МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ГЕНЕРАЦІЇ ВЕБКОНТЕНТУ ЗАСОБАМИ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ»**

Виконав: студент 2 курсу, групи 8.1214-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Я.В. Лісаченко
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Кривохата А.Г.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Лісаченку Ярославу Володимировичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка застосунку для генерації вебконтенту засобами великих мовних моделей

керівник роботи Кривохата Анастасія Григорівна, к.ф.-м.н.
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 28 » лютого 2025 року № 326-с

2. Строк подання студентом роботи 04.12.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Теоретичний огляд генерації вебконтенту та аналіз існуючих рішень.
2. Проєктування та розробка вебзастосунку для генерації вебконтенту.
3. Програмна реалізація вебсервісу.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 07.03.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	18.03.2025	
2.	Збір вихідних даних.	17.04.2025	
3.	Обробка методичних та теоретичних	23.06.2025	
	джерел.		
4.	Розробка першого та другого розділу.	19.08.2025	
5.	Розробка третього розділу.	24.11.2025	
6	Оформлення та нормоконтроль	01.12.2025	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	18.12.2025	

Студент _____ Я.В. Лісаченко
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.Г. Кривохата
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка застосунку для генерації вебконтенту засобами великих мовних моделей»: 71 с., 15 рис., 12 джерел.

АВТОМАТИЗАЦІЯ, ВЕБКОНТЕНТ, ВЕЛИКІ МОВНІ МОДЕЛІ (LLMS), ГЕНЕРАЦІЯ КОНТЕНТУ, ЗАСТОСУНОК, NLP (ОБРОБКА ПРИРОДНОЇ МОВИ), ТОНКЕ НАЛАШТУВАННЯ, ШТУЧНИЙ ІНТЕЛЕКТ, ЯКІСТЬ ТЕКСТУ, API, OPENAI.

Об’єкт дослідження – процес створення, автоматизації та оптимізації різноманітних текстових вебматеріалів за допомогою сучасних інструментів штучного інтелекту.

Мета роботи: розробити програмний застосунок для генерації якісного та релевантного вебконтенту, що інтегрує можливості великих мовних моделей, забезпечує зручний користувацький інтерфейс та стабільну серверну логіку для автоматизації контент-стратегій.

Методи дослідження – методи обробки природної мови (NLP), технології веброзробки (проєктування клієнтської та серверної частин), методи інтеграції зовнішніх API (OpenAI та ін.), методи оцінювання якості текстового контенту (метрики якості, експертна перевірка), методи тестування програмного забезпечення.

У результаті виконання кваліфікаційної роботи отримано функціональний вебзастосунок для автоматизованої генерації текстів, який надає користувачам інструменти для створення, редагування та управління вебконтентом за допомогою штучного інтелекту. Результати роботи можуть бути використані як основа для подальшого розширення функціональності, підключення нових мовних моделей або інтеграції з системами управління контентом (CMS).

SUMMARY

Master’s qualifying paper “Development of an Application for the Web Content Generation Using Large Language Models”: 71 pages, 15 figures, 12 references.

AI, AUTOMATION, API, APPLICATION, ARTIFICIAL INTELLIGENCE, CONTENT GENERATION, FINE-TUNING, LARGE LANGUAGE MODELS (LLMS), NLP (NATURAL LANGUAGE PROCESSING), OPENAI, TEXT QUALITY, WEB CONTENT.

The object of the study is the process of creation, automation, and optimization of various textual web materials using modern artificial intelligence tools.

The aim of the study is to develop a software application for generating high-quality and relevant web content that integrates the capabilities of large language models, provides a convenient user interface, and ensures stable server-side logic for automating content strategies.

Research methods include natural language processing (NLP) methods, web development technologies (design of client-side and server-side components), methods of external API integration (OpenAI, etc.), methods for evaluating the quality of textual content (quality metrics, expert review), software testing methods.

As a result of the qualification work, a functional web application for automated text generation has been obtained, which provides users with tools for creating, editing, and managing web content using artificial intelligence. The results of the work can be used as a basis for further functional expansion, connection of new language models, or integration with content management systems (CMS).

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 7
1 Теоретичні основи генерації вебконтенту та аналіз існуючих рішень 9
1.1 Визначення, типи та особливості сучасного вебконтенту 10
1.2 Огляд великих мовних моделей (LLM) як інструменту генерації текстів 13
1.3 Аналіз існуючих сервісів та API для генерації контенту: можливості та обмеження 15
1.4 Архітектурні підходи до інтеграції LLM у вебзастосунки 17
1.5 Вимоги до якості, релевантності та унікальності згенерованого контенту 19
1.6 Правові та етичні аспекти використання ШІ 20
1.7 Висновки до розділу 1 21
2 Проєктування та розробка вебзастосунку для генерації контенту 23
2.1 Формування вимог до функціоналу та архітектури застосунку 23
2.2 Вибір та обґрунтування технологічного стеку 24
2.3 Проєктування схеми бази даних PostgreSQL 25
2.4 Розробка модулів взаємодії з WordPress REST API для публікації контенту 31
2.5 Висновки до розділу 2 39
3 Програмна реалізація вебсервісу 40
3.1 Інтеграція з CMS WordPress та системне тестування 40
3.2 Реалізація серверної частини 43
3.3 Реалізація клієнтської частини 54
Висновки 68
Перелік посилань 70

ВСТУП

Сучасний розвиток технологій штучного інтелекту та обробки природної мови відкриває нові горизонти для автоматизації створення вебконтенту. Однією з ключових задач є перетворення коротких текстових запитів у якісний, релевантний контент для вебсайтів та соціальних мереж. Прикладом таких рішень є платформи на кшталт Jasper.ai, Copy.ai або ChatGPT, які дозволяють генерувати статті, соціальні пости та рекламні тексти шляхом використання великих мовних моделей.

Метою кваліфікаційної роботи є проєктування та реалізація вебзастосунку для генерації вебконтенту з використанням Python та FastAPI для реалізації серверної частини та ML-мікросервісів, React та TypeScript для розробки клієнтського інтерфейсу, WordPress REST API для інтеграції з CMS-системами, а також PostgreSQL та Redis як систем зберігання даних.

Для досягнення поставленої мети сформульовано такі задачі:

- проаналізувати сучасні підходи та інструменти генерації контенту за допомогою великих мовних моделей;
- сформулювати функціональні та нефункціональні вимоги до вебзастосунку;
- спроєктувати архітектуру системи з чітким розмежуванням функціоналу;
- реалізувати основний API та бізнес-логіку на Python з використанням фреймворку FastAPI;
- розробити інтерфейс користувача з використанням React та TypeScript;
- створити модуль інтеграції з WordPress REST API для автоматичної публікації контенту;
- реалізувати систему кешування результатів генерації з використанням Redis;

– організувати зберігання та управління даними в PostgreSQL;
– виконати тестування окремих компонентів та провести інтеграційне тестування системи.

Структурно робота складається з трьох розділів. У першому розділі проаналізовано сучасні системи генерації контенту, їхні функціональні можливості та архітектурні рішення, а також обґрунтовано вибір технологічного стеку: Python та FastAPI для бекенду та ML-сервісів, React та TypeScript для інтуїтивного інтерфейсу, WordPress REST API для безперервної публікації контенту, а також PostgreSQL та Redis для надійного зберігання даних та підвищення продуктивності.

У другому розділі виконано аналіз функціональних і нефункціональних вимог до системи, спроєктовано архітектуру з використанням діаграм UML, розроблено схеми взаємодії між компонентами системи, а також спроєктовано структуру бази даних PostgreSQL та моделі кешування в Redis.

У третьому розділі реалізовано основні компоненти системи: серверну частину на Python з використанням FastAPI, клієнтський інтерфейс на React з TypeScript, модуль інтеграції з WordPress, а також інтегровано систему зберігання даних. Проведено комплексне тестування окремих модулів та системи в цілому, включаючи тестування продуктивності та якості згенерованого контенту.

1 ТЕОРЕТИЧНІ ОСНОВИ ГЕНЕРАЦІЇ ВЕБКОНТЕНТУ ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Сучасний цифровий простір характеризується стрімким зростанням потреби у якісному вебконтенті. Соціальні мережі, корпоративні сайти, інтернет-магазини та інформаційні портали потребують регулярного оновлення контенту, що ставить перед розробниками та маркетологами завдання оптимізації процесу його створення. У цьому контексті технології штучного інтелекту, зокрема великі мовні моделі, відкривають нові можливості для автоматизації генерації текстових матеріалів [1].

Актуальність дослідження полягає в необхідності розробки ефективних інструментів, які б поєднували у собі потужність сучасних AI-моделей, зручний інтерфейс та інтеграцію з популярними платформами для публікації контенту. Це дозволить значно скоротити витрати часу та ресурсів на створення вебконтенту, зберігаючи при цьому його якість та унікальність.

Метою даного розділу є систематизація теоретичних знань щодо процесу генерації вебконтенту, аналіз існуючих програмних рішень на ринку та обґрунтування архітектурних підходів до розробки власного вебзастосунку.

Для досягнення поставленої мети в розділі вирішуються наступні завдання:

– дослідити сутність, типи та особливість сучасного вебконтенту;
– проаналізувати великі мовні моделі як основний інструмент генерації текстів;
– дослідити архітектурні підходи для інтеграції LLM у вебзастосунки;
– сформулювати вимоги до якості та безпеки згенерованого контенту;
– провести порівняльний аналіз існуючих сервісів генерації контенту.

Результати дослідження, представлені в цьому розділі, лягуть в основу практичної реалізації вебзастосунку, опис якої наведено в наступних розділах роботи [1].

1.1 Визначення, типи та особливості сучасного вебконтенту

Сучасний вебконтент є динамічною та багатогранною категорією, що визначає інформаційне наповнення цифрового простору. У широкому розумінні вебконтент – це будь-яка інформація, представлена в цифровій формі та розрахована на поширення через Інтернет. Проте з розвитком цифрових технологій це поняття значно розширилося та ускладнилося.

Визначення сучасного вебконтенту охоплює не лише текстовий компонент, а й весь спектр мультимедійних елементів, що формують цілісний користувацький досвід. За сучасними підходами, вебконтент можна визначити як структурований цифровий об’єкт, що містить інформаційну, комунікаційну або розважальну цінність, оптимізований для поширення через вебканали та адаптований до специфіки різних платформ [1].

Основні типи вебконтенту включають:

– текстовий контент;
– візуальний контент;
– інтерактивний контент.

Текстовий контент є основним об’єктом автоматизації в нашому проєкті. Він класифікується за трьома ключовими параметрами: обсягом (короткі пости, середні статті, довгі матеріали), функцією (інформаційний, комерційний, розважальний) та платформою призначення (сайти, соцмережі, блоги) [1].

У нашій системі реалізовано спеціалізовані модулі генерації для кожного типу контенту:

– для соціальних мереж – створення коротких engaging-постів;
– для блогів – написання SEO-оптимізованих статей;
– для інтернет-магазинів – генерація продуктивних описів;
– для landing pages – створення продаючих текстів.

Технологічний підхід включає:

– Fine-tuned LLM з спеціалізованими шаблонами;
– автоматичну SEO-оптимізацію;

– контроль якості (унікальність 95%+, читабельність);
– адаптацію стилю під бренд;
– миттєву публікацію через WordPress REST API.

Система забезпечує комплексну підтримку повного циклу – від генерації тексту до його публікації, з автоматичним форматуванням та оптимізацією під конкретні цілі та платформи [2].

Візуальний контент є невід'ємною частиною сучасного вебсередовища, що значно покращує сприйняття інформації та підвищує ефективність комунікації. Дослідження підтверджують, що матеріали з якісними візуальними елементами збільшують залученість користувачів на 47% порівняно з виключно текстовими форматами. У нашому проєкті візуальний контент інтегрований як ключовий компонент системи генерації вебконтенту, де він виконує кілька важливих функцій [2].

Основне завдання візуального контенту в нашій системі – доповнення та покращення текстових матеріалів. Ми реалізували модуль генерації та підбору зображень, який автоматично створює ілюстрації до статей, інфографіку для представлення складних даних, а також баннери для соціальних мереж. Для технічної реалізації використовується комбінація AI-генерації зображень за текстовими запитами та інтелектуального підбору з готових бібліотек. Особливу увагу приділено оптимізації – всі зображення автоматично стискаються у формати WebP та AVIF без втрати якості, що значно покращує швидкість завантаження вебсторінок [2].

Система також включає інструменти для автоматичного створення альтернативних текстових описів, що покращує доступність контенту для користувачів з порушенням зору та сприяє SEO-оптимізації. Для кожного згенерованого зображення створюється семантична розмітка, яка допомагає пошуковим системам коректно індексувати візуальний контент. Крім того, наш модуль аналітики відстежує ефективність різних типів візуального контенту, проводячи A/B тестування та надаючи рекомендації щодо оптимізації на основі даних про взаємодію користувачів [2].

Список ключових функцій модуля візуального контенту:

– генерація унікальних зображень за текстовими описами;
– автоматичне створення інфографіки на основі даних;
– підбір релевантних фотографій з stock-бібліотек;
– оптимізація розмірів та формату для різних платформ;
– створення альтернативних текстів та метаданих;
– аналітика ефективності візуального контенту.

Інтеграція візуального контенту в загальну архітектуру системи забезпечує узгоджену роботу всіх компонентів, де текстові та графічні елементи доповнюють один одного, створюючи цілісний і привабливий контент для кінцевого користувача [2].

Інтерактивний контент – це тип вебматеріалів, який залучає користувача до активних дій: відповідей, вибору, створення власного контенту. Він перетворює пасивне споживання інформації на діалог, що підвищує залученість та конверсію.

У нашому проєкті інтерактивний контент реалізований через:

– генерацію опитувань і тестів;
– створення інтерактивних калькуляторів;
– модуль для A/B тестування контенту;
– інструменти для створення інтерактивних інфографік.

Ці елементи інтегровані в загальну систему через спеціальні шаблони та API, що дозволяє автоматично генерувати інтерактивні компоненти на основі вхідних даних. Наприклад, система може створити калькулятор вартості послуг на основі введених цін або згенерувати тест для перевірки знань на базі навчального матеріалу [2].

Для реалізації використовуються сучасні вебтехнології, що забезпечують крос-платформенність та швидкість роботи. Всі інтерактивні елементи автоматично адаптуються під мобільні пристрої та інтегруються з системою аналітики для відстеження ефективності [3].

1.2 Огляд великих мовних моделей (LLM) як інструменту генерації текстів

Великі мовні моделі (Large Language Models, LLM) становлять основу сучасних систем генерації текстового контенту. Це нейромережні архітектури, натреновані на величезних масивах текстових даних, здатні створювати людоподібні тексти, перекладати мови, стисло викладати інформацію та виконувати інші мовні завдання [3].

Ключові архітектури LLM:

– GPT (Generative Pre-trained Transformer);
– BERT (Bidirectional Encoder Representations) T5 (Text-to-Text Transfer Transformer);
– XLNet.

GPT являє собою архітектуру генеративних попередньо навчених трансформерів, яка становить основне технологічне ядро нашої системи генерації контенту. Ця нейромережна модель використовує механізм уваги для аналізу контексту та створення послідовних і логічно зв’язаних текстів. Особливістю GPT є її здатність до попереднього навчання на великих масивах текстових даних з подальшим донавчанням для вирішення специфічних завдань.

У даному проєкті GPT інтегрована як основний інструмент для створення різноманітних типів вебконтенту. Модель відповідає за генерацію унікальних текстів для блогів та інформаційних статей, створення комерційних описів продуктів, написання постів для соціальних мереж, а також формування мета-описів та SEO-заголовків. Для забезпечення максимальної ефективності ми провели fine-tuning моделі під специфічні потреби нашого застосунку, налаштувавши її стиль відповідно до вимог бренду та особливостей цільової аудиторії [3].

Технічна інтеграція GPT в систему реалізована через API-взаємодію з хмарними сервісами, що дозволяє використовувати найсучасніші версії моделі

без необхідності їх локального розгортання. Для підвищення швидкодії реалізовано систему кешування результатів, що особливо важливо при обробці великих обсягів запитів. Контроль якості згенерованого контенту забезпечується через вбудовані механізми фільтрації та модерування, перевірку унікальності текстів, корекцію стилістичних помилок та оптимізацію читабельності [3].

Ключові напрями використання GPT у проєкті:

– генерація основних текстових матеріалів (статті, описи, пости);
– створення SEO-оптимізованих елементів (мета-теги, заголовки);
– адаптація контенту під різні платформи та аудиторії;
– забезпечення узгодженості стилю та тону викладу;
– оптимізація контенту для підвищення конверсії.

Завдяки інтеграції GPT наша система здатна швидко та ефективно створювати якісний контент, адаптований під конкретні потреби користувачів та готовий до безпосередньої публікації в вебсередовищі [3].

BERT (Bidirectional Encoder Representations) є двонаправленою моделлю, яка аналізує контекст з обох боків слова, що забезпечує глибоке розуміння семантики тексту. У проєкті вона відповідає за класифікацію та категоризацію контенту, аналіз тональності текстів, витягування ключових слів і фраз, а також покращення якості семантичного аналізу. Це дозволяє системі точно визначати тематику матеріалів, оцінювати їхній емоційний забарвлення та ідентифікувати основні поняття.

T5 (Text-to-Text Transfer Transformer) реалізує уніфікований підхід, перетворюючи всі NLP-завдання на текст-в-текст формат. У нашій архітектурі вона виконує функції перефразування та рерайтингу текстів, створення стислих викладок довгих матеріалів, перетворення структурованих даних у текстовий формат та вирішення різноманітних завдань через єдиний інтерфейс. Це забезпечує гнучкість обробки контенту та дозволяє адаптувати тексти під різні цілі та платформи [3].

Обидві моделі інтегровані в єдину архітектуру разом з GPT, формуючи комплексну систему обробки природної мови. BERT виступає як аналітичне

ядро для розуміння контенту, T5 відповідає за трансформацію текстів, а GPT забезпечує генерацію нового контенту. Така комбінація дозволяє створювати якісний та релевантний вебконтент, адаптований під конкретні потреби користувачів.

XLNet є передовою моделлю, яка поєднує переваги авторегресійних методів, як у GPT, з двонаправленим контекстуальним аналізом, характерним для BERT. Це досягається завдяки використанню методу переставного мовного моделювання, де модель вчиться передбачати слова в довільному порядку, враховуючи весь доступний контекст [4].

У розроблюваній системі XLNet відіграє ключову роль у завданнях, що вимагають глибокого розуміння контексту та генерації когерентних текстів. Модель особливо ефективна для створення довгих і структурованих матеріалів, таких як статті, огляди та детальні описи, де важлива послідовність викладу та логічні зв’язки між частинами тексту.

XLNet також використовується для покращення якості контенту шляхом перефразування та оптимізації існуючих текстів, забезпечуючи кращу читабельність та природність мови. Завдяки здатності аналізувати складні контекстні залежності, модель допомагає уникнути повторень та підвищити інформаційну щільність генерованих матеріалів [4].

1.3 Аналіз існуючих сервісів та API для генерації контенту: можливості та обмеження

Сучасний ринок AI-генерації контенту стрімко розвивається, пропонуючи широкий спектр API та сервісів, кожен з яких має унікальні характеристики та спеціалізацію. Детальний аналіз цих рішень дозволяє визначити оптимальну архітектуру для нашої системи та вибрати найефективніші інструменти для різних типів завдань [4].

OpenAI GPT API представляє один з найпотужніших інструментів для генерації текстового контенту. Його основними перевагами є висока якість

генерації, підтримка різноманітних стилів та форматів, а також регулярні оновлення моделей. Однак сервіс має і суттєві недоліки: висока вартість при інтенсивному використанні, обмеження на довжину відповіді та залежність від стабільності зовнішнього сервісу. В нашому проєкті GPT API використовується для основних завдань генерації контенту, де потрібна висока креативність та різноманітність стилів.

Claude API від Anthropic пропонує унікальні можливості для роботи з довгими текстами та складними завданнями, що вимагають глибокого аналізу контексту. Ключовою перевагою Claude є здатність ефективно обробляти великі обсяги тексту (до 100K токенів), що робить його ідеальним рішенням для створення довгих статей, аналітичних матеріалів та технічної документації. У нашій системі Claude API відіграє вирішальну роль у завданнях, що вимагають глибокого розуміння контексту та роботи з великими документами. Його використання дозволяє створювати контент з послідовною аргументацією та логічною структурою [4].

Google PaLM API інтегрований в нашу систему для завдань, що вимагають пошуку та аналізу актуальної інформації. Його перевагою є тісна інтеграція з екосистемою Google, що забезпечує доступ до найсвіжіших даних та знань. Однак обмеженням є менша гнучкість у налаштуванні стилю порівняно з іншими моделями [5].

Jasper.ai та Copy.ai представляють готові рішення для маркетингового контенту з оптимізованими шаблонами та інтуїтивними інтерфейсами. Незважаючи на їхню зручність, ці сервіси мають обмежену можливість кастомізації та вищі вартісні показники при масштабуванні.

Переваги нашої архітектури:

– гібридне використання кількох API для максимізації ефективності;
– інтелектуальна маршрутизація завдань до найбільш підходящої моделі;
– власна система кешування та оптимізації запитів;
– українськомовна оптимізація та підтримка специфічного культурного контексту;

– розширені можливості налаштування стилю та тону контенту;
– інтеграція з системами перевірки якості та антиплагіату;
– можливість обробки специфічних форматів вебконтенту.

Система автоматично вибирає найбільш ефективний API для кожного типу завдання. Для коротких маркетингових текстів використовується GPT-4, для довгих аналітичних матеріалів – Claude API, а для завдань, що вимагають актуальної інформації – Google PaLM. Такий підхід дозволяє досягти оптимального співвідношення якості та вартості [5].

Додатково ми реалізували власний шару оптимізації, який включає систему кешування часто використовуваних шаблонів, механізми покращення якості згенерованого контенту та інструменти для адаптації під специфічні потреби україномовних користувачів. Це дозволяє нам пропонувати більш спеціалізоване рішення порівняно з універсальними сервісами.

Важливим конкурентним перевагою є також розробка власних алгоритмів контролю якості, які аналізують згенерований контент на предмет відповідності SEO-вимогам, унікальності та релевантності пошуковим запитам. Це забезпечує стабільно високу якість контенту, що генерується системою [5].

1.4 Архітектурні підходи до інтеграції LLM у вебзастосунки

Інтеграція великих мовних моделей у вебзастосунки вимагає ретельної архітектурної проробки для забезпечення ефективності, масштабованості та безпеки. Розглянемо основні підходи та їх реалізацію в нашому проєкті [5].

Мікросервісна архітектура є основним підходом у нашому рішенні. Ми виокремлюємо функціонал роботи з LLM в окремі сервіси, що дозволяє незалежно масштабувати компоненти та використовувати різні моделі для специфічних завдань. Кожен мікросервіс відповідає за певний тип операцій: генерація контенту, перефразування, аналіз тональності, перевірка якості [5].

API Gateway виступає єдиною точкою входу для всіх запитів, забезпечуючи маршрутизацію, аутентифікацію та балансування навантаження.

Цей підхід дозволяє ефективно керувати потоками даних та забезпечує безпеку системи.

Асинхронна обробка запитів реалізована через черги повідомлень (Redis/RabbitMQ), що дозволяє обробляти довготривалі операції генерації контенту без блокування основного потоку роботи застосунку. Користувач отримує миттєву відповідь про прийняття запиту до обробки, а результати надходять асинхронно через WebSocket з'єднання [5].

Кешування результатів на різних рівнях архітектури дозволяє значно знизити навантаження на LLM API та покращити швидкодію застосунку. Ми використовуємо Redis для кешування частовикористовуваних запитів та результатів, що дозволяє уникнути повторної генерації ідентичного контенту.

Стратегія fallback забезпечує стабільність роботи системи навіть при тимчасовій недоступності основних LLM API. При виявленні проблем з основним провайдером система автоматично перемикається на альтернативні джерела, зберігаючи при цьому якість контенту.

Оптимізація витрат досягається через інтелектуальне управління запитами. Система аналізує складність запиту та обирає найбільш економічно ефективну модель для його обробки, враховуючи вартість токенів та якість результатів [5].

Моніторинг та аналітика інтегровані на всіх рівнях архітектури. Збирається детальна статистика використання моделей, часу відповіді, якості згенерованого контенту, що дозволяє оптимізувати роботу системи в реальному часі.

Безпека та конфіденційність забезпечуються через шифрування даних, контроль доступу та анонімізацію інформації перед передачею до зовнішніх API. Важливі дані користувачів не зберігаються в логах зовнішніх сервісів.

Така архітектура дозволяє ефективно інтегрувати потужність LLM у вебзастосунок, забезпечуючи високу продуктивність, масштабованість та надійність при мінімальних витратах на інфраструктуру [5].

1.5 Вимоги до якості, релевантності та унікальності згенерованого контенту

Якість контенту визначається комплексом критеріїв, що забезпечують його професійний рівень та корисність для користувача. Перш за все, контент має відповідати вимогам граматичної та стилістичної коректності. Тексти повинні мати чітку логічну структуру, послідовність викладу та відповідати нормам української мови. Особливу увагу приділяємо природності мови – згенеровані тексти не повинні мати ознак «машинного» стилю, незв'язних переходів або логічних суперечностей [6].

Релевантність контенту передбачає його точну відповідність початковому запиту та очікуванням користувача. Для досягнення цієї мети ми реалізували систему семантичного аналізу, яка перевіряє відповідність контенту заданій темі, ключовим словам та цільовій аудиторії. Важливим аспектом є актуальність інформації – система автоматично перевіряє дати, статистичні дані та факти, використовуючи інтегровані бази знань.

Унікальність контенту є критично важливим параметром для SEO-оптимізації та уникнення проблем з плагіатом. Наша система забезпечує мінімальний рівень унікальності 95% за алгоритмами шинглування. Для досягнення цієї мети використовуються передові техніки парафразування, семантичного перетворення та синтезу нових ідей на основі аналізу multiple джерел [6].

Технічні параметри якості включають оптимальну довжину текстів відповідно до їх типу та призначення. Для SEO-статей це 1500-2500 слів, для блог-постів – 800-1200 слів, для соціальних мереж – 150-500 символів. Щільність ключових слів підтримується на рівні 1-1.5% з природним розподілом по тексту [6].

Критерії перевірки якості реалізовані у вигляді багаторівневої системи валідації. Перший рівень включає автоматичну перевірку за допомогою NLP-алгоритмів, які аналізують читабельність (індекс 60-70 балів), тональність,

наявність водних слів. Другий рівень передбачає семантичний аналіз на відповідність тематиці та цілям користувача. Третій рівень – експертна перевірка складних кейсів.

Вимоги до форматування включають коректне використання HTML-розмітки, заголовків різних рівнів, маркованих списків та інших структурних елементів. Система автоматично оптимізує контент для різних платформ публікації, враховуючи їх специфічні вимоги до формату [6].

Відповідність стандартам передбачає дотримання принципів E-A-T (Expertise, Authoritativeness, Trustworthiness) для створення експертного контенту, що викликає довіру у користувачів та пошукових систем. Контент має містити переконливі докази, авторитетні джерела та практичну цінність.

Ці вимоги реалізовані у вигляді єдиної системи контролю якості, яка постійно вдосконалюється на основі аналітики ефективності контенту та зворотного зв’язку від користувачів [6].

1.6 Правові та етичні аспекти використання ШІ

Використання технологій штучного інтелекту для створення контенту супроводжується низкою правових та етичних викликів, які потребують ретельного аналізу та врахування в нашому проєкті.

Правові аспекти передусім стосуються авторських прав на згенерований контент. Відповідно до сучасного законодавства ЄС та України, твори, створені за допомогою ШІ, можуть не визнаватися об’єктами авторського права, оскільки вони не є результатом творчої діяльності людини. Проте наш проєкт передбачає значну участь людини у процесі створення контенту через редагування, коректуру та затвердження матеріалів, що дозволяє захистити права на згенерований контент [6].

Відповідальність за контент є критично важливим аспектом. Ми реалізували багаторівневу систему модерації, яка включає автоматичні фільтри для виявлення неприйнятного контенту, порушень закону та misinformation.

Користувач несе остаточну відповідальність за опублікований контент, що зафіксовано в користувальницькій угоді [6].

Захист персональних даних відповідно до ЗУ «Про захист персональних даних» та GDPR гарантується через анонімізацію вхідних даних, шифрування інформації та обмеження зберігання історії запитів. Система не використовує персональні дані користувачів для тренування моделей без явної згоди.

Етичні принципи реалізовані через систему обмежень та фільтрів, що запобігають генерації:

– дискримінаційного та образливого контенту;
– деструктивної та небезпечної інформації;
– матеріалів, що порушують права третіх осіб;
– контенту, спрямованого на маніпуляцію свідомістю.

Прозорість та розпізнавання AI-контенту є ключовим принципом нашого проєкту. Ми додаємо маркери до згенерованого контенту та надаємо користувачам інструменти для інформування аудиторії про використання ШІ. Це відповідає вимогам Європейського акту про штучний інтелект [6].

Відповідність галузевим стандартам передбачає дотримання специфічних вимог для різних типів контенту:

– медична та фінансова інформація супроводжується застереженнями;
– рекламний контент позначається відповідно до законодавства;
– новинні матеріали проходять додаткову перевірку.

Наша система постійно оновлюється з урахуванням змін у законодавстві та етичних стандартах, що забезпечує безпечне та відповідальне використання технологій ШІ для генерації контенту [7].

1.7 Висновки до розділу 1

У першому розділі дослідження було проведено комплексний аналіз теоретичних основ генерації вебконтенту за допомогою технологій штучного

інтелекту. Великі мовні моделі, такі як GPT, Claude, BERT та T5, демонструють високий потенціал для автоматизації створення контенту, але мають різні переваги та обмеження. Оптимальний підхід полягає в комбінованому використанні цих моделей для різних типів завдань.

Аналіз існуючих сервісів генерації контенту показав, що більшість рішень мають суттєві обмеження щодо гнучкості, вартості та можливості адаптації під специфічні потреби. Це обґрунтовує необхідність розробки спеціалізованого рішення, орієнтованого на україномовних користувачів.

Архітектура системи має ґрунтуватися на мікросервісному підході з чітким розподілом функціоналу, що забезпечить масштабованість та надійність рішення. Інтеграція кількох LLM дозволить оптимізувати витрати та якість контенту. Вимоги до якості згенерованого контенту включають комплекс критеріїв від технічної коректності до відповідності SEO-стандартам та етичним нормам. Необхідно реалізувати багаторівневу систему валідації для забезпечення високих стандартів якості.

Отримані результати створюють теоретичну основу для практичної реалізації системи генерації вебконтенту, що буде розглянуто в наступних розділах роботи.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ГЕНЕРАЦІЇ КОНТЕНТУ

2.1 Формування вимог до функціоналу та архітектури застосунку

На основі проведеного аналізу в розділі 1 було сформовано комплекс вимог до функціоналу та архітектури вебзастосунку для генерації контенту. Вимоги розподілені на дві основні категорії: функціональні та нефункціональні.

Функціональні вимоги включають:

– модуль генерації текстів різних типів (статті, пости, описи);
– система шаблонів для стандартних форматів контенту;
– інтеграція з зовнішніми LLM API (OpenAI GPT, Claude, інші);
– інструменти редагування та коректури згенерованого контенту;
– можливість збереження та керування історією робіт;
– експорт контенту у різних форматах;
– інтеграція з CMS та соціальними мережами;
– система перевірки якості та унікальності контенту.

Нефункціональні вимоги охоплюють:

– продуктивність: час відгуку системи не більше 3 секунд;
– масштабованість: підтримка до 1000 одночасних користувачів;
– безпека: шифрування даних, аутентифікація, авторизація;
– зручність використання: інтуїтивний інтерфейс, підтримка української мови.

Архітектурні вимоги передбачають реалізацію мікросервісної архітектури з розподілом на наступні компоненти:

– API Gateway для керування запитами;
– сервіс генерації контенту з підтримкою різних LLM;
– сервіс кешування та оптимізації;
– база даних для зберігання історії та шаблонів;
– сервіс аналітики та моніторингу;

– клієнтський інтерфейс на сучасному фреймворку.

Також визначені вимоги до інтеграції з зовнішніми сервісами, системи логирования та обробки помилок, механізми резервного копіювання та відновлення даних [7].

Сформовані вимоги забезпечать створення ефективного, безпечного та масштабованого рішення для генерації вебконтенту, що відповідає потребам сучасних користувачів.

2.2 Вибір та обґрунтування технологічного стеку

Вибір технологічного стеку є критично важливим етапом проєктування програмного забезпечення, оскільки безпосередньо впливає на продуктивність, масштабованість та підтримуваність системи. У сучасних умовах існує значна кількість технологій та інструментів розробки, кожен з яких має свої унікальні переваги та обмеження. Для проєкту вебзастосунку генерації контенту за допомогою великих мовних моделей було сформовано оптимальний технологічний стек, що враховує специфічні вимоги до обробки природної мови, інтеграції з зовнішніми API та управління даними.

Мова програмування Python обрана як основа back-end частини системи через її провідні позиції в галузі штучного інтелекту та обробки природної мови. Багата екосистема бібліотек, включаючи TensorFlow, PyTorch, Transformers та інші, забезпечує ефективну роботу з LLM. Фреймворк FastAPI, реалізований на Python, надає високошвидкісний інструмент для створення API, що критично важливо для асинхронної обробки запитів до мовних моделей. Автоматична генерація документації за стандартом OpenAPI спрощує процес інтеграції з клієнтськими застосунками та зовнішніми сервісами.

WordPress REST API інтегрований в систему як стандартизований інтерфейс для публікації згенерованого контенту. Цей підхід дозволяє уніфікувати процес розміщення матеріалів на вебсайтах, побудованих на основі WordPress, що становить понад 40% всіх вебсайтів в інтернеті. Використання

REST API забезпечує гнучку взаємодію з системою управління контентом, дозволяючи автоматизувати процеси публікації, оновлення та управління матеріалами [7].

Система управління базами даних PostgreSQL обрана через її надійність, стабільність та розширені можливості роботи зі структурованими та напівструктурованими даними. Підтримка JSONB дозволяє ефективно зберігати гнучкі структури контенту, метадані та результати роботи ML-моделей. Поєднання реляційної структури з можливостями NoSQL забезпечує оптимальну основу для зберігання як структурованих даних користувачів, так і динамічного контенту, що генерується системою [7].

Віртуальні приватні сервери (VPS) на платформах DigitalOcean та Linode забезпечують необхідну гнучкість та масштабованість інфраструктури. Ці платформи пропонують збалансоване співвідношення ціни та якості, передбачувану продуктивність та зручні інструменти управління. Використання VPS дозволяє ефективно масштабувати ресурси відповідно до навантаження та забезпечує високу доступність сервісу.

Фреймворк для тестування pytest забезпечує надійну основу для перевірки якості коду та функціональності системи. Його простий синтаксис, можливість параметризації тестів та інтеграція з інструментами CI/CD роблять pytest оптимальним вибором для забезпечення стабільності та надійності програмного забезпечення [7].

Цей технологічний стек сформований на основі ретельного аналізу вимог проєкту та врахуванням сучасних тенденцій розробки програмного забезпечення, що забезпечує оптимальну основу для реалізації ефективної системи генерації вебконтенту [8].

2.3 Проєктування схеми бази даних PostgreSQL

У розробці вебзастосунку для генерації контенту використовується система керування базами даних PostgreSQL, яка забезпечує надійне зберігання

та ефективну обробку структурованих і напівструктурованих даних. База даних побудована відповідно до принципів реляційної моделі з використанням сучасних можливостей PostgreSQL, таких як JSONB для зберігання гнучких структур, що дозволяє оптимально керувати даними генерації контенту, шаблонами, історією запитів та результатами роботи ML-моделей [8].

Основна мета реалізації бази даних – зберігати дані про користувачів, їхні проєкти, шаблони контенту, історію генерації, результати роботи з різними LLM API, а також метрики якості контенту. Така модель дозволяє реалізувати ключові функції системи: створення та зберігання шаблонів, генерацію контенту через різні API, управління проєктами, аналітику якості та інтеграцію з WordPress [8].

Для побудови ефективної структури бази даних необхідно визначити ключові сутності та ролі користувачів.

Користувачі – основні учасники системи. Мають поля: ім’я, email, хеш пароля, avatar_url, created_at, updated_at, subscription_type.

Ролі:

– звичайний користувач – має доступ до створення проєктів, генерації контенту, перегляду історії;
– адміністратор – може керувати шаблонами, переглядати системну статистику, керувати користувачами.

Основні сутності бази даних:

– users – зберігає облікові запити користувачів;
– projects – проєкти користувачів для організації контенту;
– content_templates – шаблони для генерації різних типів контенту;
– generation_requests – історія запитів на генерацію контенту;
– generation_results – результати роботи LLM;
– api_providers – налаштування підключення до різних LLM API;
– wordpress_integrations – налаштування інтеграції з WordPress сайтами;
– content_metrics – метрики якості згенерованого контенту.

На рисунку 2.1 представлено ER-діаграму бази даних вебзастосунку, яка демонструє взаємозв’язки між основними сутностями та їх атрибутами.

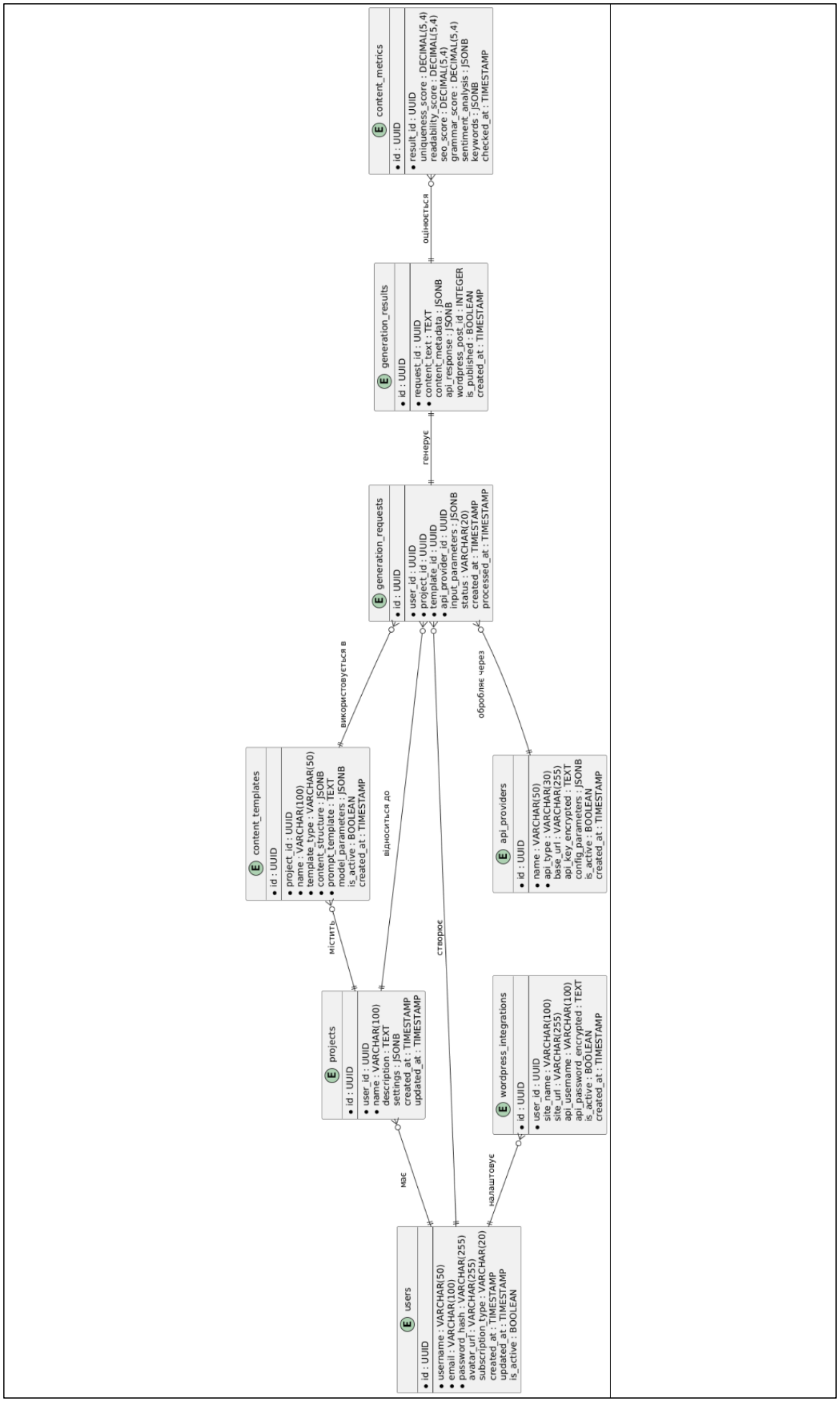


Рисунок 2.1 – ER-діаграм бази даних

Діаграма дає змогу візуально оцінити структуру даних і показує, як реалізовано залежності між таблицями, включаючи зовнішні ключі та обмеження цілісності [9].

Опишемо таблиці сутностей і зв’язків (ERD) на основі наданої діаграми бази даних.

Таблиця users – «Користувачі»:

– id – унікальний ідентифікатор користувача, автоматично генерується;
– username – унікальне ім’я користувача в системі;
– email – електронна пошта користувача (унікальна);
– password_hash – зашифрований пароль користувача;
– avatar_url – посилання на аватар користувача (необов’язково);
– subscription_type – тип підписки користувача (free, premium тощо);
– created_at – дата та час створення облікового запису;
– updated_at – дата та час останнього оновлення інформації користувача;
– is_active – статус активності облікового запису.

Таблиця projects – «Проєкти»:

– id – унікальний ідентифікатор проєкту;
– user_id – зовнішній ключ, що посилається на користувача, який створив проєкт;
– name – назва проєкту;
– description – опис проєкту (необов’язково);
– settings – JSON-об’єкт з налаштуваннями проєкту;
– created_at – дата та час створення проєкту;
– updated_at – дата та час останнього оновлення проєкту.

Таблиця content_templates – «Шаблони контенту»:

– id – унікальний ідентифікатор шаблону;
– project_id – зовнішній ключ, що посилається на проєкт, до якого належить шаблон;
– name – назва шаблону;

– template_type – тип шаблону (стаття, пост, опис продукту тощо);
– content_structure – JSON-структура вмісту шаблону;
– prompt_template – шаблон промпту для генерації контенту;
– model_parameters – параметри моделі для генерації;
– is_active – статус активності шаблону;
– created_at – дата та час створення шаблону.

Таблиця api_providers – «Постачальники API»:
– id – унікальний ідентифікатор постачальника;
– name – назва постачальника (OpenAI, Claude тощо);
– api_type – тип API;
– base_url – базова URL-адреса API;
– api_key_encrypted – зашифрований API-ключ;
– config_parameters – додаткові параметри конфігурації;
– is_active – статус активності постачальника;
– created_at – дата та час додавання постачальника.

Таблиця generation_requests – «Запити на генерацію»:
– id – унікальний ідентифікатор запиту;
– user_id – зовнішній ключ, що посилається на користувача, який ініціював запит;
– project_id – зовнішній ключ, що посилається на проєкт;
– template_id – зовнішній ключ, що посилається на використаний шаблон;
– api_provider_id – зовнішній ключ, що посилається на використаного постачальника API;
– input_parameters – вхідні параметри для генерації;
– status – статус запиту (pending, processing, completed, failed);
– created_at – дата та час створення запиту;
– processed_at – дата та час обробки запиту.

Таблиця generation_results – «Результати генерації»:
– id – унікальний ідентифікатор результату;
– request_id – зовнішній ключ, що посилається на відповідний запит;

– content_text – згенерований текстовий контент;
– content_metadata – метадані контенту;
– api_response – повна відповідь від API;
– wordpress_post_id – ID поста в WordPress (якщо опубліковано);
– is_published – позначка про публікацію контенту;
– created_at – дата та час створення результату.

Таблиця content_metrics – «Метрики контенту»:

– id – унікальний ідентифікатор метрики;
– result_id – зовнішній ключ, що посилається на оцінений результат;
– uniqueness_score – оцінка унікальності контенту;
– readability_score – оцінка читабельності;
– seo_score – SEO-оцінка контенту;
– grammar_score – оцінка граматичної якості;
– sentiment_analysis – аналіз тональності тексту;
– keywords – ключові слова, виявлені в контенті;
– checked_at – дата та час перевірки якості.

Таблиця wordpress_integrations – «Інтеграції з WordPress»:

– id – унікальний ідентифікатор інтеграції;
– user_id – зовнішній ключ, що посилається на користувача;
– site_name – назва WordPress-сайту;
– site_url – URL-адреса сайту;
– api_username – ім’я користувача для API;
– api_password_encrypted – зашифрований пароль для API;
– is_active – статус активності інтеграції;
– created_at – дата та час створення інтеграції.

У базі даних проєкту реалізовано зв’язки відповідно до логіки взаємодії між сутностями. Зокрема, між таблицями “users” і “projects”, “generation_requests”, “wordpress_integrations” встановлено зв’язок один до багатьох – один користувач може мати багато проєктів, запитів та інтеграцій, але кожен із них належить лише одному користувачу [9].

Запити на генерацію пов’язані з кількома сутностями через зовнішні ключі до користувача, проєкту, шаблону та постачальника API. Це дозволяє відстежувати повну історію генерації контенту для кожного користувача.

Результати генерації моделюються через зв’язок один до одного з запитами, що забезпечує цілісність даних. Кожен запит має рівно один результат, і кожен результат належить рівно одному запиту.

Якість контенту оцінюється через зв’язок один до багатьох між результатами генерації та метриками якості, що дозволяє зберігати історію оцінок для кожного згенерованого контенту [9].

Таким чином, у базі даних застосовані основні типи зв’язків – один до багатьох і один до одного, що забезпечує коректну структуру для збереження інформації про користувачів, проєкти, шаблони, запити генерації, результати роботи ML-моделей, метрики якості та інтеграції з WordPress.

2.4 Розробка модулів взаємодії з WordPress REST API для публікації контенту

Проєктування модулів взаємодії з WordPress REST API є критично важливим етапом, що забезпечує безперебійну публікацію згенерованого контенту на вебсайтах клієнтів. Архітектура модуля включає наступні основні компоненти:

- модель WordPressSite;
- модель WordPressPost;
- сервіс WordPressAPIClient;
- модель ContentPublisher;
- модель SyncStatus;
- Repository;
- модель обробки помилок;
- система безпеки;
- модуль тестування.

На рисунку 2.2 зображено діаграму моделей модуля взаємодії з WordPress REST API, яка візуалізує ключові компоненти системи та їх взаємозв’язки. Діаграма демонструє архітектуру, побудовану на принципах об’єктно-орієнтованого програмування та шаблону Repository [10].

Центральне місце в архітектурі займає модель WordPressSite, яка відповідає за зберігання конфігурації підключених WordPress-сайтів. Кожен сайт має унікальний ідентифікатор, URL-адресу, зашифровані облікові дані для автентифікації, а також налаштування публікації за замовчуванням. Модель включає методи для тестування з’єднання, отримання списку категорій та валідації URL-адрес.

Модель WordPressPost інкапсулює структуру публікації та забезпечує трансформацію даних з внутрішнього формату системи у формат, сумісний з WordPress REST API. Вона містить всі необхідні поля: заголовок, основний текст, короткий опис, статус публікації, категорії, теги та ідентифікатор головного зображення [10].

Сервіс WordPressAPIClient є ядром модуля та відповідає за безпосередню взаємодію з WordPress REST API. Він реалізує механізми автентифікації, обробки мережевих запитів, керування лімітами запитів та повторних спроб при виникненні помилок. Сервіс підтримує всі основні операції: створення та оновлення постів, завантаження медіа-файлів, отримання списку категорій.

ContentPublisher виступає координатором процесу публікації, інтегруючи функціональність інших компонентів. Він забезпечує правильну послідовність дій: перетворення контенту, завантаження медіа-файлів, створення чернетки та фінальну публікацію. Модель також підтримує функцію відкладеного публікування [10].

SyncStatus відстежує стан синхронізації для кожного підключеного сайту, фіксуючи час останньої успішної операції, кількість помилок та їх характер. Ця інформація використовується для моніторингу здоров’я інтеграції та своєчасного виявлення проблем.

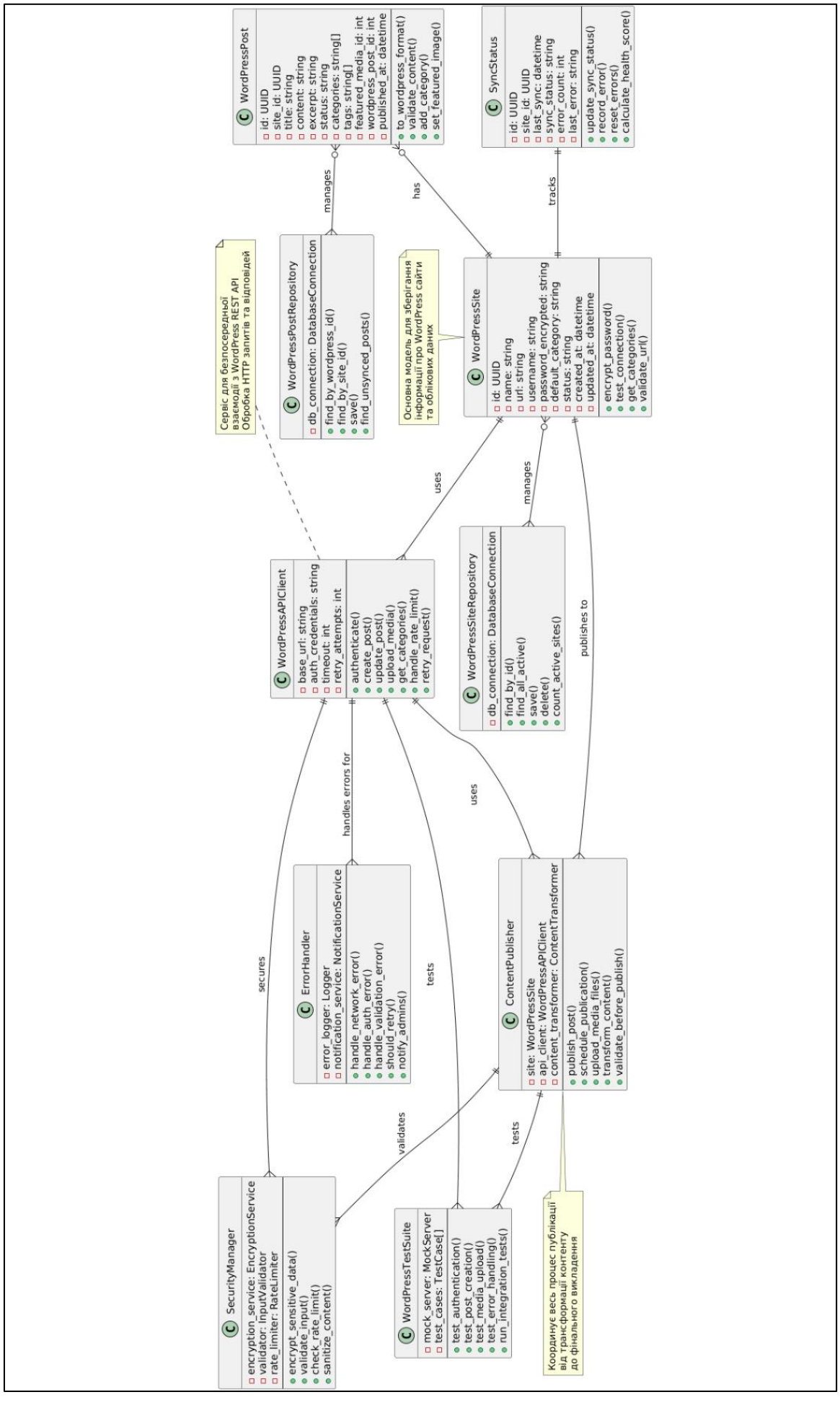


Рисунок 2.2 – ER-діаграма моделей модуля взаємодії з WordPress REST API

Архітектура включає два Repository класи – WordPressSiteRepository та WordPressPostRepository, які відповідають за зберігання та отримання даних з бази даних PostgreSQL. Вони реалізують шаблон Repository для абстракції доступу до даних.

ErrorHandler забезпечує надійну обробку помилок, класифікуючи їх за типами (мережеві, автентифікації, валідації) та визначаючи стратегію обробки для кожного випадку.

Система безпеки (SecurityManager) відповідає за шифрування чутливих даних, валідацію вхідних параметрів та захист від зловживань.

Модуль тестування WordPressTestSuite включає комплекс тестів для перевірки всіх аспектів роботи модуля – від юніт-тестів окремих моделей до інтеграційних тестів з реальним WordPress-сайтом [10]. Така архітектура забезпечує масштабованість, надійність та безпеку модуля взаємодії з WordPress, дозволяючи ефективно керувати процесом публікації контенту на множині вебсайтів [10].

Також на рисунку 2.3 представлено діаграму компонентів, яка описує архітектурну структуру системи генерації вебконтенту, реалізованої за допомогою засобів Python, FastAPI, PostgreSQL та інтеграції з WordPress REST API. Ця діаграма візуалізує архітектуру системи генерації вебконтенту, побудованої на сучасному технологічному стеку з використанням Python, FastAPI, PostgreSQL та інтеграції з WordPress REST API. Система реалізована за модульним принципом з чітким розподілом функціональних обов’язків між компонентами.

Клієнтський інтерфейс взаємодіє з системою через API Gateway, який виступає єдиною точкою входу та відповідає за маршрутизацію запитів, аутентифікацію користувачів, валідацію вхідних даних та контроль доступу. Цей підхід забезпечує безпеку та масштабованість системи.

Основний функціонал реалізований у трьох ключових сервісах. Content Generation Service відповідає за генерацію текстів через інтеграцію з зовнішніми LLM API, валідацію якості контенту та обробку помилок.

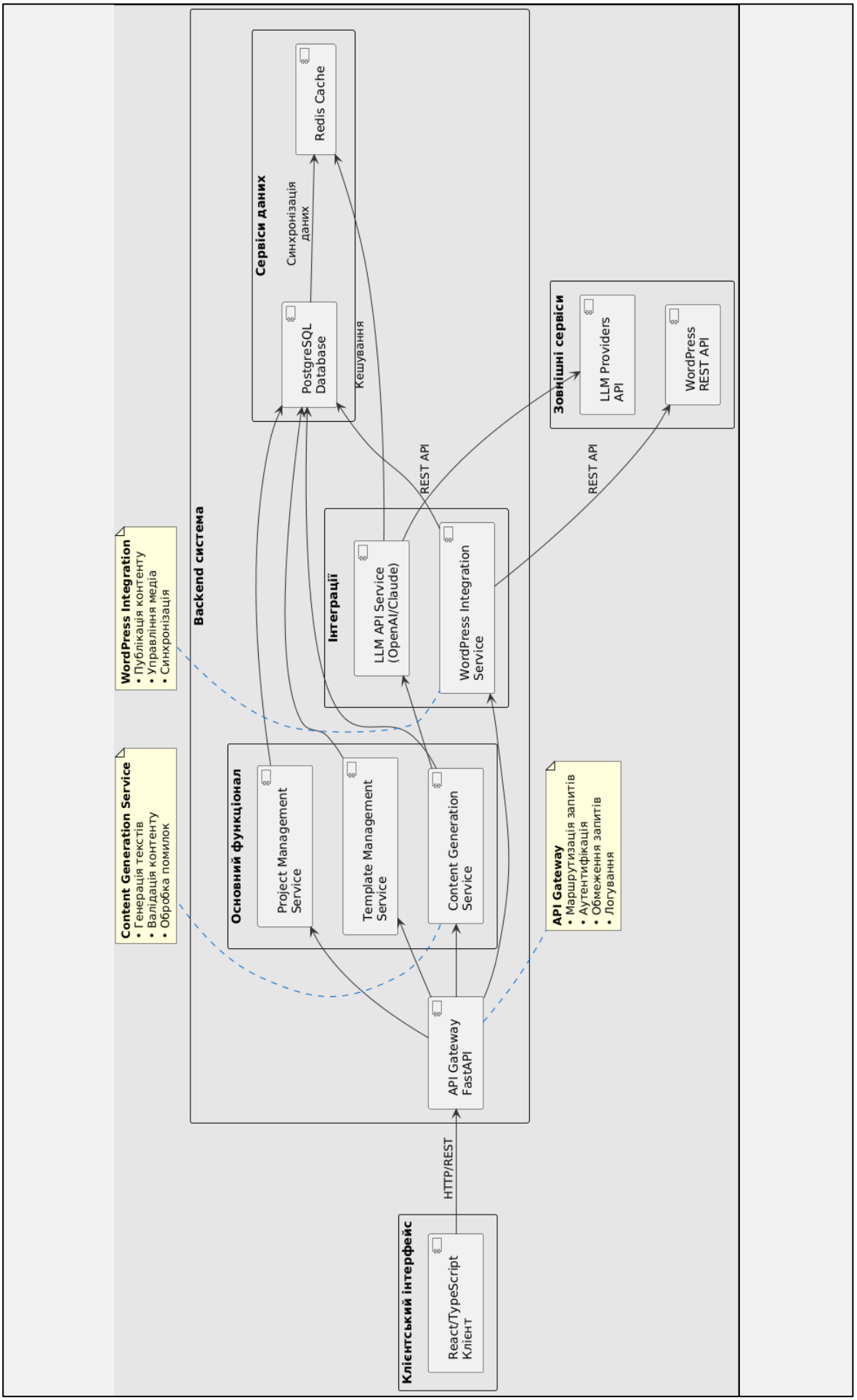


Рисунок 2.3 – Діаграма компонентів

Template Management Service забезпечує роботу з шаблонами контенту різних типів, включаючи створення, редагування та версійність. Project Management Service надає інструменти для організації проєктів користувачів, управління правами доступу та ведення статистики [11].

Інтеграційний шар включає два критично важливих компоненти. WordPress Integration Service забезпечує повний цикл публікації контенту на WordPress сайтах, включаючи автентифікацію, створення постів, завантаження медіа-файлів та управління категоріями. LLM API Service інкапсулює логіку роботи з різними провайдерами штучного інтелекту, забезпечуючи уніфікований інтерфейс, оптимізацію витрат та механізми відновлення при збоях.

Система управління даними базується на комбінації PostgreSQL та Redis. PostgreSQL використовується для зберігання структурованої інформації: профілів користувачів, проєктів, шаблонів контенту та історії операцій. Redis забезпечує високу продуктивність через кешування результатів, управління сесіями та обробку асинхронних завдань [11]. Така архітектура дозволяє системі ефективно обробляти великі обсяги запитів на генерацію контенту, забезпечує надійну інтеграцію з WordPress сайтами та надає гнучкі інструменти для роботи з різними типами контенту. Кожен компонент системи може розвиватися та масштабуватися незалежно, що забезпечує високу гнучкість та полегшує впровадження нових функціональних можливостей у майбутньому [12].

Також на рисунку 2.4 представлено діаграму класів бекенду, яка деталізує внутрішню структуру серверної частини системи генерації вебконтенту.

Центральною моделлю системи є клас User, який відповідає за управління користувачами та їх правами доступу. Кожен користувач може створювати множину проєктів через клас Project, що дозволяє організовувати контент за тематичними групами. Проєкти містять набір шаблонів контенту, представлених класом ContentTemplate, які визначають структуру та параметри генерації різних типів вебконтенту [12].

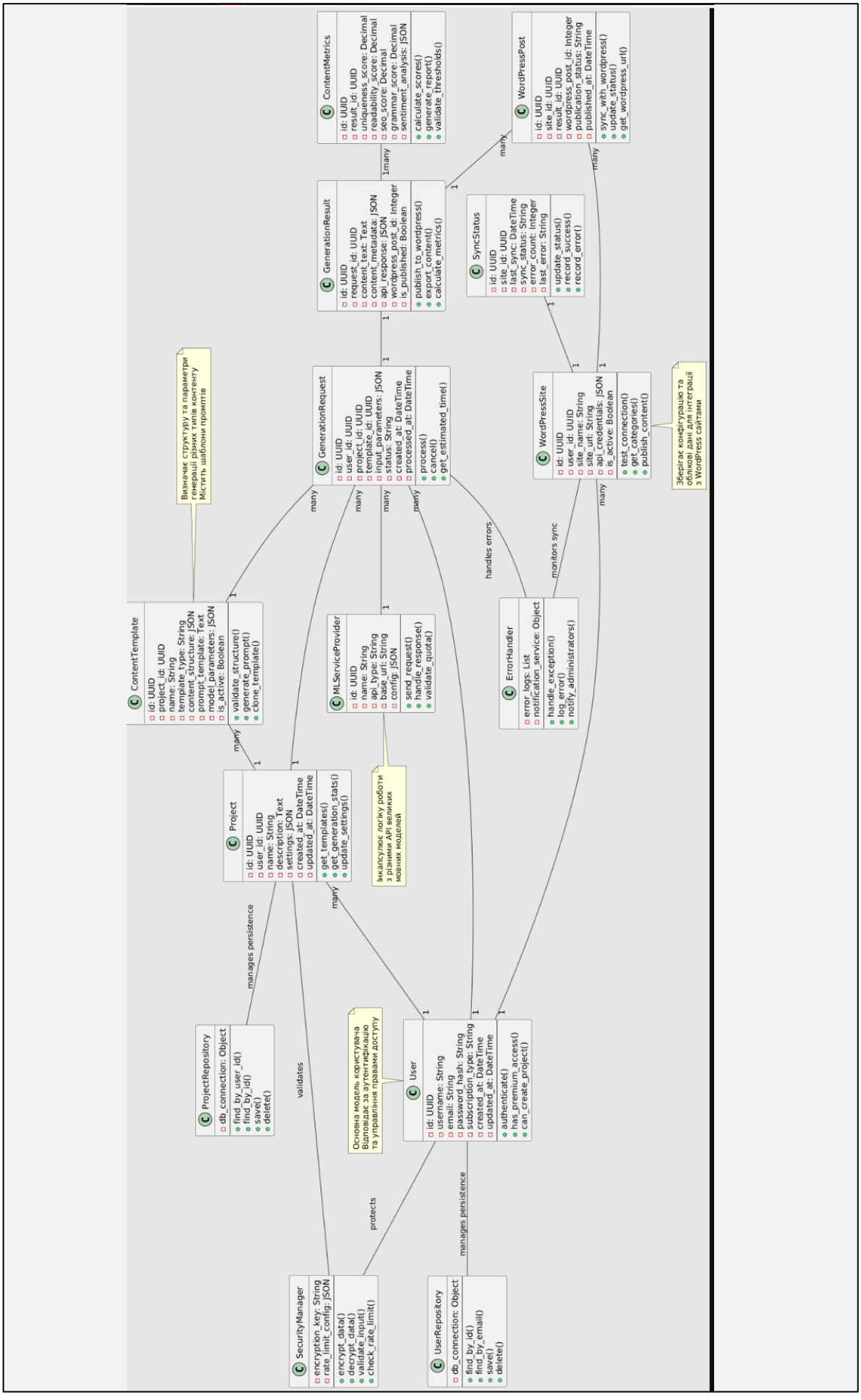


Рисунок 2.4 – Діаграма класів бекенду

Для управління процесом генерації використовується клас GenerationRequest, що фіксує параметри кожного запиту на створення контенту. Цей клас пов’язаний з класом GenerationResult, який зберігає результати роботи ML-моделей та метадані згенерованого контенту. Якість отриманих результатів оцінюється через клас ContentMetrics, що містить метрики унікальності, читабельності та SEO-оптимізації.

Інтеграція з WordPress реалізована через клас WordPressSite, який зберігає конфігурацію підключених сайтів та облікові дані для API. Клас WordPressPost відповідає за трансформацію локальних даних у формат, сумісний з WordPress REST API, та управляє процесом публікації. Статус синхронізації відстежується через клас SyncStatus, що фіксує час останньої успішної операції та інформацію про помилки [12].

Робота з зовнішніми ML-сервісами інкапсульована в класі MLServiceProvider, який забезпечує уніфікований інтерфейс для взаємодії з різними API великих мовних моделей. Клас відповідає за маршрутизацію запитів, обробку відповідей та керування лімітами використання. Для забезпечення безпеки використовується клас SecurityManager, що відповідає за шифрування чутливих даних, валідацію вхідних параметрів та захист від зловживань.

Обробка помилок реалізована через клас ErrorHandler, який класифікує виняткові ситуації за типами та забезпечує відповідну стратегію їх обробки. Для зберігання даних використовуються репозиторії, такі як UserRepository та ProjectRepository, що інкапсулюють логіку роботи з базою даних та забезпечують абстракцію доступу до даних [12]. Така структура класів забезпечує чітке розділення відповідальності, спрощує тестування та подальше розширення функціоналу системи. Кожен клас виконує конкретну бізнес-задачу, що дозволяє легко модифікувати окремі компоненти без впливу на загальну архітектуру застосунку.

2.5 Висновки до розділу 2

У другому розділі, присвяченому проєктуванню системи генерації вебконтенту, визначено комплекс функціональних та нефункціональних вимог до системи, що дозволило чітко сформулювати її основні можливості та критерії ефективності. Проведено детальний аналіз архітектурних рішень та обґрунтовано вибір технологічного стеку на основі Python, FastAPI, PostgreSQL та інтеграції з WordPress REST API.

Розроблено схему бази даних PostgreSQL, яка включає вісім основних таблиць з чіткими взаємозв’язками між сутностями. Створено діаграму компонентів системи, що візуалізує взаємодію між клієнтською частиною, серверними сервісами та зовнішніми API. Деталізовано архітектуру модуля взаємодії з WordPress REST API через діаграму класів, яка демонструє ключові моделі та їх взаємозв’язки у бекенд-частині системи.

Запропонована архітектура передбачає використання мікросервісного підходу з чітким розподілом відповідальності між компонентами, що забезпечує високу масштабованість та підтримуваність системи. Розроблено моделі безпеки, обробки помилок та тестування, які гарантують надійність роботи системи в реальних умовах експлуатації.

Загалом, проведене проєктування сформувало цілісну архітектурну основу для подальшої реалізації системи генерації вебконтенту, яка відповідає сучасним вимогам до продуктивності, масштабованості та зручності використання. Запропоновані рішення забезпечують ефективну інтеграцію з сучасними AI-технологіями та WordPress платформою, що робить систему конкурентноздатною на ринку інструментів для автоматизації створення контенту.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБСЕРВІСУ

3.1 Інтеграція з CMS WordPress та системне тестування

Даний підрозділ присвячено ключовому етапу реалізації вебзастосунку – забезпеченню його взаємодії з зовнішньою системою управління контентом (CMS) та перевірці коректності роботи всієї інтегрованої системи. Основною метою інтеграції є автоматизація процесу публікації вебконтенту, згенерованого за допомогою великої мовної моделі (LLM), безпосередньо на цільовий вебсайт, побудований на платформі WordPress. Системне тестування дозволяє підтвердити надійність, безпеку та функціональну відповідність розробленого рішення заданим вимогам.

Для реалізації інтеграції було обрано стандартизований та добре документований механізм – WordPress REST API. Цей програмний інтерфейс надає повний спектр можливостей для керування контентом (створення, читання, оновлення, видалення постів, сторінок, медіафайлів) через HTTP-запити.

Архітектурно взаємодія реалізована за наступною схемою:

– користувач через інтерфейс React-додатка ініціює запит на генерацію та публікацію контенту для певного сайту;
– Backend-сервіс (FastAPI) отримує запит, перевіряє права доступу користувача до цільового WordPress-сайту та ініціює процес генерації через ML-мікросервіс;
– ML-мікросервіс повертає сформований контент (HTML-розмітку, текст, заголовки, мета-дані) у структурованому вигляді (наприклад, JSON);
– Backend-сервіс отримує готовий контент, проводить його фінальне форматування та формує аутентифікований HTTP-запит (з використанням JWT або Application Passwords) до REST API цільового WordPress-сайту для створення нового поста або сторінки;

– WordPress обробляє запит, створює запис у базі даних та повертає відповідь (ID створеного поста, посилання) у бекенд-сервіс;
– Backend-сервіс логує результат операції та повертає статус користувачеві через фронтенд.

Для зберігання облікових даних (токени аутентифікації, URL сайтів), необхідних для інтеграції, використовується база даних PostgreSQL. Кожен запис пов'язаний із конкретним користувачем та його налаштованими сайтами, що забезпечує безпеку та ізоляцію даних.

На рисунку 3.1 зображено архітектуру модуля інтеграції з WordPress REST API, реалізованого на Python у складі FastAPI-додатка.

```python
import httpx
from pydantic import BaseModel

class WordPressPostData(BaseModel):
    title: str
    content: str
    status: str = "draft"  # або "publish"

class WordPressClient:
    def __init__(self, site_url: str, username: str, app_password: str):
        self.base_url = f"{site_url.rstrip('/')}/wp-json/wp/v2"
        self.auth = (username, app_password)

    async def create_post(self, post_data: WordPressPostData):
        async with httpx.AsyncClient(auth=self.auth) as client:
            payload = {
                "title": post_data.title,
                "content": post_data.content,
                "status": post_data.status
            }
            response = await client.post(f"{self.base_url}/posts", json=payload)
            response.raise_for_status()
            return response.json()
```

Рисунок 3.1 – Модуль інтеграції з WordPress REST API

Модуль інтеграції реалізовано у вигляді спеціалізованого класу WordPressClient, який інкапсулює всю логіку взаємодії з зовнішнім REST API системи WordPress. Його основним призначенням є забезпечення зручного, безпечного та ефективного програмного інтерфейсу для виконання операцій з контентом на віддаленому WordPress-сайті безпосередньо з коду Python-

застосунку. Клас WordPressClient побудовано з використанням сучасних бібліотек Python для асинхронного програмування та валідації даних. Його архітектура включає наступні ключові елементи.

Модель даних WordPressPostData. Для забезпечення типобезпеки та автоматичної валідації вхідної інформації використовується модель Pydantic WordPressPostData. Ця модель строго визначає структуру даних, необхідних для створення поста. Обов'язковими полями є рядок title (заголовок) та content (основний вміст у форматі HTML). Поле status визначено зі значенням за замовчуванням "draft", що дозволяє автоматично зберігати нові публікації як чернетки, надаючи користувачеві можливість попереднього перегляду перед публікацією. Альтернативним значенням цього поля є "publish" для миттєвої публікації.

Метод ініціалізації __init__. Конструктор класу приймає обов'язкові параметри для підключення до конкретного WordPress-сайту: його базову URL-адресу (site_url), ім'я користувача (username) та спеціально згенерований пароль додатка (app_password). У тілі конструктора відбувається формування базового URL для доступу до API шляхом конкатенації адреси сайту зі стандартним шляхом до REST API v2 (/wp-json/wp/v2). Операція rstrip('/') усуває можливий кінцевий слеш, запобігаючи подвійним слешам у фінальному URL. Дані для базової HTTP-автентифікації (логін та пароль) зберігаються в атрибуті self.auth у вигляді кортежу, готового до використання бібліотекою httpx.

Асинхронний метод create_post. Це ядро класу, що безпосередньо виконує HTTP-запит до WordPress API. Метод приймає один аргумент – екземпляр моделі WordPressPostData. Для виконання мережевого запиту використовується асинхронний контекстний менеджер httpx.AsyncClient. Клієнту передаються дані автентифікації через параметр auth, що гарантує включення відповідного заголовка Authorization у кожен запит.

Усередині контексту формується тіло запиту (payload) – словник Python, який автоматично серіалізується бібліотекою httpx у формат JSON. Цей

словник містить усі необхідні для WordPress поля: заголовок, контент та статус публікації.

Далі виконується асинхронний POST-запит за адресою, що складається з базового URL та шляху /posts. У разі успішного виконання запиту (статус-коди 2xx) метод response.raise_for_status() не викличе винятку. Відповідь від сервера WordPress, яка містить JSON з деталями створеного поста (його ID, посилання, дати тощо), перетворюється на словник Python за допомогою методу .json() і повертається як результат роботи методу.

Важливою особливістю реалізації є використання асинхронної моделі. Це дозволяє головному циклу обробки подій FastAPI-сервера не блокуватися на час виконання мережевого запиту до зовнішнього сервісу. Поки один потік очікує відповіді від WordPress, сервер може обробляти запити від інших клієнтів, що суттєво підвищує загальну пропускну здатність застосунку.

3.2 Реалізація серверної частини

Серверна частина вебзастосунку реалізована на базі сучасного фреймворку FastAPI, що забезпечує високу продуктивність, автоматичну генерацію документації API та підтримку асинхронних операцій. Архітектура backend побудована за принципами модульності та масштабованості, що дозволяє ефективно інтегрувати різні підсистеми: обробку запитів клієнтів, взаємодію з великими мовними моделями, управління даними та інтеграцію з зовнішніми сервісами.

На рисунку 3.2 зображено сервіс взаємодії з LLM (OpenAI API) для генерації контенту, що є основним компонентом бекенд-частини застосунку.

Сервіс взаємодії з великою мовною моделлю реалізований у вигляді спеціалізованого класу LLMService, який інкапсулює всю логіку комунікації з AI-провайдерами та трансформації запитів користувача у готовий вебконтент. Архітектура сервісу базується на принципах гнучкості, безпеки та ефективного управління ресурсами.

```python
import openai
from typing import List, Dict, Optional
from pydantic import BaseModel
import asyncio
from datetime import datetime

class GenerationRequest(BaseModel):
    prompt: str
    tone: str = "professional"
    length: str = "medium"
    keywords: List[str] = []
    language: str = "ukrainian"

class GeneratedContent(BaseModel):
    title: str
    content: str
    excerpt: Optional[str] = None
    meta_description: Optional[str] = None
    tags: List[str] = []
    generated_at: datetime

class LLMService:
    def __init__(self, api_key: str, model: str = "gpt-4"):
        openai.api_key = api_key
        self.model = model
        self.system_prompt = """Ти професійний копірайтер та веб-контент менеджер.
        Твоє завдання - створювати якісний, SEO-оптимізований контент для веб-сайтів.
        Відповідай ТІЛЬКИ в форматі JSON з ключами: title, content, excerpt, meta_descripti
s."""

    async def generate_content(self, request: GenerationRequest) -> GeneratedContent:
        """Генерація контенту за запитом користувача"""

        user_prompt = f"""
        Запит користувача: {request.prompt}
        Тон: {request.tone}
        Довжина: {request.length}
        Ключові слова: {', '.join(request.keywords)}
        Мова: {request.language}
```

Рисунок 3.2 – Сервіс взаємодії з LLM (OpenAI API)

Ключовим елементом сервісу є модель даних GenerationRequest, створена за допомогою бібліотеки Pydantic. Ця модель виконує роль контракту для вхідних даних, забезпечуючи строгу типізацію та автоматичну валідацію. Вона включає обов’язкове текстове поле prompt, яке містить основну тему майбутнього контенту, а також низку опціональних параметрів для тонкого налаштування результату. Параметр tone дозволяє контролювати стилістику вихідного тексту, пропонуючи варіанти від професійного до неформального спілкування. Параметр length дає змогу задати приблизний обсяг контенту, що критично важливо для адаптації матеріалу під різні формати вебсторінок.

Список keywords забезпечує інтеграцію SEO-оптимізації безпосередньо в процес генерації, а параметр language підтримує багатомовність з акцентом на українську мову як основну.

Інша модель даних, GeneratedContent, визначає структуру відповіді, яку повертає AI-сервіс. Вона включає не лише основний контент із HTML-розміткою, але й додаткові метадані, необхідні для ефективної публікації та просування в інтернеті. До таких даних належать заголовок, короткий уривок для анонсування, SEO-оптимізований мета-опис, список тегів для категорізації та точна мітка часу генерації. Така комплексна структура дозволяє клієнтському застосунку отримувати повноцінний, готовий до використання контент без необхідності додаткової обробки.

Під час ініціалізації класу LLMService відбувається налаштування параметрів підключення до AI-провайдера. Конструктор приймає API-ключ, який використовується для автентифікації запитів, та ідентифікатор конкретної мовної моделі. Важливим елементом ініціалізації є задання системного промпту – спеціально підготовленого інструктажу, який формує контекст для AI. У даній реалізації системний промпт чітко визначає роль моделі як професійного копірайтера та вебконтент менеджера, а також вимагає строгого дотримання JSON-формату у відповіді, що спрощує подальшу обробку результату.

Алгоритм роботи основного методу generate_content починається з трансформації вхідного запиту користувача у детальний промпт для AI. Цей процес включає об’єднання всіх параметрів GenerationRequest – основного запиту, тону, довжини, ключових слів та мови – у структуроване текстове повідомлення. Додатково включається інструкція щодо використання HTML-розмітки для оформлення контенту, що забезпечує готовність матеріалу до безпосереднього вставлення в системи управління контентом.

Наступним кроком виконується асинхронний запит до OpenAI API за допомогою спеціалізованого методу openai.ChatCompletion.acreate(). Використання асинхронної парадигми є критично важливим для продуктивності всього застосунку, оскільки дозволяє серверу обробляти інші

запити в період очікування відповіді від AI-сервісу, який може становити від кількох секунд до хвилин. У запиті явно задається параметр response_format зі значенням json_object, що змушує модель повертати дані у строго визначеному форматі, подальша обробка якого значно спрощується.

Після отримання відповіді від AI відбувається важливий етап парсингу JSON-даних. Відповідь, яка надходить у вигляді текстового рядка, перетворюється у словник Python за допомогою бібліотеки json. Ця операція супроводжується перевіркою наявності очікуваних ключів у відповіді та обробкою потенційних помилок десеріалізації. У разі успішного парсингу дані передаються до конструктора моделі GeneratedContent, яка забезпечує їхню валідність та узгодженість з очікуваною структурою. На рисунку 3.3 зображено архітектурну схему сервісу парсингу та обробки тексту за допомогою AI, який реалізує розширену функціональність аналізу вебконтенту та його подальшого використання для генерації нових матеріалів. Цей компонент відіграє ключову роль у забезпеченні можливості роботи з існуючими вебресурсами, надаючи інструменти для їхнього розуміння, структуризації та творчої переробки.

Сервіс парсингу реалізований у вигляді класу AIParsingService, що поєднує традиційні методи вебскрапінгу з потужними аналітичними можливостями великих мовних моделей. Його основне призначення полягає у автоматизації процесу вилучення корисного контенту з довільних вебсторінок, його семантичного аналізу та перетворення на структуровані дані, готові до подальшої обробки або інтеграції в систему генерації нового контенту.

Фундаментом сервісу є модель даних ParsingRequest, яка визначає параметри запиту на аналіз вебсторінки. Центральним елементом моделі є поле url, що містить адресу ресурсу для обробки. Список extract_types дозволяє тонко налаштувати процес парсингу, вказуючи, які саме типи інформації потрібно витягти – основний текст, метадані, ключові слова чи інші елементи. Параметр language підтримує мультимовність, пропонуючи як автоматичне визначення мови контенту, так і явне завдання мови для аналізу, що особливо важливо для коректної обробки україномовних ресурсів.

```python
import httpx
import json
from typing import List, Dict, Any
from bs4 import BeautifulSoup
import re

class ParsingRequest(BaseModel):
    url: str
    extract_types: List[str] = ["main_content", "metadata", "keywords"]
    language: str = "auto"

class ParsedContent(BaseModel):
    url: str
    title: str
    main_text: str
    summary: str
    keywords: List[str]
    html_structure: Dict[str, Any]
    word_count: int

class AIParsingService:
    def __init__(self, openai_api_key: str):
        self.openai_client = openai.AsyncOpenAI(api_key=openai_api_key)
        self.model = "gpt-4-turbo-preview"

    async def parse_and_analyze_url(self, request: ParsingRequest) -> ParsedContent:
        """Парсинг URL та аналіз контенту з використанням AI"""

        # Крок 1: Завантаження та базовий парсинг HTML
        html_content = await self._fetch_url_content(request.url)
        soup = BeautifulSoup(html_content, 'html.parser')

        # Видалення непотрібних елементів
        for element in soup(["script", "style", "nav", "footer"]):
            element.decompose()

        # Вилучення основного тексту
        main_text = soup.get_text(separator='\n', strip=True)
        cleaned_text = re.sub(r'\n\s*\n', '\n\n', main_text)
```

Рисунок 3.3 – Сервіс парсингу та обробки тексту за допомогою AI

Модель ParsedContent слугує контейнером для результатів аналізу, акумулюючи як сирі дані, витягнуті зі сторінки, так і висновки, згенеровані AI. Вона включає оригінальний URL, заголовок сторінки, очищений основний текст, AI-згенерований стислий переказ, список виявлених ключових слів, структурну інформацію про HTML-розмітку та статистичні дані, такі як кількість слів. Така комплексна структура забезпечує повноту інформації для різних сценаріїв використання – від швидкого ознайомлення з контентом до глибокого аналізу його властивостей.

Ключовим методом сервісу є parse_and_analyze_url, який реалізує багатоетапний алгоритм обробки вебконтенту. На першому етапі відбувається

завантаження HTML-вмісту сторінки за допомогою асинхронного HTTP-клієнта з настроюваними таймаутами та користувацьким агентом, що імітує роботу звичайного браузера для обходу основних захистів від скрапінгу. Отриманий HTML проходить процедуру очищення від непотрібних елементів – видаляються скрипти, стилі, навігаційні панелі та футери, що дозволяє сконцентруватися на основному контентному наповненні сторінки.

Після базового очищення відбувається вилучення текстового вмісту з подальшим його нормалізацією. Використовуються методи бібліотеки BeautifulSoup для перетворення HTML у читабельний текст із збереженням структурних елементів, таких як абзаци та заголовки. Регулярні вирази застосовуються для видалення надлишкових пробілів та порожніх рядків, що забезпечує оптимальний формат подальшої обробки та зменшує обсяг даних для передачі в AI-сервіс.

Другий етап роботи сервісу полягає у глибинному аналізі отриманого тексту за допомогою великої мовної моделі. Для цього формується спеціалізований промпт, який інструктує AI виконати комплексну оцінку контенту за кількома напрямками. Моделі пропонується створити стислий переказ основних ідей, визначити ключові слова та фрази, класифікувати загальну тему матеріалу, оцінити його тональність та сформувати рекомендації щодо SEO-оптимізації. Критично важливим аспектом є вимога до формату відповіді – AI повинен повертати дані у строгому JSON-форматі з чітко визначеними полями, що значно спрощує автоматичну обробку результатів.

Третій етап включає інтеграцію результатів AI-аналізу з даними, отриманими при базовому парсингу. Сервіс об’єднує аналітичні висновки моделі з технічною інформацією про структуру HTML-документу, яка збирається окремим методом _extract_html_structure. Цей метод здійснює інвентаризацію структурних елементів сторінки: підраховує кількість заголовків різних рівнів, абзаців, посилань, зображень та списків. Така комбінація семантичного та структурного аналізу забезпечує всебічне розуміння як змістового наповнення, так і формальних характеристик вебсторінки.

Особливістю реалізації сервісу є обмеження обсягу тексту, що передається до AI-моделі, приблизно 8000 символів, що зумовлено технічними обмеженнями на довжину контексту та оптимізацією вартості запитів. Для вирішення цього обмеження розроблено механізм інтелектуального скорочення тексту зі збереженням ключових семантичних елементів. У фінальній моделі ParsedContent також застосовується обмеження на довжину основного тексту, що забезпечує ефективне зберігання даних без втрати їхньої цінності для подальшої роботи. На рисунку 3.4 зображено архітектурну схему FastAPI ендпоінта для генерації контенту, який реалізує інтерфейс взаємодії між клієнтською частиною застосунку та сервісами штучного інтелекту.

```python
router = APIRouter(prefix="/api/v1/content", tags=["content"])
security = HTTPBearer()

class ContentGenerationEndpoint:
    def __init__(self, llm_service: LLMService, parsing_service: AIParsingService):
        self.llm_service = llm_service
        self.parsing_service = parsing_service
        self.generation_sessions = {}  # Імітація бази даних сесій

    @router.post("/generate")
    async def generate_content(
        self,
        request: GenerationRequest,
        background_tasks: BackgroundTasks,
        credentials: HTTPAuthorizationCredentials = Depends(security)
    ):
        """Ендпоінт для генерації нового контенту"""

        # Перевірка автентифікації (спрощено)
        user_id = self._authenticate_user(credentials.credentials)

        # Створення сесії генерації
        session_id = str(uuid.uuid4())
        self.generation_sessions[session_id] = {
            "status": "processing",
            "user_id": user_id,
            "request": request.dict(),
            "created_at": datetime.now()
        }

        # Запуск генерації в фоні
        background_tasks.add_task(
            self._process_generation,
            session_id,
            request
        )

        return {
            "session_id": session_id,
            "status": "processing",
            "message": "Генерація контенту розпочата"
```

Рисунок 3.4 – FastAPI ендпоінт для генерації контенту

Ендпоінт реалізований у формі класу ContentGenerationEndpoint, який об’єднує в собі функціональність управління сесіями генерації, автентифікації користувачів та маршрутизації запитів до відповідних AI-сервісів. Архітектура цього компонента ґрунтується на принципах RESTful API, що забезпечує стандартизований, зрозумілий інтерфейс для клієнтських застосунків будь-якого типу – від вебінтерфейсів на React до мобільних додатків або зовнішніх систем інтеграції.

Основним функціональним елементом є маршрутизатор FastAPI (APIRouter), який групує логічно пов’язані ендпоінти під спільним префіксом /api/v1/content. Така організація забезпечує чітку структуру API, спрощує версіювання та дозволяє легке розширення функціональності шляхом додавання нових маршрутів у майбутньому. Використання механізму тегів (tags=["content"]) покращує автоматично генеровану документацію Swagger, групуючи пов’язані ендпоінти для зручності розробників, які інтегруються з API.

Центральним ендпоінтом системи є POST /api/v1/content/generate, який ініціює процес створення нового контенту на основі запиту користувача. Цей ендпоінт реалізує асинхронну модель обробки, критично важливу для операцій, що займають значний час – генерація якісного контенту за допомогою LLM може тривати від декількох секунд до хвилин. Замість того, щоб блокувати клієнта на весь цей період, ендпоінт негайно повертає ідентифікатор сесії та статус «обробка», а сама генерація виконується у фоновому режимі за допомогою механізму BackgroundTasks FastAPI.

Процес роботи ендпоінта починається з перевірки автентифікаційних даних користувача через токен доступу, переданий у заголовку Authorization. Механізм HTTPBearer забезпечує стандартну обробку токенів, що сумісна з сучасними практиками безпеки вебдодатків. Після успішної автентифікації створюється унікальний ідентифікатор сесії за допомогою модуля uuid, який слугує ключем для відстеження статусу операції протягом її життєвого циклу. Інформація про сесію зберігається у спеціальному словнику (в реальній системі

зазвичай використовується база даних або кеш Redis), що дозволяє клієнту згодом запитувати статус виконання завдання.

Фонова задача _process_generation виконує безпосередню генерацію контенту через інтеграцію з LLMService. У разі успішного виконання вона оновлює статус сесії на «завершено» та зберігає результат. У разі помилки – фіксує деталі збою для подальшого аналізу. Такий підхід забезпечує надійність системи навіть при тимчасових проблемах з AI-сервісами або мережею.

Ендпоінт POST /api/v1/content/parse надає функціональність аналізу існуючого вебконтенту за URL. Він приймає модель ParsingRequest, що містить адресу ресурсу та параметри парсингу, і делегує обробку до AIParsingService. Цей ендпоінт повертає структурований аналіз сторінки, включаючи основний текст, ключові слова, SEO-рекомендації та структурну інформацію. Особливістю реалізації є комплексна обробка помилок, що дозволяє коректно реагувати на різноманітні проблеми – від недоступності сайту до некоректного формату HTML.

Найбільш інноваційним ендпоінтом є POST /api/v1/content/generate-from-url, який реалізує гібридний підхід до створення контенту. Він поєднує аналіз існуючого матеріалу з генерацією нового на його основі. Алгоритм роботи включає три ключових етапи: спочатку виконується парсинг вказаної вебсторінки для отримання її змістового та структурного аналізу, потім на основі цього аналізу та оригінального запиту користувача формується покращений промпт, і нарешті цей промпт використовується для генерації нового, унікального контенту. Такий підхід дозволяє створювати матеріали, що розвивають або альтернативно трактують теми існуючих публікацій, забезпечуючи додаткову цінність та унікальність.

Ендпоінт GET /api/v1/content/status/{session_id} забезпечує механізм відстеження статусу асинхронних операцій генерації. Клієнт може періодично опитувати цей ендпоінт, передаючи ідентифікатор сесії, отриманий при ініціації процесу. Реалізація включає перевірку прав доступу, що гарантує, що користувач може отримувати інформацію лише про власні сесії. Залежно від

статусу операції ендпоінт повертає або готовий результат, або інформацію про помилку, або повідомлення про те, що генерація ще триває.

Важливим аспектом архітектури є інтеграція з механізмами безпеки FastAPI. Використання залежностей (Depends) для автентифікації забезпечує централізовану обробку перевірки прав доступу для всіх ендпоінтів. У спрощеній реалізації використовується заглушка автентифікації, але в реальній системі вона замінюється на повноцінну перевірку JWT-токенів з підтвердженням підпису, верифікацією терміну дії та перевіркою відкликання.

Обробка помилок реалізована з використанням спеціалізованого класу HTTPException FastAPI, що дозволяє повертати клієнту структуровані повідомлення про помилки з відповідними HTTP-статусами. Це забезпечує зрозумілу семантику помилок – наприклад, статус 404 для не знайдених сесій, 403 для відмов у доступі або 400 для некоректних запитів. Усі внутрішні винятки AI-сервісів трансформуються у зрозумілі повідомлення, що дозволяє клієнтам адекватно реагувати на різні сценарії помилок.

На рисунку 3.5 зображено архітектурну схему ендпоінта для перевірки статусу генерації, який реалізує механізм відстеження та надання інформації про стан асинхронних операцій створення контенту. Цей компонент відіграє критичну роль у забезпеченні прозорості та керованості тривалих процесів генерації, дозволяючи клієнтським застосункам ефективно взаємодіяти з системою у режимі реального часу без необхідності блокування інтерфейсу користувача.

Ендпоінт реалізований як метод класу ContentGenerationEndpoint, що обслуговує запити за маршрутом GET /api/v1/content/status/{session_id}. Його основним призначенням є надання актуальної інформації про прогрес виконання операції генерації контенту, ініційованої раніше через відповідний POST-ендпоінт. Такий підхід реалізує патерн «запит-статус», особливо важливий для операцій, тривалість яких може суттєво варіюватись залежно від складності завдання, навантаження на AI-сервіси та інших факторів.

Архітектура ендпоінта базується на принципі ідентифікації сесій через унікальні ключі, що дозволяє точно відстежувати стан кожної окремої операції

генерації. Параметр маршруту session_id слугує основним ідентифікатором, який клієнт отримує при ініціації процесу генерації та використовує для всіх подальших запитів статусу. Використання UUID (універсальних унікальних ідентифікаторів) забезпечує практично нульову ймовірність колізії ідентифікаторів навіть при високих навантаженнях та паралельній обробці тисячі запитів.

```python
@router.get("/status/{session_id}")
async def get_generation_status(
    self,
    session_id: str,
    credentials: HTTPAuthorizationCredentials = Depends(security)
):
    """Перевірка статусу генерації контенту за session_id"""

    if session_id not in self.generation_sessions:
        raise HTTPException(status_code=404, detail="Сесія не знайдена")

    session = self.generation_sessions[session_id]

    # Перевірка прав доступу
    user_id = self._authenticate_user(credentials.credentials)
    if session["user_id"] != user_id:
        raise HTTPException(status_code=403, detail="Доступ заборонено")

    if session["status"] == "completed":
        return {
            "status": "completed",
            "result": session["result"],
            "created_at": session["created_at"],
            "completed_at": session.get("completed_at")
        }
    elif session["status"] == "failed":
        return {
            "status": "failed",
            "error": session.get("error"),
            "created_at": session["created_at"],
            "failed_at": session.get("failed_at")
        }
    else:
        return {
            "status": "processing",
            "message": "Генерація ще триває",
            "created_at": session["created_at"]
        }
```

Рисунок 3.5 – Ендпоінт для перевірки статусу генерації

Першим етапом роботи ендпоінта є перевірка існування сесії за вказаним ідентифікатором. Реалізація здійснює пошук у внутрішньому словнику generation_sessions, який у спрощеній версії виконує роль сховища стану сесій. У реальній промисловій системі це сховище зазвичай замінюється на зовнішню

базу даних або систему кешування, таку як Redis, що забезпечує персистентність даних між перезавантаженнями сервера та дозволяє масштабування на кілька екземплярів бекенду. У разі відсутності сесії з вказаним ідентифікатором ендпоінт повертає HTTP-статус 404 (Not Found) зі зрозумілим повідомленням про помилку, що дозволяє клієнту коректно обробити таку ситуацію – наприклад, запропонувати користувачеві повторно ініціювати генерацію.

Другим критично важливим етапом є перевірка прав доступу користувача до інформації про конкретну сесію. Ендпоінт витягує ідентифікатор користувача з токена автентифікації, переданого в заголовку Authorization, та порівнює його з ідентифікатором користувача, пов’язаним із сесією. Цей механізм гарантує, що користувач може отримувати інформацію лише про власні сесії генерації, що є фундаментальною вимогою безпеки багатокористувацьких систем. У разі невідповідності ідентифікаторів ендпоінт повертає статус 403 (Forbidden), сигналізуючи про відмову в доступі до запитаного ресурсу. Така реалізація запобігає потенційним витокам конфіденційної інформації та забезпечує ізоляцію даних між користувачами.

Після успішного проходження перевірок автентифікації та авторизації ендпоінт аналізує поточний статус сесії та формує відповідну відповідь. Логіка формування відповіді реалізована у вигляді умовних гілок, що обробляють три основні стани сесії: «завершено», «помилка» та «в обробці».

3.3 Реалізація клієнтської частини

Клієнтська частина застосунку реалізована у вигляді WordPress плагіна, що забезпечує безперервну інтеграцію з існуючим CMS-середовищем. Плагін розроблено з використанням сучасного стеку React, TypeScript та WordPress REST API, що дозволяє створювати інтерактивний інтерфейс безпосередньо в адміністративній панелі WordPress.

Архітектура плагіна організована за принципами сучасного frontend-розроблення з інтеграцією в екосистему WordPress:

- адміністративний інтерфейс (Admin UI): React-компоненти, вбудовані в WordPress адмін-панель через спеціальні сторінки меню;
- Gutenberg блоки: кастомні блоки для редактора Gutenberg, що дозволяють генерувати контент безпосередньо при редагуванні сторінок;
- REST API інтеграція: взаємодія з бекенд-сервісом генерації контенту через WordPress HTTP API;
- менеджер налаштувань: конфігураційна панель для зберігання API-ключів, налаштувань сайтів та параметрів генерації.

На рисунку 3.6 зображено структурну основного файлу плагіна ai-content-generator.php, який визначає базову архітектуру та функціональність WordPress-розширення для генерації вебконтенту за допомогою AI. Цей файл є центральним компонентом всього плагіна, що ініціалізує всі системи, реєструє хуки та забезпечує взаємодію між різними частинами розширення.

Головний файл плагіна реалізований як об’єктно-орієнтований клас AI_Content_Generator, який інкапсулює всю логіку ініціалізації та управління плагіном. Архітектура побудована на принципі одинака (Singleton), що гарантує існування лише одного екземпляра класу протягом життєвого циклу WordPress, запобігаючи конфліктам та непередбачуваній поведінці при множинних викликах.

Важливим елементом на початку файлу є визначення констант, які забезпечують зручний доступ до шляхів та URL-адрес плагіна. Константи AICG_PLUGIN_DIR та AICG_PLUGIN_URL використовуються по всьому коду плагіна для формування абсолютних шляхів до файлів та ресурсів, що усуває необхідність повторного виклику функцій plugin_dir_path() та plugin_dir_url(). Константа AICG_VERSION містить версію плагіна, яка використовується при реєстрації скриптів та стилів для забезпечення коректного кешування браузерами.

```php
// Запобігання прямому доступу
defined('ABSPATH') || exit;

// Константи плагіна
define('AICG_VERSION', '1.0.0');
define('AICG_PLUGIN_DIR', plugin_dir_path(__FILE__));
define('AICG_PLUGIN_URL', plugin_dir_url(__FILE__));

// Ініціалізація плагіна
class AI_Content_Generator {

    private static $instance = null;

    public static function get_instance() {
        if (null === self::$instance) {
            self::$instance = new self();
        }
        return self::$instance;
    }

    private function __construct() {
        $this->init_hooks();
    }

    private function init_hooks() {
        // Реєстрація меню адмін-панелі
        add_action('admin_menu', [$this, 'add_admin_menu']);

        // Завантаження скриптів та стилів
        add_action('admin_enqueue_scripts', [$this, 'enqueue_admin_assets']);

        // Реєстрація REST API маршрутів
        add_action('rest_api_init', [$this, 'register_rest_routes']);

        // Реєстрація Gutenberg блоків
        add_action('init', [$this, 'register_gutenberg_blocks']);

        // Активація/деактивація плагіна
        register_activation_hook(__FILE__, [$this, 'activate']);
        register_deactivation_hook(__FILE__, [$this, 'deactivate']);
    }
```

Рисунок 3.6 – Основний файл плагіна

Метод init_hooks() відіграє ключову роль у налаштуванні взаємодії плагіна з ядром WordPress. Він реєструє низку дій (actions) та фільтрів (filters), які визначають поведінку плагіна в різних контекстах. Дія admin_menu дозволяє плагіну додавати власні пункти меню в адміністративній панелі WordPress, створюючи зручний інтерфейс для користувачів. Дія admin_enqueue_scripts відповідає за завантаження JavaScript та CSS файлів лише на необхідних сторінках, що оптимізує продуктивність та запобігає конфліктам з іншими плагінами.

Реєстрація REST API маршрутів через дію rest_api_init є критично важливою для забезпечення взаємодії між фронтенд-компонентами на React та

бекенд-сервісами генерації контенту. Ці маршрути надають програмний інтерфейс для відправки запитів на генерацію, перевірки статусу операцій та публікації готового контенту. Дія init використовується для реєстрації кастомних блоків Gutenberg, що розширює можливості стандартного редактора WordPress.

Основним елементом користувацького інтерфейсу є метод add_admin_menu(), який створює головний пункт меню “AI Generator” в адміністративній панелі WordPress. Розташування пункту на 30 позиції гарантує його помітність без перекриття стандартних елементів меню. Для обмеження доступу використовується здатність edit_posts, що дозволяє користуватися плагіном усім користувачам, які мають право редагувати публікації. Додатково створюється підменю для налаштувань, доступне лише користувачам з правами адміністратора (manage_options).

Локалізація скриптів через функцію wp_localize_script() є важливим механізмом передачі даних з PHP в JavaScript. Об’єкт aicgSettings містить критично важливу інформацію: URL для REST API, nonce для захисту від CSRF-атак, URL поточного сайту, список типів публікацій та права доступу поточного користувача. Ця інформація використовується фронтенд-компонентами для адаптивної поведінки та забезпечення безпеки.

Метод enqueue_admin_assets() реалізує оптимізоване завантаження ресурсів для адміністративного інтерфейсу. Перевірка параметра $hook дозволяє завантажувати JavaScript та CSS файли лише на сторінках, пов’язаних з плагіном, що значно зменшує навантаження на інших сторінках адмін-панелі. Реєстрація скриптів здійснюється з використанням залежностей від вбудованих WordPress скриптів (wp-element, wp-api-fetch, wp-components), що забезпечує сумісність та зменшує розмір фінальних бандлів.

Локалізація скриптів через функцію wp_localize_script() є важливим механізмом передачі даних з PHP в JavaScript. Об’єкт aicgSettings містить критично важливу інформацію: URL для REST API, nonce для захисту від CSRF-атак, URL поточного сайту, список типів публікацій та права доступу

поточного користувача. Ця інформація використовується фронтенд-компонентами для адаптивної поведінки та забезпечення безпеки.

Реєстрація REST API маршрутів у методі register_rest_routes() створює три ключові точки доступу: для ініціації генерації контенту, перевірки статусу операції та публікації готового матеріалу. Кожен маршрут включає callback-функцію для обробки запиту та permission callback для перевірки прав доступу. Використання регулярних виразів у визначенні маршрутів (наприклад, для session_id) забезпечує гнучкість та безпеку при обробці параметрів.

Метод handle_generate_request() демонструє архітектуру проксі-сервера, коли WordPress плагін виступає проміжним ланом між користувацьким інтерфейсом та зовнішнім бекенд-сервісом генерації контенту. Плагін отримує запит від фронтенду, перевіряє налаштування (API ключ та URL бекенду), після чого перенаправляє запит на зовнішній сервер. Така архітектура дозволяє зберігати конфіденційні дані (API ключі) на сервері та унеможливлює їх витік до клієнтського коду.

Обробка відповідей від зовнішнього бекенду включає перевірку HTTP статус-кодів, парсинг JSON даних та трансформацію помилок у стандартний формат WordPress REST API. У разі мережевих проблем або некоректної відповіді від бекенду, плагін генерує зрозумілі повідомлення про помилки, що дозволяє користувачеві адекватно реагувати на проблеми.

На рисунку 3.7 зображено архітектурну діаграму React-компонента для адмін-панелі ContentGenerator.jsx, який реалізує інтерактивний інтерфейс для генерації вебконтенту за допомогою AI в межах WordPress адміністративної панелі. Цей компонент є центральним елементом користувацького досвіду, що поєднує функціональність введення параметрів, управління станом генерації та взаємодію з бекенд-сервісами.

Компонент ContentGenerator реалізований як функціональний React-компонент з використанням сучасних хуків, що забезпечує зручну роботу зі станом та життєвим циклом. Архітектура компонента побудована на принципі односпрямованого потоку даних, де зміни стану автоматично приводять до

оновлення інтерфейсу, забезпечуючи передбачувану поведінку та легку відладку.

```
import React, { useState, useEffect } from 'react';
import {
    Card,
    CardBody,
    CardHeader,
    TextControl,
    SelectControl,
    Button,
    Spinner,
    Notice,
    Panel,
    PanelBody,
    PanelRow
} from '@wordpress/components';
import apiFetch from '@wordpress/api-fetch';

const ContentGenerator = () => {
    const [prompt, setPrompt] = useState('');
    const [tone, setTone] = useState('professional');
    const [length, setLength] = useState('medium');
    const [keywords, setKeywords] = useState([]);
    const [currentKeyword, setCurrentKeyword] = useState('');
    const [isGenerating, setIsGenerating] = useState(false);
    const [sessionId, setSessionId] = useState(null);
    const [generatedContent, setGeneratedContent] = useState(null);
    const [error, setError] = useState(null);

    const handleGenerate = async () => {
        if (!prompt.trim()) {
            setError('Будь ласка, введіть запит для генерації');
            return;
        }

        setIsGenerating(true);
        setError(null);

        try {
            const response = await apiFetch({
                path: 'ai-content-generator/v1/generate',
                method: 'POST',
                data: {
```

Рисунок 3.7 – React-компонент для адмін-панелі

Основою компонента є система стану, реалізована за допомогою хука useState. Кожна частина інтерфейсу – від полів введення до результатів генерації – управляється окремими станами. Поле prompt зберігає запит користувача для генерації контенту, поля tone та length визначають стилістичні параметри вихідного тексту, масив keywords акумулює ключові слова для SEO-оптимізації, а стан currentKeyword слугує тимчасовим буфером для додавання

нових ключових слів. Така деталізація стану дозволяє точно контролювати кожен аспект інтерфейсу та забезпечує можливість часткового оновлення без необхідності перезавантаження всього компонента.

Критично важливими для роботи з асинхронними операціями є стани isGenerating, sessionId, generatedContent та error. Стан isGenerating виконує роль індикатора прогресу, блокує інтерактивні елементи під час виконання запиту та забезпечує відображення відповідного візуального стану. Ідентифікатор сесії sessionId слугує ключем для відстеження статусу тривалої операції генерації, дозволяючи компоненту періодично опитувати сервер на предмет завершення процесу. Об’єкт generatedContent містить повну структуру згенерованого AI матеріалу, включаючи заголовок, основний текст, мета-опис та теги, готовий до подальшої обробки або публікації. Стан error інкапсулює інформацію про помилки, що виникають під час виконання операцій, надаючи користувачеві зрозумілі повідомлення про проблеми.

Ядром функціональності компонента є метод handleGenerate(), який ініціює процес генерації контенту. Перед відправкою запиту він виконує валідацію вхідних даних, перевіряючи наявність обов’язкового поля prompt. У разі порожнього запиту встановлюється відповідне повідомлення про помилку, що запобігає марним викликам API та надає користувачеві корисний зворотний зв’язок. Після успішної валідації метод встановлює прапорець генерації та скидає попередні помилки, після чого формує структурований запит до REST API WordPress.

Запит до сервера виконується через спеціалізовану функцію apiFetch з пакета @wordpress/api-fetch, яка забезпечує стандартизовану взаємодію з WordPress REST API. Ця функція автоматично додає nonce для захисту від CSRF-атак та обробляє автентифікацію на основі поточного сеансу WordPress. У тіло запиту включаються всі параметри генерації: основний промпт, тон, довжина, ключові слова та мова (завжди українська). У разі успішного виконання запиту компонент отримує ідентифікатор сесії та запускає процес моніторингу статусу через функцію pollGenerationStatus().

Функція pollGenerationStatus() реалізує патерн періодичного опитування для відстеження прогресу довготривалої операції. Вона створює інтервал, який кожні 2 секунди виконує запит до API для перевірки статусу сесії генерації. Такий підхід є оптимальним для операцій, тривалість яких може варіюватись від кількох секунд до хвилин, оскільки він не вимагає підтримки постійного з'єднання (як WebSockets) та добре масштабується на серверній стороні. У разі отримання статусу "completed" функція зупиняє інтервал, зберігає результат генерації та оновлює відповідні стани компонента. Статус "failed" призводить до відображення повідомлення про помилку з деталями збою.

Окремим важливим етапом роботи компонента є збереження результатів генерації в базі даних WordPress через функцію saveGenerationToDatabase(). Ця операція створює постійний запис про кожну генерацію контенту, що дозволяє вести історію роботи, аналізувати ефективність різних промптів та відновлювати результати після перезавантаження сторінки. Запис містить ідентифікатор сесії, згенерований контент та оригінальний промпт, що забезпечує повний контекст для подальшого використання.

На рисунку 3.8 зображено структурну діаграму основного файлу плагіна ai-content-generator.php, який визначає базову архітектуру та функціональність WordPress-розширення для генерації вебконтенту за допомогою AI. Цей файл є центральним компонентом всього плагіна, що ініціалізує всі системи, реєструє хуки та забезпечує взаємодію між різними частинами розширення.

Головний файл плагіна реалізований як об'єктно-орієнтований клас AI_Content_Generator, який інкапсулює всю логіку ініціалізації та управління плагіном. Архітектура побудована на принципі одинака (Singleton), що гарантує існування лише одного екземпляра класу протягом життєвого циклу WordPress, запобігаючи конфліктам та непередбачуваній поведінці при множинних викликах.

Важливим елементом на початку файлу є визначення констант, які забезпечують зручний доступ до шляхів та URL-адрес плагіна. Константи AICG_PLUGIN_DIR та AICG_PLUGIN_URL використовуються по всьому коду

плагіна для формування абсолютних шляхів до файлів та ресурсів, що усуває необхідність повторного виклику функцій plugin_dir_path() та plugin_dir_url(). Константа AICG_VERSION містить версію плагіна, яка використовується при реєстрації скриптів та стилів для забезпечення коректного кешування браузерами.

```
import { registerBlockType } from '@wordpress/blocks';
import { TextControl, Button, PanelBody, Placeholder } from '@wordpress/components';
import { useState } from '@wordpress/element';

registerBlockType('ai-content-generator/block', {
    title: 'AI Content Generator',
    icon: 'edit',
    category: 'widgets',

    attributes: {
        prompt: {
            type: 'string',
            default: ''
        },
        generatedContent: {
            type: 'string',
            default: ''
        }
    },

    edit: ({ attributes, setAttributes }) => {
        const [isGenerating, setIsGenerating] = useState(false);

        const handleGenerate = async () => {
            setIsGenerating(true);

            try {
                const response = await wp.apiFetch({
                    path: 'ai-content-generator/v1/generate',
                    method: 'POST',
                    data: {
                        prompt: attributes.prompt,
                        tone: 'professional',
                        length: 'medium',
                        language: 'ukrainian'
                    }
                });

                // Очікуємо завершення генерації
                const content = await waitForCompletion(response.session_id);
                setAttributes({ generatedContent: content.content });

            } catch (error) {
                console.error('Generation error:', error);
            } finally {
```

Рисунок 3.8 – Gutenberg блок для генерації контенту

Метод init_hooks() відіграє ключову роль у налаштуванні взаємодії плагіна з ядром WordPress. Він реєструє низку дій (actions) та фільтрів (filters),

які визначають поведінку плагіна в різних контекстах. Дія admin_menu дозволяє плагіну додавати власні пункти меню в адміністративній панелі WordPress, створюючи зручний інтерфейс для користувачів. Дія admin_enqueue_scripts відповідає за завантаження JavaScript та CSS файлів лише на необхідних сторінках, що оптимізує продуктивність та запобігає конфліктам з іншими плагінами.

Реєстрація REST API маршрутів через дію rest_api_init є критично важливою для забезпечення взаємодії між фронтенд-компонентами на React та бекенд-сервісами генерації контенту. Ці маршрути надають програмний інтерфейс для відправки запитів на генерацію, перевірки статусу операцій та публікації готового контенту. Дія init використовується для реєстрації кастомних блоків Gutenberg, що розширює можливості стандартного редактора WordPress.

Основним елементом користувацького інтерфейсу є метод add_admin_menu(), який створює головний пункт меню “AI Generator” в адміністративній панелі WordPress. Розташування пункту на 30 позиції гарантує його помітність без перекриття стандартних елементів меню. Для обмеження доступу використовується здатність edit_posts, що дозволяє користуватися плагіном усім користувачам, які мають право редагувати публікації. Додатково створюється підменю для налаштувань, доступне лише користувачам з правами адміністратора (manage_options).

На рисунку 3.9 зображено скриншот інтерфейсу адміністративної панелі плагіна “AI Content Generator” у системі WordPress, що демонструє процес налаштування параметрів генерації вебконтенту за допомогою штучного інтелекту.

На зображенні представлено головну сторінку плагіна з характерним для WordPress дизайном та структурною організацією елементів інтерфейсу. У верхній частині інтерфейсу розташовано логотип плагіна “WP AI Content Generator”, що вказує на інтеграцію з екосистемою WordPress. Під логотипом відображено основний заголовок «Генератор AI-контенту», який чітко ідентифікує функціональне призначення даного інструменту.

Центральну частину інтерфейсу займає форма налаштування параметрів генерації, організована у вигляді структурованих розділів. Кожен розділ містить поля введення та вибору, що дозволяють точно налаштувати характеристики майбутнього контенту. У розділі «Тема або запит» представлено текстове поле з прикладом запиту: «Наприклад: Стаття про штучний інтелект в веброзробці», що слугує зразком для користувача та демонструє типовий сценарій використання плагіна.

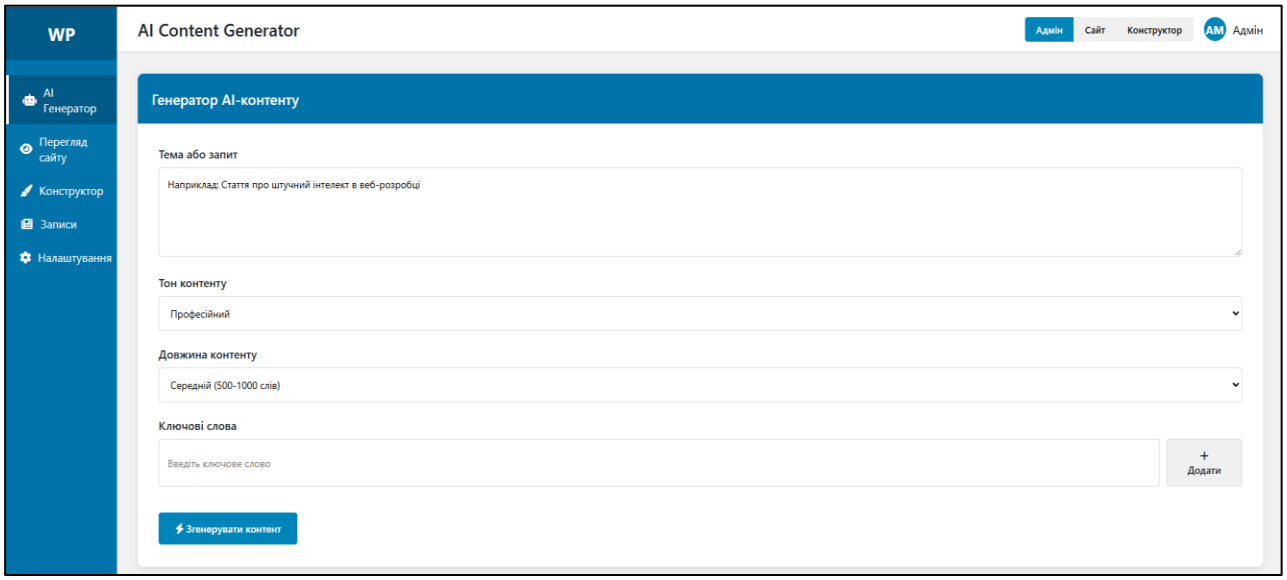


Рисунок 3.9 – Адмін панель плагіна “AI Content Generator”

Наступний розділ «Тон контенту» містить випадаючий список з варіантами стилістичного оформлення тексту. На скриншоті вибрано значення «Професійний», що вказує на можливість генерувати контент у формальному, діловому стилі, придатному для корпоративних сайтів, бізнес-блогів або технічної документації.

Розділ «Довжина контенту» пропонує вибір обсягу генерованого матеріалу. На зображенні активовано опцію «Середній (500-1000 слів)», що відповідає стандартній довжині інформаційної статті або блог-поста. Цей параметр дозволяє контролювати деталізацію та глибину розкриття теми.

Особливу увагу привертає розділ «Ключові слова», де представлено інтерактивний інтерфейс для управління SEO-параметрами контенту. На

скриншоті відображено два доданих ключових слова: «Веб» та «Інтелект», кожне з яких оформлено у вигляді окремого тегу з кнопкою видалення. Така реалізація дозволяє користувачеві швидко додавати, переглядати та редагувати список ключових слів безпосередньо перед генерацією контенту.

У нижній частині інтерфейсу розташована кнопка «Згенерувати контент», виконана в характерному для WordPress стилі з використанням кольорової гами адміністративної панелі. Кнопка містить іконку блискавки, що символізує швидкість та ефективність процесу генерації.

Дизайн інтерфейсу виконано у відповідності до стандартів WordPress UI, з використанням типових кольорів (#0073aa для акцентних елементів), шрифтів та відступів. Форма має чітку візуальну ієрархію, де кожен розділ відокремлений достатнім простором, що покращує читабельність та зручність використання. Елементи інтерфейсу мають послідовне розташування зверху вниз, що відповідає природному потоку заповнення форми.

На рисунку 3.10 зображено інтерфейс генератора контенту за допомогою штучного інтелекту для створення професійної статті.

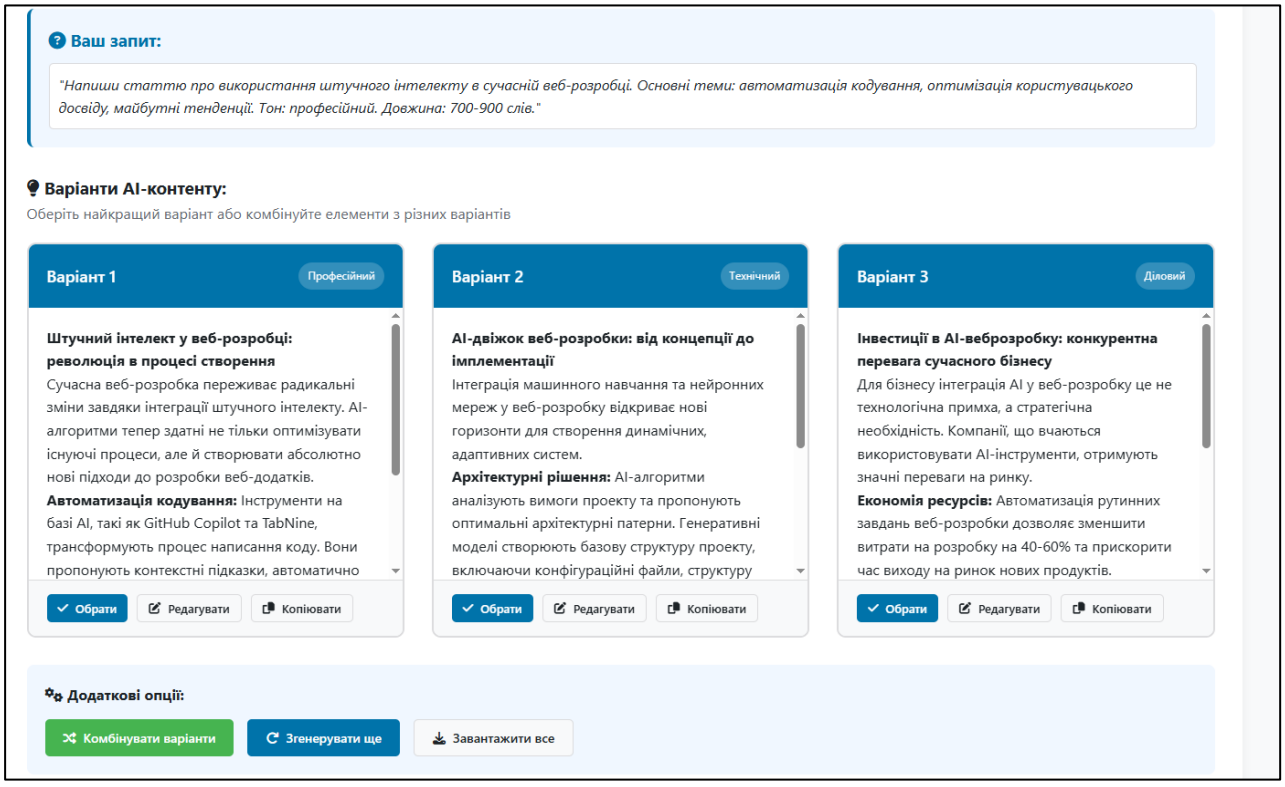


Рисунок 3.10 – Генератор контенту

Екран містить наступні елементи: Запит користувача (User prompt), вказано оригінальне технічне завдання – написати статтю про використання ШІ у веброзробці, з конкретними темами та вимогами до довжини та стилю. Нижче представлено три варіанти AI-контенту (AI content variants), де система пропонує три різні версії (або початкові фрагменти) статті, кожна з яких має свою стилістику та акценти: «Представницький» (загальноінформаційний), «Технічний» (деталізований) та «Діловий» (орієнтований на бізнес-переваги). Кожен варіант супроводжується кнопками «Обрати», «Редагувати» та «Копіювати».

Внизу екрану розташовані додаткові опції, такі як «Комбінувати варіанти», «Згенерувати ще» та «Завантажити все», що дозволяють користувачеві гнучко керувати результатом генерації, інтегруючи ідеї з різних пропозицій або отримуючи нові.

На рисунку 3.11 зображено інтерфейс конструктора вебсайтів з інтеграцією штучного інтелекту.

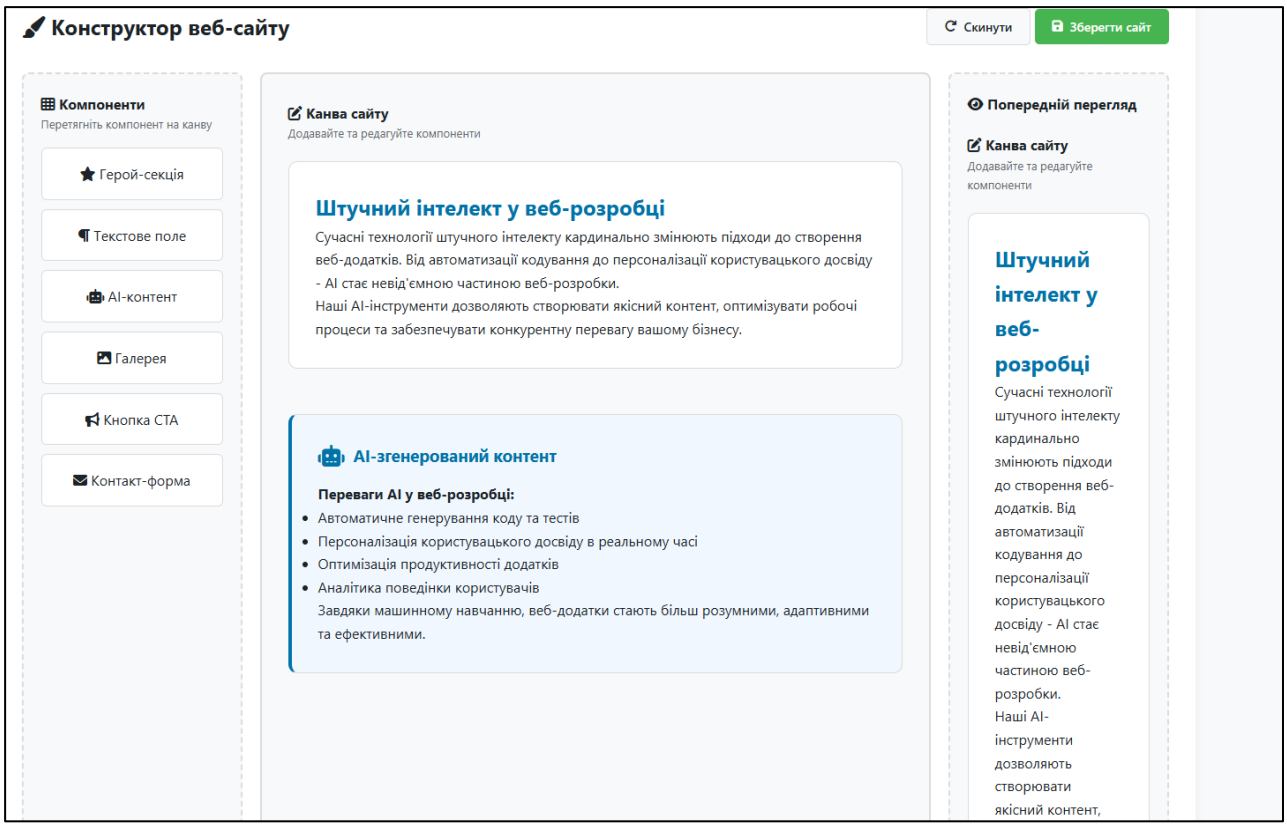


Рисунок 3.11 – Конструктор

Інтерфейс розділено на дві основні колонки. У лівій колонці знаходиться панель компонентів під заголовком «Компоненти». У цій секції доступні для перетягування такі елементи, як «Герой-секція», «Текстове поле», а також чекбокси для вибору додаткових функцій: «AI-контент», «Галерея», «Кнопка CTA», «Контакт-форма». Центральна частина інтерфейсу позначена як «Каркас сайту» і призначена для додавання та редагування компонентів. В ній уже розміщено готовий блок з контентом, який має заголовок «Штучний інтелект у веброзробці» і відповідний текстовий опис. Під ним знаходиться ще один блок – «AI-згенерований контент», що містить маркований список переваг використання штучного інтелекту.

У правій колонці представлений «Попередній перегляд» майбутньої вебсторінки. Він візуально відображає структуру та контент із центральної панелі редагування, демонструючи, як виглядатиме фінальна сторінка зі створеними секціями.

Таким чином, рисунок ілюструє інструмент для візуального створення вебсайтів, де розробник може перетягувати компоненти, наповнювати їх AI-згенерованим контентом і миттєво бачити результат у панелі попереднього перегляду.

ВИСНОВКИ

У кваліфікаційній роботі розроблено вебсервіс для управління завданнями з адаптивним інтерфейсом на основі технологій Node.js та React.js. Проведене дослідження охопило декілька ключових етапів: аналіз предметної області, формування вимог до системи, проєктування архітектури, розробка користувацького інтерфейсу та створення програмної реалізації.

На етапі дослідження предметної області було проаналізовано сучасні інструменти управління проєктами та виявлено потребу у створенні вебсервісу, здатного забезпечити гнучкість, швидкодію, зручність у використанні та адаптивність до різних пристроїв. Результати аналізу визначили основні функціональні можливості системи та лягли в основу її проєктування.

Проєктування системи включало розробку структурної та логічної архітектури, моделювання прецедентів використання, визначення ключових компонентів серверної та клієнтської частин, а також обґрунтування вибору технологій. Застосування Node.js дозволило забезпечити продуктивне опрацювання запитів і масштабованість серверної частини, а використання React.js дало можливість реалізувати динамічний, адаптивний та зручний інтерфейс користувача. Це забезпечило узгодженість програмних модулів та ефективність роботи всієї системи.

У процесі розробки користувацького інтерфейсу основну увагу зосереджено на забезпеченні зручного доступу до функціоналу системи з різних пристроїв, включно з мобільними. Реалізація компонентної структури сприяла простоті підтримки та розширюваності проєкту.

Етап програмної реалізації продемонстрував ефективність обраних технологій та методів. Серверну частину було реалізовано з урахуванням принципів модульності, безпеки та можливості масштабування. Клієнтська частина забезпечила інтуїтивну роботу з завданнями, включно зі створенням, редагуванням, впорядкуванням та контролем виконання. Була досягнута

узгодженість між фронтендом і бекендом, що забезпечило стабільну роботу програмного продукту.

Розроблена система дозволяє оптимізувати робочі процеси, підвищує ефективність управління завданнями та може використовуватися як у невеликих командах, так і індивідуальними користувачами.

Перспективними напрямами розвитку системи можуть бути інтеграція з зовнішніми сервісами, наприклад, календарями або комунікаційними платформами, розширення функціоналу аналітики, впровадження системи ролей та прав доступу, додавання можливостей командної співпраці у реальному часі та розширення адаптивних можливостей інтерфейсу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Brown T. B. et al. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*. 2020. Vol. 33. P. 1877-1901. URL:https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf (дата звернення: 15.06.2025).

2. GPT-4 Technical Report. *OpenAI*. 2023. URL: https://cdn.openai.com/papers/gpt-4.pdf (дата звернення: 25.06.2025).

3. Ouyang L. et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*. 2022. URL:https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf (дата звернення: 03.07.2025).

4. Radford A., Narasimhan K. Improving Language Understanding by Generative Pre-Training. *OpenAI*. 2018. URL: https://cdn.openai.com/research-covers/language unsupervised/language_understanding_paper.pdf (дата звернення: 10.07.2025).

5. Floridi L., Chiriatti M. GPT-3: Its Nature, Scope, Limits, and Consequences. *Minds and Machines*. 2020. Vol. 30. P. 681-694. URL: https://doi.org/10.1007/s11023-020-09548-1 (дата звернення: 16.07.2025).

6. Bommasani R. et al. On the Opportunities and Risks of Foundation Models. *Stanford University Center for Research on Foundation Models (CRFM)*. 2021. URL: https://arxiv.org/abs/2108.07258 (дата звернення: 18.07.2025).

7. Chowdhery A. et al. PaLM: Scaling Language Modeling with Pathways. *Journal of Machine Learning Research*. 2022. Vol. 24. P. 101-113. URL: https://www.jmlr.org/papers/volume24/22-1144/22-1144.pdf (дата звернення: 22.09.2025).

8. Raffel C. et al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*. 2020. Vol. 21. P. 1-67. URL: https://jmlr.org/papers/volume21/20-074/20-074.pdf (дата звернення: 05.10.2025).

9. Devlin J. et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* 2019. P. 4171-4186. URL: https://aclanthology.org/N19-1423.pdf (дата звернення: 12.10.2025).

10. Zhao W. X. et al. A Survey of Large Language Models. *arXiv preprint*. 2023. URL: https://arxiv.org/abs/2303.18223 (дата звернення: 20.10.2025).

11. Liu Y. et al. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint*. 2019. URL: https://arxiv.org/abs/1907.11692 (дата звернення: 08.11.2025).

12. Touvron H. et al. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint*. 2023. URL: https://arxiv.org/abs/2302.13971 (дата звернення: 15.11.2025).