasimhan K., Deshpande A. GEO: Generative Engine Optimization / Pranav Aggarwal, Vishvak Murahari, Tanmay Rajpurohit, Ashwin Kalyan, Karthik Narasimhan, Ameet Deshpande. – In: KDD '24 Proceedings. – ACM, 2024. – 12 с. – DOI: <https://doi.org/10.1145/3637528.3671900>.

31. Google Search Central. Google Search's Guidance About AI-Generated Content [Електронний ресурс] / Google Search Central Blog. – 2024. – Режим доступу: <https://developers.google.com/search/blog/2023/02/google-search-and-ai-content> (дата звернення: 30.04.2026).

32. Jones H. The Impact of AI and LLM on Your Organic Traffic [Електронний ресурс] / Henry Jones. – Proof3, 2025. – Режим доступу: <https://proof3.co/insights/impact-of-ai-and-llm-on-organic-traffic> (дата звернення: 30.04.2026).

33. Basecom GmbH. Digital Insights: AI, SEO and LLMO in E-Commerce [Електронний ресурс] / basecom. – 2025. – Режим доступу: <https://basecom.de/en/news/digital-insights-ai-seo-and-llmo-in-e-commerce> (дата звернення: 30.04.2026).

34. Single Grain. How to Evaluate AI SEO Tools for Agencies in 2025 [Електронний ресурс] / Single Grain. – 2025. – Режим доступу:

<https://www.singlegrain.com/artificial-intelligence/how-to-evaluate-ai-seo-tools-for-agencies-in-2025> (дата звернення: 30.04.2026).

35. eWEEK. 6 Best AI SEO Tools (Rank Faster with Less Work) [Електронний ресурс] / eWEEK. – 2025. – Режим доступу: <https://www.eweek.com/artificial-intelligence/best-ai-seo-tools/> (дата звернення: 30.04.2026).

36. n8n. Auto-Generate Product Descriptions with Azure OpenAI for Adobe Commerce/Magento [Електронний ресурс] / n8n workflow template. – 2026. – Режим доступу: <https://n8n.io/workflows/8475-auto-generate-product-descriptions-with-azure-openai-for-adobe-commerce-magento> (дата звернення: 30.04.2026).

37. n8n. Auto-Generate SEO Meta Descriptions & Keywords for Magento 2 with GPT-4.1 & LangChain [Електронний ресурс] / n8n workflow template. – 2025. – Режим доступу: <https://n8n.io/workflows/6654-auto-generate-seo-meta-descriptions-and-keywords-for-magento-2-with-gpt-41-and-langchain/> (дата звернення: 30.04.2026).

ДОДАТКИ

Додаток А

Фрагменти коду програми:

App.py:

```

"""
app.py – Flask веб-інтерфейс для системи SEO AI
Використовує готові модулі: ai/, db/, processing/, utils/, config/

Запуск: python app.py
Відкрити: http://localhost:5000
"""

import os
import sys
import json
import threading
import traceback
from datetime import datetime

from flask import Flask, render_template, request, jsonify

# Гарантуємо що модулі проекту доступні
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))

from dotenv import load_dotenv
load_dotenv()

app = Flask(__name__)
app.secret_key = os.urandom(24)

# — Глобальний стан поточного завдання —————
_job = {
    "running": False,
    "log": [],
    "stats": {"total":0,"done":0,"success":0,"skipped":0,"errors":0,"tokens":0},
    "db_cfg": None, # dict – зберігаємо між запитами
    "ai_cfg": None,
}

# =====

```

```

# ДОПОМІЖНІ: підключення через модулі проєкту
# =====

def make_db_connector(cfg: dict):
    """
    Створює DatabaseConnector з db/connector.py використовуючи
    параметри з веб-форми (не з .env).
    """
    from db.connector import DatabaseConnector
    return DatabaseConnector(
        host      = cfg["host"],
        port      = int(cfg["port"]),
        database   = cfg["database"],
        user       = cfg["user"],
        password   = cfg["password"],
    )

def make_generator(ai_cfg: dict, tracker=None):
    """
    Повертає генератор з ai/ залежно від провайдера.
    Тимчасово виставляємо env-змінні щоб конструктори підхопили ключі.
    """
    provider = ai_cfg["provider"]
    key       = ai_cfg.get("api_key", "")
    model     = ai_cfg.get("model", "")

    if provider == "gemini":
        os.environ["GEMINI_API_KEY"] = key
        if model: os.environ["GEMINI_MODEL"] = model
        from ai.gemini_generator import GeminiContentGenerator
        return GeminiContentGenerator(api_key=key, model=model or "gemini-2.0-flash",
                                      tracker=tracker)

    elif provider == "openai":
        os.environ["OPENAI_API_KEY"] = key
        if model: os.environ["OPENAI_MODEL"] = model
        from ai.content_generator import ContentGenerator
        return ContentGenerator(api_key=key, model=model or "gpt-4o-mini",
                                tracker=tracker)

    elif provider == "claude":
        os.environ["ANTHROPIC_API_KEY"] = key

```

```

        if model: os.environ["CLAUDE_MODEL"] = model
        from ai.claude_generator import ClaudeContentGenerator
        return ClaudeContentGenerator(api_key=key, model=model or "claude-haiku-4-5-
20251001",
                                     tracker=tracker)

    raise ValueError(f"Невідомий провайдер: {provider}")

def _log(msg: str):
    """Додає рядок у лог поточного завдання."""
    ts = datetime.now().strftime("%H:%M:%S")
    _job["log"].append(f"[{ts}] {msg}")

# =====
#  СТОПІНКИ
# =====

@app.route("/")
def index():
    return render_template("index.html")

# =====
#  API – БАЗА ДАНИХ
# =====

@app.route("/api/db/test", methods=["POST"])
def api_db_test():
    cfg = request.json
    try:
        # Тимчасово підмінюємо налаштування prefix/language_id
        _patch_settings(cfg)
        db = make_db_connector(cfg)
        P = cfg.get("prefix", "fdb_")
        row = db.fetchone(f"SELECT COUNT(*) AS cnt FROM `{P}product`")
        cnt = row["cnt"] if row else 0
        db.close()
        _job["db_cfg"] = cfg
        return jsonify({"ok": True, "products": cnt,
                       "message": f"Підключено! Знайдено {cnt:,} товарів."})
    except Exception as e:
        return jsonify({"ok": False, "message": str(e)})

```

```

@app.route("/api/db/save", methods=["POST"])
def api_db_save():
    cfg = request.json
    _patch_settings(cfg)
    _job["db_cfg"] = cfg
    return jsonify({"ok": True})

@app.route("/api/categories", methods=["POST"])
def api_categories():
    cfg = _job.get("db_cfg") or request.json.get("db_config")
    if not cfg:
        return jsonify({"ok": False, "message": "БД не підключена"})
    try:
        _patch_settings(cfg)
        db = make_db_connector(cfg)
        P = cfg.get("prefix", "fdb_")
        LID = int(cfg.get("language_id", 5))
        rows = db.fetchall(f"""
            SELECT cd.category_id, cd.name, COUNT(ptc.product_id) AS cnt
            FROM `{P}category_description` cd
            LEFT JOIN `{P}product_to_category` ptc ON cd.category_id=ptc.category_id
            WHERE cd.language_id=%s
            GROUP BY cd.category_id ORDER BY cd.name
            """, (LID,))
        db.close()
        return jsonify({"ok": True, "categories": rows})
    except Exception as e:
        return jsonify({"ok": False, "message": str(e)})

@app.route("/api/products", methods=["POST"])
def api_products():
    cfg = _job.get("db_cfg") or request.json.get("db_config")
    if not cfg:
        return jsonify({"ok": False, "message": "Спочатку підключіть БД"})
    try:
        _patch_settings(cfg)
        db = make_db_connector(cfg)
        P = cfg.get("prefix", "fdb_")
        LID = int(cfg.get("language_id", 5))

```

```

search    = (request.json or {}).get("search","").strip()
category  = (request.json or {}).get("category","")

# Будуємо WHERE-умови та params у правильному порядку
# Спочатку йдуть %s з JOIN-ів (LID, LID), потім з WHERE
join_params = [LID, LID]    # для ON pd.language_id=%s і cd.language_id=%s

where_parts = ["pd.language_id = %s"]
where_params = [LID]

if search:
    where_parts.append("pd.name LIKE %s")
    where_params.append(f"%{search}%")
if category:
    where_parts.append("ptc.category_id = %s")
    where_params.append(int(category))

# PyMySQL підставляє %s зліва направо – JOIN-и йдуть ДО WHERE у SQL
all_params = tuple(join_params + where_params)

rows = db.fetchall(f"""
    SELECT p.product_id, pd.name AS product_name, p.status,
           p.price, cd.name AS category,
           COALESCE(m.name, '') AS manufacturer,
           CASE WHEN (pd.meta_title IS NULL OR pd.meta_title='') THEN 0 ELSE
1 END AS has_seo
    FROM `{P}product` p
    JOIN `{P}product_description` pd
      ON p.product_id=pd.product_id AND pd.language_id=%s
    LEFT JOIN `{P}product_to_category` ptc ON p.product_id=ptc.product_id
    LEFT JOIN `{P}category_description` cd
      ON ptc.category_id=cd.category_id AND cd.language_id=%s
    LEFT JOIN `{P}manufacturer` m ON p.manufacturer_id=m.manufacturer_id
    WHERE {" AND ".join(where_parts)}
    GROUP BY p.product_id ORDER BY p.product_id LIMIT 1000
""", all_params)
db.close()
return jsonify({"ok": True, "products": rows, "total": len(rows)})
except Exception as e:
    return jsonify({"ok": False, "message": str(e)})

```

```
# API – III
```

```
#
```

```
@app.route("/api/ai/test", methods=["POST"])
def api_ai_test():
    ai_cfg = request.json
    try:
        from utils.cost_tracker import CostTracker
        tracker = CostTracker(model=ai_cfg.get("model","?"))
        gen = make_generator(ai_cfg, tracker)
        # Мінімальний тест – генеруємо коротке речення
        text = gen._call(
            "Напиши одне речення українською про домашній текстиль.",
            max_tokens=60, temperature=0.5
        )
        _job["ai_cfg"] = ai_cfg
        return jsonify({"ok": True,
                        "sample": text,
                        "message": f"Підключено! Провайдер: {ai_cfg['provider']} | "
                        f"{ai_cfg.get('model','')}"}))
    except Exception as e:
        return jsonify({"ok": False, "message": str(e)})
```

```
@app.route("/api/ai/save", methods=["POST"])
def api_ai_save():
    _job["ai_cfg"] = request.json
    return jsonify({"ok": True})
```

```
#
```

```
# API – ЗАПУСК ОБРОБКИ (використовує ProductProcessor)
```

```
#
```

```
@app.route("/api/process", methods=["POST"])
def api_process():
    if _job["running"]:
        return jsonify({"ok": False, "message": "Вже виконується обробка. "
        "Зачекайте."})
```

```
    data = request.json or {}
    db_cfg = _job.get("db_cfg")
    ai_cfg = _job.get("ai_cfg")
    product_ids = data.get("product_ids", [])
```

```

dry_run      = data.get("dry_run", False)
fields       = data.get("fields",
["description","meta_title","meta_description","meta_keyword"])

if not db_cfg:
    return jsonify({"ok": False, "message": "Налаштуйте підключення до БД"})
if not ai_cfg:
    return jsonify({"ok": False, "message": "Налаштуйте ШІ-провайдер"})
if not product_ids:
    return jsonify({"ok": False, "message": "Оберіть товари для обробки"})

_job["running"] = True
_job["log"]      = []
_job["stats"]    = {"total": len(product_ids), "done":0,
                    "success":0, "skipped":0, "errors":0, "tokens":0}

def run():
    try:
        _patch_settings(db_cfg)
        _set_ai_env(ai_cfg)

        from db.connector import DatabaseConnector
        from db.queries import (
            get_product_full, get_product_attributes,
            update_product_seo, insert_optimization_log,
        )
        from utils.cost_tracker import CostTracker
        from utils.validator import validate_all, all_valid,
log_validation_results

        tracker = CostTracker(model=ai_cfg.get("model","?"))
        gen      = make_generator(ai_cfg, tracker)

        db = make_db_connector(db_cfg)

        # Переконаємось що таблиці логу існують
        _ensure_tables(db)

        total = len(product_ids)
        for idx, pid in enumerate(product_ids, 1):
            pid = int(pid)
            stats = _job["stats"]

```

```

# Читаємо повний профіль через get_product_full (з db/queries.py)
product = get_product_full(db, pid)
if not product:
    _log(f"⚠ Товар id={pid} не знайдено – пропускаємо")
    stats["skipped"] += 1
    stats["done"] += 1
    continue

name = product.get("product_name", "?")
_log(f"[{idx}/{total}] ➡ {name} (id={pid})")

try:
    result = {}
    tokens_before = tracker.total_tokens

    # meta_title – детерміновано, не через LLM
    if "meta_title" in fields:
        result["meta_title"] = gen.generate_meta_title(product)

    # description – через LLM
    if "description" in fields:
        _log(" → Генерація опису...")
        result["description"] =
gen.generate_product_description(product)

    # meta_description – через LLM
    if "meta_description" in fields:
        _log(" → Генерація meta description...")
        result["meta_description"] =
gen.generate_meta_description(product)

    # meta_keyword – через LLM (додаткове поле)
    if "meta_keyword" in fields:
        _log(" → Генерація meta keywords...")
        result["meta_keyword"] = _generate_keywords(gen, product)

    tokens_used = tracker.total_tokens - tokens_before
    stats["tokens"] += tokens_used

    # Валідація через utils/validator.py
    v_results = validate_all(
        description      = result.get("description"),
        meta_title       = result.get("meta_title"),

```

```

        meta_description = result.get("meta_description"),
    )
    valid = all_valid(v_results)
    if not valid:
        issues = "; ".join(
            f"{k}: {r}" for k,r in v_results.items() if not r.valid
        )
        _log(f" ⚠️ Валідація: {issues}")

    if dry_run:
        desc_words = len((result.get("description") or "").split())
        title_len = len(result.get("meta_title", ""))
        meta_len = len(result.get("meta_description", ""))
        kw_count = len((result.get("meta_keyword") or
"".split(",")

        _log(f" ✓ [DRY RUN] Опис: {desc_words} сл. | "
            f"Title: {title_len} симв. | Meta: {meta_len} симв. | "
            f"KW: {kw_count} | Tokens: {tokens_used}")
        stats["success"] += 1

    else:
        # Резервна копія старих SEO-полів
        _backup_seo(db, db_cfg, pid)

        # Оновлення через update_product_seo (з db/queries.py)
        # (передаємо тільки поля що обираються)
        update_product_seo(
            db,
            product_id = pid,
            description = result.get("description") if
"description" in fields else None,
            meta_title = result.get("meta_title") if
"meta_title" in fields else None,
            meta_description = result.get("meta_description") if
"meta_description" in fields else None,
        )

        # Якщо обрано meta_keyword – оновлюємо окремо
        if "meta_keyword" in fields and result.get("meta_keyword"):
            db.execute(
                f"UPDATE
`{db_cfg.get('prefix','fdb_')}product_description` "

```

```

        f"SET meta_keyword=%s WHERE product_id=%s AND
language_id=%s",
        (result["meta_keyword"], pid,
int(db_cfg.get("language_id",5)))
    )

    # Лог чеpez insert_optimization_log (з db/queries.py)
    import hashlib
    h =
    hashlib.md5((result.get("description","")).encode()).hexdigest()
    insert_optimization_log(
        db,
        entity_type    = "product",
        entity_id      = pid,
        operation_type  = "full_seo",
        status         = "success" if valid else "skipped",
        tokens_used     = tokens_used,
        model_version   = ai_cfg.get("model","?"),
        content_hash    = h,
    )

    _log(f"  ✓ Збережено у БД | Tokens: {tokens_used}")
    stats["success"] += 1

except Exception as e:
    _log(f"  X Помилка: {e}")
    stats["errors"] += 1
    if not dry_run:
        try:
            insert_optimization_log(
                db, "product", pid, "full_seo", "error",
                model_version=ai_cfg.get("model","?")
            )
        except Exception:
            pass

    stats["done"] += 1

db.close()

# Фінальна статистика через cost_tracker
tracker.log_summary()
s = _job["stats"]

```

```

        _log(f"{'-'*55}")
        _log(f"✅ Завершено! Успішно: {s['success']} | "
              f"Помилка: {s['errors']} | Токенів: {s['tokens']:,}")
        _log(f"💰 {tracker.summary()}")

    except Exception as e:
        _log(f"❌ Критична помилка: {e}")
        _log(traceback.format_exc())
    finally:
        _job["running"] = False

    threading.Thread(target=run, daemon=True).start()
    return jsonify({"ok": True})

@app.route("/api/process/status")
def api_process_status():
    return jsonify({
        "running": _job["running"],
        "stats": _job["stats"],
        "log": _job["log"][-60:],
    })

@app.route("/api/process/stop", methods=["POST"])
def api_process_stop():
    _job["running"] = False
    _log(f"🛑 Зупинено користувачем")
    return jsonify({"ok": True})

# =====
# API – СТАТИСТИКА
# =====

@app.route("/api/stats")
def api_stats():
    cfg = _job.get("db_cfg")
    if not cfg:
        return jsonify({"ok": False, "message": "БД не підключена"})
    try:
        _patch_settings(cfg)
        db = make_db_connector(cfg)
        P = cfg.get("prefix", "fdb_")

```

```

LID = int(cfg.get("language_id",5))

from db.queries import get_optimization_stats

total      = db.fetchval(f"SELECT COUNT(*) FROM `{P}`product` WHERE status=1")
or 0

no_seo      = db.fetchval(
    f"SELECT COUNT(*) FROM `{P}`product_description` "
    f"WHERE language_id=%s AND (meta_title IS NULL OR meta_title='')",
(LID,)) or 0
optimized = db.fetchval(
    "SELECT COUNT(DISTINCT entity_id) FROM ai_optimization_log "
    "WHERE entity_type='product' AND status='success'") or 0
tokens     = db.fetchval(
    "SELECT COALESCE(SUM(tokens_used),0) FROM ai_optimization_log") or 0
recent     = db.fetchall(
    "SELECT entity_id, status, executed_at, model_version, tokens_used "
    "FROM ai_optimization_log ORDER BY executed_at DESC LIMIT 10")

# Резервні копії
backups = 0
try:
    backups = db.fetchval("SELECT COUNT(*) FROM ai_seo_backup") or 0
except Exception:
    pass

db.close()
return jsonify({
    "ok": True,
    "total_products": total,
    "no_seo":        no_seo,
    "optimized":     optimized,
    "total_tokens":  tokens,
    "backups":       backups,
    "recent":        recent,
})
except Exception as e:
    return jsonify({"ok": False, "message": str(e)})

```

```

# =====
# ВНУТРІШНІ ДОПОМІЖНІ ФУНКЦІЇ
# =====

```

```

def _patch_settings(cfg: dict):
    """
    Тимчасово виставляє змінні середовища щоб модулі config/settings.py
    підхоплювали значення з веб-форми.
    """
    import config.settings as S
    if cfg.get("prefix"): S.TABLE_PREFIX = cfg["prefix"]
    if cfg.get("language_id"): S.LANGUAGE_ID = int(cfg["language_id"])

def _set_ai_env(ai_cfg: dict):
    """Виставляє env-змінні для AI-провайдерів."""
    p = ai_cfg.get("provider", "")
    k = ai_cfg.get("api_key", "")
    m = ai_cfg.get("model", "")
    if p == "gemini":
        os.environ["GEMINI_API_KEY"] = k
        if m: os.environ["GEMINI_MODEL"] = m
    elif p == "openai":
        os.environ["OPENAI_API_KEY"] = k
        if m: os.environ["OPENAI_MODEL"] = m
    elif p == "claude":
        os.environ["ANTHROPIC_API_KEY"] = k
        if m: os.environ["CLAUDE_MODEL"] = m

def _generate_keywords(gen, product: dict) -> str:
    """Генерує meta_keyword через LLM. Використовує внутрішній метод _call."""
    name = product.get("product_name", "")
    cat = product.get("category", "")
    attrs = "; ".join(f"{k}: {v}" for k, v in (product.get("attributes") or
    {}).items()) or ""
    prompt = (
        f"Склади список ключових слів (meta keywords) для товару.\n"
        f"Товар: {name}\nКатегорія: {cat}\nАтрибути: {attrs}\n\n"
        f"Вимоги:\n- Мова: українська\n- 5-8 ключових фраз через кому\n"
        f"- Включи комерційні запити (купити, ціна, замовити)\n"
        f"- Повертай ТІЛЬКИ список ключових слів, без пояснень"
    )
    return gen._call(prompt, max_tokens=120, temperature=0.4)

def _ensure_tables(db):

```

```

"""Створює допоміжні таблиці якщо їх немає."""
db.execute("""
    CREATE TABLE IF NOT EXISTS `ai_optimization_log` (
        `log_id`          INT(11) NOT NULL AUTO_INCREMENT,
        `entity_type`     VARCHAR(20) NOT NULL,
        `entity_id`       INT(11) NOT NULL,
        `operation_type`  VARCHAR(50) NOT NULL DEFAULT 'full_seo',
        `executed_at`     DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
        `status`          VARCHAR(20) NOT NULL,
        `tokens_used`     INT(11) DEFAULT NULL,
        `model_version`   VARCHAR(50) DEFAULT NULL,
        `content_hash`    VARCHAR(64) DEFAULT NULL,
        PRIMARY KEY (`log_id`),
        KEY `idx_entity` (`entity_type`, `entity_id`)
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
""")
db.execute("""
    CREATE TABLE IF NOT EXISTS `ai_seo_backup` (
        `backup_id`       INT(11) NOT NULL AUTO_INCREMENT,
        `product_id`      INT(11) NOT NULL,
        `language_id`     INT(11) NOT NULL DEFAULT 5,
        `backed_up_at`    DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
        `old_description` MEDIUMTEXT,
        `old_meta_title`  VARCHAR(255),
        `old_meta_desc`   VARCHAR(255),
        `old_meta_kw`     MEDIUMTEXT,
        PRIMARY KEY (`backup_id`),
        KEY `idx_product` (`product_id`)
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
""")

def _backup_seo(db, cfg: dict, product_id: int):
    """Зберігає поточні SEO-поля товару перед перезаписом."""
    P = cfg.get("prefix", "fdb_")
    LID = int(cfg.get("language_id", 5))
    row = db.fetchone(
        f"SELECT description, meta_title, meta_description, meta_keyword "
        f"FROM `{P}product_description` WHERE product_id=%s AND language_id=%s",
        (product_id, LID)
    )
    if row:
        db.execute(

```

```

        "INSERT INTO ai_seo_backup "
        "(product_id, language_id, old_description, old_meta_title,
old_meta_desc, old_meta_kw) "
        "VALUES (%s,%s,%s,%s,%s,%s)",
        (product_id, LID,
        row.get("description"), row.get("meta_title"),
        row.get("meta_description"), row.get("meta_keyword"))
    )

# =====
# ЗАПУСК
# =====

if __name__ == "__main__":
    print("=" * 55)
    print("  FionaHome SEO AI – Веб-інтерфейс")
    print("  Відкрийте браузер: http://localhost:5000")
    print("=" * 55)
    app.run(debug=False, host="0.0.0.0", port=5000, threaded=True)

```

Main.py:

```

import argparse
import sys
import os
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
def cmd_products(args) -> None:
    """Команда: ОПТИМІЗАЦІЯ ТОВАРІВ."""
    from db.connector import DatabaseConnector
    from processing.product_processor import ProductProcessor

    with DatabaseConnector() as db:
        processor = ProductProcessor(
            db,
            limit=args.limit,
            skip_optimized=not args.force,
            dry_run=args.dry_run,
        )

        if args.id:
            ok = processor.run_single(args.id)
            print("✓ Товар оброблено." if ok else "X Товар не оброблено.")
        else:
            stats = processor.run()
            _print_stats("ТОВАРИ", stats)

```

```

def cmd_categories(args) -> None:
    """Команда: оптимізація категорій."""
    from db.connector import DatabaseConnector
    from processing.category_processor import CategoryProcessor

    with DatabaseConnector() as db:
        processor = CategoryProcessor(
            db,
            limit=args.limit,
            skip_optimized=not args.force,
            dry_run=args.dry_run,
        )
        stats = processor.run()
        _print_stats("КАТЕГОРІЇ", stats)

def cmd_cluster(args) -> None:
    """Команда: семантична кластеризація пошукових запитів."""
    from ai.semantic_cluster import SemanticClusterer, cluster_from_gsc

    queries = []

    if args.input:
        # Читаємо запити з текстового файлу (по одному на рядок)
        if not os.path.exists(args.input):
            print(f"X Файл не знайдено: {args.input}")
            sys.exit(1)
        with open(args.input, encoding="utf-8") as f:
            queries = [line.strip() for line in f if line.strip()]
        print(f"Завантажено {len(queries)} запитів із {args.input}")
    else:
        # Читаємо з Google Search Console
        try:
            from config.settings import GSC_CREDENTIALS_FILE, GSC_SITE_URL
            if not os.path.exists(GSC_CREDENTIALS_FILE):
                print(
                    f"X Файл credentials GSC не знайдено: {GSC_CREDENTIALS_FILE}\n"
                    "    Вкажіть GSC_CREDENTIALS_FILE у .env або використайте -"
                    "-input"
                )
                sys.exit(1)

            # GSC клієнт (описаний у документі, тут – скорочений виклик)
            from google.oauth2 import service_account
            from googleapiclient.discovery import build
            from datetime import date, timedelta

            scopes = ["https://www.googleapis.com/auth/webmasters.readonly"]
            creds = service_account.Credentials.from_service_account_file(
                GSC_CREDENTIALS_FILE, scopes=scopes
            )
            service = build("searchconsole", "v1", credentials=creds)

```

```

end_date = date.today()
start_date = end_date - timedelta(days=90)
body = {
    "startDate": str(start_date),
    "endDate": str(end_date),
    "dimensions": ["query"],
    "rowLimit": 1000,
}
resp = (
    service.searchanalytics()
    .query(siteUrl=GSC_SITE_URL, body=body)
    .execute()
)
queries = [r["keys"][0] for r in resp.get("rows", [])]
print(f"Отримано {len(queries)} запитів із Google Search
Console.")
except ImportError:
    print(
        "X Бібліотека google-api-python-client не встановлена.\n"
        "  pip install google-api-python-client google-auth\n"
        "  Або використайте --input файл_із_запитами.txt"
    )
    sys.exit(1)

if not queries:
    print("X Список запитів порожній.")
    sys.exit(1)

clusterer = SemanticClusterer()
clusters = clusterer.cluster_queries(queries, n_clusters=args.k or None)

print(f"\n{'-'*60}")
print(f"Результат кластеризації ({len(clusters)} кластерів):")
print(f"{'-'*60}")
for name, qs in clusters.items():
    print(f"\n✎ {name} ({len(qs)} запитів):")
    for q in qs[:5]:
        print(f"    • {q}")
    if len(qs) > 5:
        print(f"    ... і ще {len(qs) - 5}")

# Збереження у БД (опціонально)
if args.save:
    from db.connector import DatabaseConnector
    from db.queries import insert_semantic_cluster
    saved = 0
    with DatabaseConnector() as db:
        for cluster_name, qs in clusters.items():
            for q in qs:
                insert_semantic_cluster(
                    db,
                    cluster_name=cluster_name,
                    query_text=q,
                )

```

```

        saved += 1
    print(f"\n✓ Збережено {saved} записів у ai_semantic_cluster.")

def cmd_stats(args) -> None:
    """Команда: статистика оптимізацій."""
    from db.connector import DatabaseConnector
    from db.queries import get_optimization_stats

    with DatabaseConnector() as db:
        rows = get_optimization_stats(db)

    if not rows:
        print("Лог оптимізацій порожній – жодних записів ще немає.")
        return

    print(f"\n{'-'*70}")
    print(f"{'entity_type':<12} {'operation':<15} {'status':<10} {'count':>6} {'tokens':>10}")
    print(f"{'-'*70}")
    total_tokens = 0
    for r in rows:
        tokens = r.get("total_tokens") or 0
        total_tokens += tokens
        print(
            f"{r['entity_type']:<12} "
            f"{r['operation_type']:<15} "
            f"{r['status']:<10} "
            f"{r['cnt']:>6} "
            f"{tokens:>10,}"
        )
    print(f"{'-'*70}")
    print(f"{'РАЗОМ':>39} {total_tokens:>10,} токенів")

def cmd_setup_db(args) -> None:
    """Команда: створення розширених таблиць у БД."""
    SQL = """
CREATE TABLE IF NOT EXISTS `ai_optimization_log` (
  `log_id`          INT(11)          NOT NULL AUTO_INCREMENT,
  `entity_type`     ENUM('product','category','image') NOT NULL,
  `entity_id`       INT(11)          NOT NULL,
  `operation_type`  VARCHAR(50)      NOT NULL,
  `executed_at`     DATETIME         NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `status`          ENUM('success','error','skipped') NOT NULL,
  `tokens_used`     INT(11)          DEFAULT NULL,
  `model_version`   VARCHAR(50)      DEFAULT NULL,
  `content_hash`    VARCHAR(64)      DEFAULT NULL,
  PRIMARY KEY (`log_id`),
  KEY `idx_entity` (`entity_type`, `entity_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE IF NOT EXISTS `ai_semantic_cluster` (
  `cluster_id`      INT(11)          NOT NULL AUTO_INCREMENT,

```

```

        `cluster_name`      VARCHAR(255) NOT NULL,
        `query_text`        VARCHAR(500) NOT NULL,
        `embedding_vector`  JSON          DEFAULT NULL,
        `assigned_url`      VARCHAR(500)   DEFAULT NULL,
        `monthly_volume`    INT(11)        DEFAULT NULL,
        `created_at`        DATETIME       NOT NULL DEFAULT CURRENT_TIMESTAMP,
        PRIMARY KEY (`cluster_id`),
        KEY `idx_cluster_name` (`cluster_name`)
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
"""

from db.connector import DatabaseConnector
with DatabaseConnector() as db:
    for statement in SQL.strip().split(";"):
        stmt = statement.strip()
        if stmt:
            db.execute(stmt)

print("✓ Таблиці ai_optimization_log та ai_semantic_cluster створені (або вже існують).")

def _print_stats(label: str, stats: dict) -> None:
    print(f"\n{'='*50}")
    print(f"    Результати: {label}")
    print(f"{'='*50}")
    print(f"    Всього:      {stats['total']}")
    print(f"    Успішно:     {stats['success']}")
    print(f"    Пропущено:   {stats['skipped']}")
    print(f"    Помилки:     {stats['errors']}")
    print(f"    Токенів:     {stats['tokens']:,}")
    print(f"    Вартість:    ~${stats['cost_usd']:.4f}")
    print(f"    Провайдер:   {stats['provider']}")
    print(f"{'='*50}\n")

def build_parser() -> argparse.ArgumentParser:
    parser = argparse.ArgumentParser(
        prog="fionahome_seo_ai",
        description="ШІ-система SEO-оптимізації для fionahome.com.ua",
    )
    sub = parser.add_subparsers(dest="command", required=True)

    p_prod = sub.add_parser("products", help="Оптимізувати товари")
    p_prod.add_argument("--limit", type=int, default=0,
                        help="Максимальна кількість товарів (0 = всі)")
    p_prod.add_argument("--id", type=int, default=None,
                        help="Обробити лише один товар за product_id")
    p_prod.add_argument("--force", action="store_true",
                        help="Повторно обробляти вже оптимізовані товари")
    p_prod.add_argument("--dry-run", action="store_true",
                        help="Показати результат без запису у БД")
    p_prod.set_defaults(func=cmd_products)

    p_cat = sub.add_parser("categories", help="Оптимізувати категорії")
    p_cat.add_argument("--limit", type=int, default=0)

```

```

p_cat.add_argument("--force", action="store_true")
p_cat.add_argument("--dry-run", action="store_true")
p_cat.set_defaults(func=cmd_categories)

p_cl = sub.add_parser("cluster", help="Семантична кластеризація запитів")
p_cl.add_argument("--input", type=str, default=None,
                  help="Текстовий файл із запитамі (по одному на рядок)")
p_cl.add_argument("--k", type=int, default=None,
                  help="Кількість кластерів (None = авто)")
p_cl.add_argument("--save", action="store_true",
                  help="Зберегти результати у таблицю
ai_semantic_cluster")
p_cl.set_defaults(func=cmd_cluster)
p_st = sub.add_parser("stats", help="Статистика оптимізацій")
p_st.set_defaults(func=cmd_stats)
p_db = sub.add_parser("setup-db", help="Створити розширені таблиці у БД")
p_db.set_defaults(func=cmd_setup_db)
return parser
if __name__ == "__main__":
    parser = build_parser()
    args = parser.parse_args()
    args.func(args)

```

Preview.py:

```

import argparse
import sys
import os
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))

from dotenv import load_dotenv
load_dotenv()

from config.settings import AI_PROVIDER, LANGUAGE_ID
from db.connector import DatabaseConnector
from db.queries import get_product_full, get_products_for_optimization

def _get_generator():
    if AI_PROVIDER == "claude":
        from ai.claude_generator import ClaudeContentGenerator
        return ClaudeContentGenerator()
    elif AI_PROVIDER == "gemini":
        from ai.gemini_generator import GeminiContentGenerator
        return GeminiContentGenerator()
    else:
        from ai.content_generator import ContentGenerator
        return ContentGenerator()

def show_product(product_id: int = None, limit: int = 1):
    gen = _get_generator()

    with DatabaseConnector() as db:
        if product_id:

```

```

        products = [get_product_full(db, product_id)]
    else:
        products = get_products_for_optimization(db, limit=limit,
skip_optimized=False)
        # Завантажуємо атрибути
        from db.queries import get_product_attributes
        for p in products:
            p["attributes"] = get_product_attributes(db, p["product_id"])

for product in products:
    if not product:
        print("Товар не знайдено.")
        continue

    pid = product["product_id"]
    name = product.get("product_name", "")

    print("\n" + "=" * 70)
    print(f"  ТОВАР: {name} (id={pid})")
    print(f"  Категорія: {product.get('category', '-')}")
    print(f"  Виробник: {product.get('manufacturer', '-')}")
    attrs = product.get("attributes", {})
    if attrs:
        print(f"  Атрибути: {'; '.join(f'{k}: {v}' for k,v in
attrs.items())}")
    print("=" * 70)

    print("\n└ БУЛО (поточний контент у БД) " + "-" * 37)

    old_title = product.get("meta_title", "") or ""
    old_desc = product.get("meta_description", "") or ""
    old_text = product.get("existing_desc", "") or ""

    # Прибираємо HTML-теги для читабельності
    import re
    clean = lambda s: re.sub(r"<[^>]+>", "", s).replace("<",
"<").replace(">", ">").replace("&", "&").strip()

    old_text_clean = clean(old_text)
    words_before = len(old_text_clean.split()) if old_text_clean else 0

    print(f"| Meta title ({len(old_title)} симв.): {old_title or '▲
порожньо'}")
    print(f"| Meta desc  ({len(old_desc)} симв.): {old_desc or '▲
порожньо'}")
    print(f"| Опис ({words_before} сл.):")
    if old_text_clean:
        # Перші 300 символів
        preview = old_text_clean[:300] + ("..." if len(old_text_clean) >
300 else "")
        for line in preview.split("\n")[:5]:
            if line.strip():
                print(f"| {line.strip()}")
    else:

```

```

print("|   ⚠   порожньо")

print("\n└─ СТАНЕ (після ШІ-оптимізації) " + "-" * 37)
print("  Генерація контенту через ШІ...\n")

new_desc  = gen.generate_product_description(product)
new_title = gen.generate_meta_title(product)
new_meta  = gen.generate_meta_description(product)

words_after = len(new_desc.split())

print(f"  Meta title ({len(new_title)} симв.): {new_title}")
print(f"  Meta desc  ({len(new_meta)} симв.): {new_meta}")
print(f"  Опис  ({words_after} сл.):")
print()
# Друкуємо весь опис з відступом
for line in new_desc.split("\n"):
    if line.strip():
        print(f"      {line.strip()}")

print("\n" + "-" * 70)
print("  ПОРІВНЯННЯ:")
print(f"  Опис:          {words_before} сл. → {words_after} сл.")
print(f"  Meta title: {len(old_title)} симв. → {len(new_title)} симв.")
print(f"  Meta desc:  {len(old_desc)} симв. → {len(new_meta)} симв.")
print("-" * 70)

```

```

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Попередній перегляд SEO-оптимізації")
    parser.add_argument("--id",      type=int, default=None, help="product_id товару")
    parser.add_argument("--limit",   type=int, default=1,   help="Кількість товарів")
    args = parser.parse_args()
    show_product(product_id=args.id, limit=args.limit)

```

Prompts.py:

```

PRODUCT_DESCRIPTION_PROMPT = """
Напиши SEO-оптимізований опис товару для інтернет-магазину домашнього текстилю FionaHome.

```

```

Товар: {product_name}
Категорія: {category}
Виробник: {manufacturer}
Атрибути: {attributes}
Існуючий опис (якщо є): {existing_desc}

```

Вимоги:

- Мова: виключно українська
- Обсяг: 180-250 слів

- Структура: 3 абзаци
 - 1) Загальна характеристика товару – природно вкажи назву у першому реченні
 - 2) Матеріал, склад, властивості
 - 3) Переваги та сфера застосування; закінчи закликом до дії
 - Природне включення ключових слів (щільність не більше 2-3%)
 - НЕ використовуй слова: "чудовий", "неперевершений", "ідеальний", "найкращий", "унікальний"
 - НЕ використовуй маркований або нумерований список
 - Стиль: інформативний, довірливий, не рекламний
- "".strip()

PRODUCT_META_DESCRIPTION_PROMPT = ""

Склади мета-опис (meta description) для сторінки товару інтернет-магазину FionaHome.

Товар: {product_name}
 Категорія: {category}
 Атрибути: {attributes}

Вимоги:

- Мова: українська
 - Довжина: 140-160 символів (це критично важливо!)
 - Включи заклик до дії (наприклад: "Замовляйте онлайн", "Купуйте вигідно")
 - Природно включи 1-2 ключових слова
 - НЕ використовуй лапки та спеціальні символи
 - Повертай ТІЛЬКИ текст мета-опису, без пояснень
- "".strip()

CATEGORY_META_DESCRIPTION_PROMPT = ""

Склади мета-опис (meta description) для сторінки категорії інтернет-магазину FionaHome.

Категорія: {category_name}
 Кількість товарів у категорії: {product_count}
 Підкатегорії / типи товарів: {subcategories}

Вимоги:

- Мова: українська
 - Довжина: 140-160 символів (суворо дотримуйся!)
 - Включи заклик до дії
 - Природно включи назву категорії
 - Повертай ТІЛЬКИ текст мета-опису
- "".strip()

CATEGORY_DESCRIPTION_PROMPT = ""

Напиши SEO-оптимізований текстовий опис для сторінки категорії інтернет-магазину FionaHome.

Категорія: {category_name}
 Типові товари в категорії: {product_examples}
 Ключові атрибути товарів: {key_attributes}

Вимоги:

- Мова: українська
 - Обсяг: 120-180 слів
 - Структура: 2 абзаци
 - 1) Що входить до категорії, для кого підходить
 - 2) Переваги покупки в FionaHome + заклик до дії
 - Природне вживання назви категорії
 - НЕ використовуй слова: "чудовий", "ідеальний", "найкращий", "унікальний"
 - НЕ використовуй список
- "".strip()

PRODUCT_TITLE_TEMPLATE = "{keyword} – купити в Україні | {store}"

CATEGORY_TITLE_TEMPLATE = "{category_name} – купити онлайн | {store}"

```
SYSTEM_MESSAGE = (
    "Ти – досвідчений SEO-копірайтер для українського інтернет-магазину "
    "домашнього текстилю FionaHome (fionahome.com.ua). "
    "Пишеш виключно українською мовою. "
    "Твій стиль: чіткий, інформативний, без надмірної реклами. "
    "Точно дотримуйся вимог щодо обсягу та структури."
)
```

Settings.py:

```
import os
from dotenv import load_dotenv

load_dotenv()

DB_HOST      = os.getenv("DB_HOST", "localhost")
DB_PORT      = int(os.getenv("DB_PORT", 3306))
DB_NAME      = os.getenv("DB_NAME", "fionahome")
DB_USER      = os.getenv("DB_USER", "root")
DB_PASSWORD  = os.getenv("DB_PASSWORD", "")
DB_CHARSET   = os.getenv("DB_CHARSET", "utf8mb4")

# Префікс таблиць OpenCart (стандартний – "oc_"; у вас – "fdb_")
TABLE_PREFIX = os.getenv("TABLE_PREFIX", "fdb_")

# language_id для української мови в OpenCart
LANGUAGE_ID = int(os.getenv("LANGUAGE_ID", 5))  # у fionahome українська =
language_id 5

OPENAI_API_KEY = os.getenv("OPENAI_API_KEY", "")
OPENAI_MODEL   = os.getenv("OPENAI_MODEL", "gpt-4o-mini")  # gpt-4o або
gpt-4o-mini
OPENAI_MAX_TOKENS_DESC = int(os.getenv("OPENAI_MAX_TOKENS_DESC", 700))
OPENAI_MAX_TOKENS_META = int(os.getenv("OPENAI_MAX_TOKENS_META", 200))
```

```

OPENAI_TEMPERATURE      = float(os.getenv("OPENAI_TEMPERATURE", 0.7))

GEMINI_API_KEY   = os.getenv("GEMINI_API_KEY", "")
GEMINI_MODEL     = os.getenv("GEMINI_MODEL", "gemini-2.0-flash")
# "openai" або "gemini"
AI_PROVIDER = os.getenv("AI_PROVIDER", "gemini")

SLEEP_BETWEEN_REQUESTS = float(os.getenv("SLEEP_BETWEEN_REQUESTS", 0.5))

# Кількість товарів за один запуск (0 = без ліміту)
BATCH_LIMIT = int(os.getenv("BATCH_LIMIT", 0))

# Пропускати товари, що вже успішно оптимізовані (рекомендовано True)
SKIP_ALREADY_OPTIMIZED = os.getenv("SKIP_ALREADY_OPTIMIZED", "true").lower()
== "true"

LOG_DIR   = os.getenv("LOG_DIR", "logs")
LOG_LEVEL = os.getenv("LOG_LEVEL", "INFO")      # DEBUG | INFO | WARNING | ERROR

DESC_MIN_WORDS = int(os.getenv("DESC_MIN_WORDS", 150))
DESC_MAX_WORDS = int(os.getenv("DESC_MAX_WORDS", 300))
TITLE_MIN_CHARS = int(os.getenv("TITLE_MIN_CHARS", 40))
TITLE_MAX_CHARS = int(os.getenv("TITLE_MAX_CHARS", 65))
META_MIN_CHARS  = int(os.getenv("META_MIN_CHARS", 130))
META_MAX_CHARS  = int(os.getenv("META_MAX_CHARS", 165))

# Заборонені шаблонні слова в згенерованому тексті
BANNED_PHRASES = [
    "чудовий", "неперевершений", "ідеальний", "найкращий",
    "унікальний", "неймовірний", "фантастичний",
]

SEMANTIC_MODEL_NAME = os.getenv(
    "SEMANTIC_MODEL_NAME", "intfloat/multilingual-e5-large"
)
CLUSTER_MAX_K       = int(os.getenv("CLUSTER_MAX_K", 20))
CLUSTER_MIN_QUERIES = int(os.getenv("CLUSTER_MIN_QUERIES", 3))

GSC_CREDENTIALS_FILE = os.getenv("GSC_CREDENTIALS_FILE",
    "gsc_credentials.json")
GSC_SITE_URL         = os.getenv("GSC_SITE_URL",
    "https://fionahome.com.ua/")

STORE_NAME   = "FionaHome"
STORE_DOMAIN = "fionahome.com.ua"

```

```
ANTHROPIC_API_KEY = os.getenv("ANTHROPIC_API_KEY", "")
CLAUDE_MODEL      = os.getenv("CLAUDE_MODEL", "claude-haiku-4-5-20251001")
```

Connector.py:

```
import time
import pymysql
import pymysql.cursors
from typing import Any, Optional

from config.settings import (
    DB_HOST, DB_PORT, DB_NAME, DB_USER, DB_PASSWORD, DB_CHARSET
)
from utils.logger import get_logger

logger = get_logger(__name__)

class DatabaseConnector:
    """
    Обгортка над PyMySQL.
    Підтримує:
    - параметризовані запити (захист від SQL-ін'єкцій)
    - авто-перепідключення при втраті з'єднання
    - повернення результатів у вигляді словників
    """

    def __init__(
        self,
        host: str      = DB_HOST,
        port: int       = DB_PORT,
        database: str   = DB_NAME,
        user: str        = DB_USER,
        password: str   = DB_PASSWORD,
        charset: str    = DB_CHARSET,
        autocommit: bool = False,
        retries: int     = 3,
        retry_delay: float = 2.0,
    ):
        self._config = {
            "host":      host,
            "port":      port,
            "db":        database,
            "user":      user,
            "password":  password,
            "charset":   charset,
            "autocommit": autocommit,
            "connect_timeout": 10,
            "cursorclass": pymysql.cursors.DictCursor,
        }
        self._retries      = retries
        self._retry_delay  = retry_delay
        self._conn         = None
        self._connect()
```

```

def _connect(self) -> None:
    for attempt in range(1, self._retries + 1):
        try:
            self._conn = pymysql.connect(**self._config)
            logger.debug(
                "MariaDB: підключено до %s@%s:%s",
                self._config["db"],
                self._config["host"],
                self._config["port"],
            )
            return
        except pymysql.Error as exc:
            logger.warning(
                "MariaDB: спроба %d/%d не вдалась: %s",
                attempt, self._retries, exc
            )
            if attempt < self._retries:
                time.sleep(self._retry_delay)
    raise ConnectionError(
        f"Не вдалось підключитись до MariaDB після {self._retries}
спроб.\n"
        f"Перевір DB_HOST, DB_PORT, DB_USER, DB_PASSWORD у файлі .env"
    )

def _ensure_connected(self) -> None:
    try:
        if self._conn is None:
            self._connect()
            return
        self._conn.ping(reconnect=True)
    except Exception:
        self._connect()

def close(self) -> None:
    if self._conn:
        try:
            self._conn.close()
        except Exception:
            pass
        logger.debug("MariaDB: з'єднання закрито.")

def fetchall(self, query: str, params: tuple = ()) -> list[dict]:
    self._ensure_connected()
    with self._conn.cursor() as cursor:
        cursor.execute(query, params)
        return cursor.fetchall()

def fetchone(self, query: str, params: tuple = ()) -> Optional[dict]:
    self._ensure_connected()
    with self._conn.cursor() as cursor:
        cursor.execute(query, params)
        return cursor.fetchone()

```

```

    def fetchval(self, query: str, params: tuple = (), default: Any = None) -
> Any:
        row = self.fetchone(query, params)
        if row is None:
            return default
        return next(iter(row.values()), default)

    def execute(self, query: str, params: tuple = ()) -> int:
        self._ensure_connected()
        with self._conn.cursor() as cursor:
            try:
                cursor.execute(query, params)
                self._conn.commit()
                return cursor.rowcount
            except Exception as exc:
                self._conn.rollback()
                logger.error("execute error: %s\nQuery: %s", exc, query)
                raise

    def executemany(self, query: str, params_list: list[tuple]) -> int:
        if not params_list:
            return 0
        self._ensure_connected()
        with self._conn.cursor() as cursor:
            try:
                cursor.executemany(query, params_list)
                self._conn.commit()
                return cursor.rowcount
            except Exception as exc:
                self._conn.rollback()
                logger.error("executemany error: %s", exc)
                raise

    def lastrowid(self, query: str, params: tuple = ()) -> Optional[int]:
        self._ensure_connected()
        with self._conn.cursor() as cursor:
            try:
                cursor.execute(query, params)
                self._conn.commit()
                return cursor.lastrowid
            except Exception as exc:
                self._conn.rollback()
                logger.error("lastrowid error: %s", exc)
                raise

    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.close()
        return False

```

