

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

на тему:

**«Розробка сервісу автоматичного генерування контенту та
оптимізації управління сайтами блогів з використанням
машинного навчання»**

Виконав студент групи ЗППЗм-24-2
ЛІСОВСЬКИЙ Микола Сергійович

Керівник роботи:
ЛОБАНЧИКОВА Надія Миколаївна

Рецензент:
БАКАЛЮК Тетяна Анатоліївна

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

«ЗАТВЕРДЖУЮ»

зав. кафедри інженерії програм-
ного забезпечення

Тетяна ВАКАЛЮК

«18» вересня 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу

Здобувач вищої освіти: **ЛІСОВСЬКИЙ Микола Сергійович**

Керівник роботи: **ЛОБАНЧИКОВА Надія Миколаївна** Тема роботи: **«Розробка сервісу автоматичного генерування контенту та оптимізації управління сайтами блогів з використанням машинного навчання»**, затверджена Наказом закладу вищої освіти від **«18» вересня 2025 р. № 492/с.**

Вихідні дані для роботи: Об'єктом дослідження є алгоритми генерації тексту на основі рекурентних нейронних мереж. Предметом дослідження є методи, алгоритми та моделі машинного навчання, зокрема LSTM-мережі, а також принципи їх інтеграції у веб-системи для автоматичної генерації та управління контентом. Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Лобанчикова Н.М.	22.09.2025р.	22.09.2025р.
2	Лобанчикова Н.М.	01.10.2025р.	01.10.2025р.
3	Лобанчикова Н.М.	03.11.2025р.	03.11.2025р.

РЕФЕРАТ

Метою кваліфікаційної роботи є розробка інтелектуального сервісу автоматичного генерування текстового контенту та централізованої системи оптимізації управління мережею сайтів з використанням методів машинного навчання.

Об'єктом дослідження є процеси генерації тексту на основі рекурентних нейронних мереж.

Предметом дослідження є методи, алгоритми та моделі машинного навчання, зокрема LSTM-мережі, а також принципи їх інтеграції у веб-системи для автоматичної генерації та управління контентом.

Теоретична значущість роботи полягає у дослідженні ефективності використання архітектури LSTM для формування вузькотематичних корпусів текстів мовою оригіналу та обґрунтуванні гібридної архітектури взаємодії між високонавантаженими веб-фреймворками та обчислювальними ML-модулями.

Практична цінність роботи полягає у створенні діючого прототипу CRM-системи, що дозволяє автоматизувати процес наповнення великої кількості незалежних вебресурсів унікальним контентом. Розроблена система забезпечує високу швидкість генерації, асинхронну обробку завдань через механізм черг та гнучке управління параметрами, що значно знижує витрати на підтримку мережі сайтів.

Магістерська кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел, із 31 найменувань (3 сторінок), 2-х додатків (12 сторінок). Повний обсяг роботи становить 83 сторінки, з них 69 сторінок основного тексту.

КЛЮЧОВІ СЛОВА: CRM-СИСТЕМА, НЕЙРОННІ МЕРЕЖІ, LSTM, ГЕНЕРАЦІЯ КОНТЕНТУ, LARAVEL, PYTHON, PYTORCH, DOCKER, АВТОМАТИЗАЦІЯ SEO.

ABSTRACT

The objective of the qualification work is to develop an intelligent service for automated text content generation and a centralized system for optimizing the management of a website network using machine learning methods.

The object of the research is the text generation algorithms based on recurrent neural networks.

The subject of the research is the methods, algorithms, and models of machine learning, specifically LSTM networks, as well as the principles of their integration into web systems for automated content generation and management.

The theoretical significance of the work lies in investigating the effectiveness of the LSTM architecture for forming narrow-thematic text corpora in the source language and substantiating a hybrid architecture for interaction between high-load web frameworks and computational ML modules.

The practical value of the work consists in creating a functional CRM system prototype that allows for the automated population of a large number of independent web resources with unique content. The developed system ensures high generation speed, asynchronous task processing via a queue mechanism, and flexible parameter management, which significantly reduces the costs of maintaining a website network.

The Master's qualification work consists of an introduction, three chapters, conclusions, and a list of references with 31 titles (3 pages), and 2 appendices (12 pages). The total volume of the work is 83 pages, including 69 pages of the main text.

KEYWORDS: CRM SYSTEM, NEURAL NETWORKS, LSTM, CONTENT GENERATION, LARAVEL, PYTHON, PYTORCH, DOCKER, SEO AUTOMATION.

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ.....	9
1.1 Постановка задачі.....	9
1.2 Аналіз аналогів програмного продукту	10
1.3 Огляд та обґрунтування вибору фреймворку для створення CRM-системи	14
1.4 Теоретичні основи машинного навчання для генерації тексту	18
1.5.Обґрунтування вибору інструментальних засобів для розробки програмного комплексу	21
Висновки до першого розділу.....	21
РОЗДІЛ 2.ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО КОМПЛЕКСУ ГЕНЕРАЦІЇ КОНТЕНТУ ТА УПРАВЛІННЯ МЕРЕЖЕЮ САЙТІВ	25
2.1. Розробка нейромережевого модуля генерації тексту на основі архітектури LSTM.....	25
2.2. Проектування об'єктно-орієнтованої архітектури CRM системи	33
2.3. Розробка CRM системи	37
2.4. Розгортання проєкту та зв'язування LSTM моделі з CRM.....	47
Висновки до другого розділу	50
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПРОГРАМНИМ КОМПЛЕКСОМ.....	51
3.1 Аналіз результатів навчання та генерації нейромережевого модуля	51
3.2. Тестування інтерфейсу адміністратора та системи управління конфігураціями.....	55
3.3. Тестування механізмів асинхронної взаємодії та черг генерації контенту між CRM та LSTM моделлю	58
3.4. Валідація динамічного відображення контенту та структурних елементів сателіта.....	61

Висновки до розділу 3	65
ВИСНОВКИ	66
ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	68
ДОДАТКИ.....	71

ВСТУП

У сучасних умовах стрімкого розвитку цифрових технологій та зростання конкуренції в інформаційному середовищі особливо важливим стає ефективне й оперативне наповнення веб-ресурсів якісним контентом.

Розвиток машинного навчання відкриває для користувачів нові можливості. Завдяки цьому люди зможуть автоматизувати свою працю, оскільки традиційні підходи до створення сайтів з унікальним контентом вимагають значних часових та фінансових ресурсів, що обмежує масштабування платформ і знижує їхню конкурентоспроможність. Використання машинного навчання дає змогу прискорити створення матеріалів та адаптувати контент під потрібні тематики.

Водночас, актуальність теми підсилюється необхідністю оптимізації управління ресурсами. Сучасні платформи генерують великі обсяги даних, що потребує інтелектуальних рішень для автоматичної класифікації контенту, персоналізації видачі та підвищення загальної релевантності публікацій. Таким чином, інтеграція інтелектуальних алгоритмів є ключовою для переходу від ручного адміністрування до масштабованої та саморегульованої системи контент-менеджменту.

Метою дослідження є розробка сервісу для автоматичного генерування контенту та оптимізації управління сайтами блогів з використанням машинного навчання. Для досягнення поставленої мети необхідно вирішити такі основні завдання:

1. Обрати та обґрунтувати метод побудови CRM-системи для керування мережею сайтів.
2. Дослідити принципи роботи LSTM-моделей та їх застосування для генерації текстового контенту.
3. Проаналізувати алгоритми та інструменти машинного навчання, що використовуються для автоматичної генерації текстів та прогнозування тематичної інформації.

4. Розробити архітектуру сервісу, що поєднує PHP (Laravel) для керування сайтами та Python для ML-модулів.
5. Створити прототип модуля автоматичної генерації контенту, адаптованого до тематичного наповнення блогів.
6. Розробити механізм автоматичного вибору і відображення категорій контенту для кожного сайту.
7. Провести тестування розробленого сервісу та оцінити його ефективність у порівнянні з ручним контент-менеджментом.

Об'єктом дослідження є процеси генерації тексту на основі рекурентних нейронних мереж.

Предметом дослідження є методи, алгоритми та моделі машинного навчання, зокрема LSTM-мережі, а також принципи їх інтеграції у веб-системи для автоматичної генерації та управління контентом.

Наукова новизна роботи полягає в удосконаленні методів глибокого машинного навчання до задачі автоматизованого наповнення вебресурсів на основі аналізу корпусів без необхідності залучення копірайтерів. Особливістю запропонованого підходу є інтерпретація тематичних текстових масивів як послідовностей токенів з подальшим формуванням векторних представлень та використання рекурентної моделі LSTM для прогнозування наступних елементів послідовності. Це дозволяє зосередити увагу не просто на механічному копіюванні даних, а на генерації унікальних та семантично зв'язних статей.

Практична цінність роботи полягає у розробці програмного прототипу інтелектуальної CRM-системи, яка дозволяє прогнозувати та автоматично генерувати необхідні обсяги контенту без необхідності ручного написання кожним адміністратором окремо. Запропонований підхід може бути використаний у процесі розробки та оптимізації великих мереж вебресурсів для попереднього наповнення їх релевантною інформацією, що сприяє значному скороченню витрат часу та фінансових ресурсів на копірайтинг і підвищенню ефективності пошукової оптимізації всього проєкту.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ

1.1 Постановка задачі

Цифровий простір швидко розширюється, тому для бізнесів та інформаційних ресурсів життєво важливо мати якісний контент на своїх веб-сайтах. Статті на сайтах дуже важливі в стратегії контент-маркетингу, оскільки вони генерують органічний трафік за допомогою релевантних статей, оптимізованих за пошуковими запитами для охоплення цільової аудиторії. Крім того, часта публікація високоякісного контенту дозволить сайтам виглядати як експертні ресурси у своїй галузі, і з часом сайт матиме більше відвідувачів, яких можна буде перетворити на клієнтів.

Проте, при управлінні мережею блог-сайтів або великими інформаційними порталами, виникають суттєві проблеми, які обмежують масштабування та підвищують витрати. Одна з головних проблем полягає у надмірних потребах в ресурсах: кожен сайт у мережі вимагає постійного оновлення, що створює високий попит на велику кількість ресурсів, таких як контент-менеджери, копірайтери та редактори. Підтримка такого штату стає економічно не вигідною при спробі швидкого масштабування. Іншою критичною проблемою є необхідність забезпечення унікальності та релевантності контенту. Потрібно генерувати великі обсяги тексту, який має бути унікальним для уникнення санкцій пошукових систем, і водночас максимально адаптованим під специфічну тематику кожного окремого сайту чи категорії, що є складним завданням для ручного виконання. Окремим викликом є неефективність ручного управління розподіленою мережею ресурсів. Без централізованих інструментів автоматизації контроль за публікаціями та моніторинг показників на великій кількості доменів стає інертним процесом, схильним до операційних помилок, що унеможлиблює швидке розширення системи. Для досягнення цієї глобальної мети необхідно вирішити складну задачу,

що вимагає розробки гібридної системи, яка поєднує можливості централізованого управління та інтелектуальної генерації контенту.

Ключовим завданням є створення CRM-системи, яка буде керувати мережею незалежних блог-сайтів. Ця система, що буде розроблена на базі веб-фреймворку, має підтримувати мультидоменну структуру, забезпечуючи для кожного домену індивідуальні налаштування, унікальний набір статей та власні ієрархії категорій. Така централізована CRM стане єдиною точкою для моніторингу та планування контенту, а також для контролю параметрів кожного підключеного ресурсу.

Паралельно з CRM-системою необхідно створити LSTM модуль. Основна функція цього модуля полягає в автоматичному генеруванні статей на основі навченої моделі, яка отримує знання зі зразків інших статей. У цьому дослідженні передбачається використання LSTM-моделей для мовного моделювання, що дозволить створювати логічно зв'язний, граматично коректний і тематично адаптований контент.

Фінальним і критично важливим завданням є забезпечення надійної інтеграції цих двох компонентів. Згенеровані ML-модулем статті повинні передаватися до CRM-системи через програмний інтерфейс (API). CRM-система, у свою чергу, використовуватиме цей API для ініціалізації завдань генерації та для отримання готового контенту, який потім буде автоматично розподілено та опубліковано на відповідному сайті в мережі, згідно з його індивідуальними налаштуваннями. Таким чином, формалізація задачі зводиться до аналізу, обґрунтування та практичної реалізації цієї гібридної архітектури.

1.2 Аналіз аналогів програмного продукту

У сучасній інженерії програмного забезпечення існує кілька підходів до побудови мереж сайтів, серед яких найбільш розповсюдженим є використання універсальних систем керування контентом, як WordPress. Проте архітектура WordPress базується на монолітному принципі, що створює значне системне навантаження при спробі масштабування до сотень ресурсів на одному сервері.

Велика кількість задіяних PHP-процесів та важке ядро системи вимагають надмірних апаратних ресурсів, що робить утримання великих мереж економічно неефективним. Крім того, WordPress не має вбудованих інструментів інтелектуального навчання, тому генерація контенту в ньому реалізується виключно через сторонні плагіни, які працюють як посередники до платних пропрієтарних API. Це створює критичну залежність від зовнішніх провайдерів та унеможливорює навчання моделі на локальних вузькоспеціалізованих наборах даних. Інтерфейс створення статті в WordPress зображено на рис. 1.1.

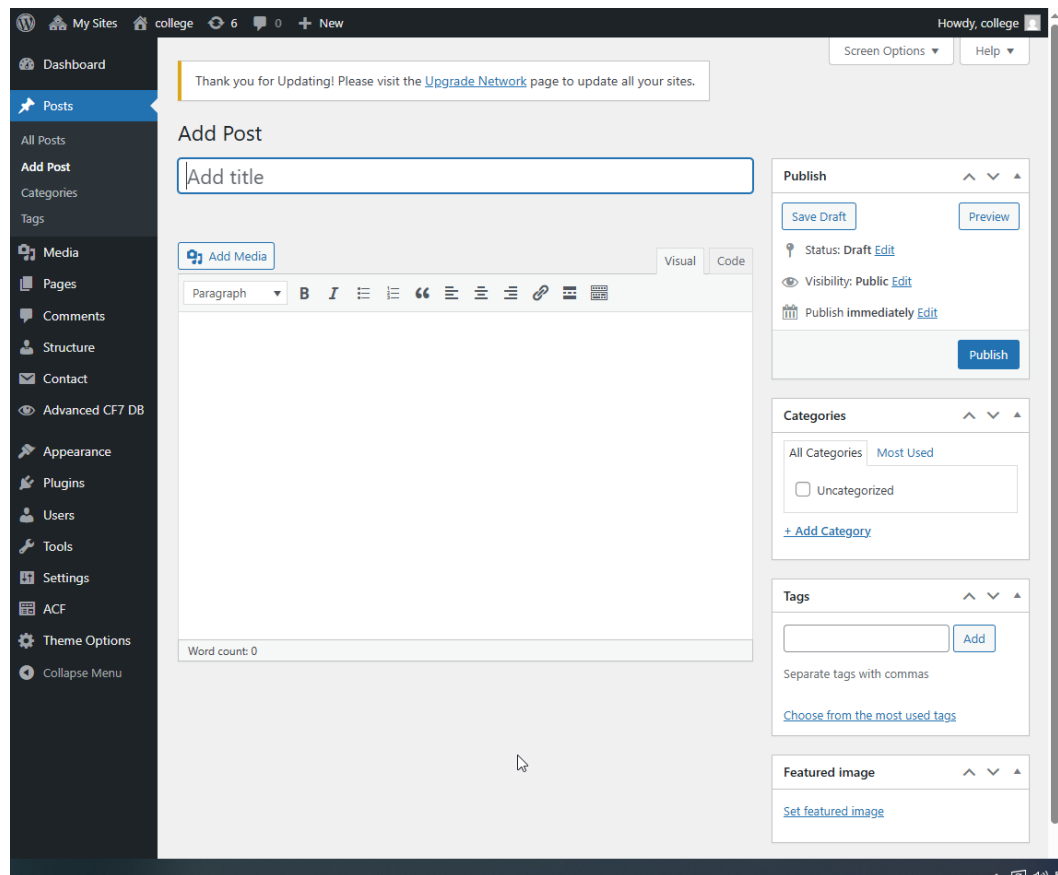


Рис 1.1. Створення статті в WordPress

Іншим сучасним аналогом є платформа Ghost, яка позиціонується як швидка та професійна альтернатива для видавців. Ghost побудований на базі Node.js і працює значно швидше за класичні системи на PHP, пропонуючи сучасний інтерфейс та вбудовані можливості для SEO. Однак Ghost орієнтований на створення поодиноких високоякісних медіа-ресурсів, а не на масове

автоматизоване розгортання сателітів. Впровадження в таку систему власного модуля машинного навчання вимагає складної перебудови архітектури, оскільки платформа не передбачає нативної підтримки важких обчислень на PyTorch або TensorFlow. Як і у випадку з іншими CMS, Ghost залишається закритою екосистемою в розрізі генеративного штучного інтелекту, змушуючи розробника покладатися на зовнішні хмарні сервіси генерації. Інтерфейс платформи Ghost зображено на рис. 1.2.

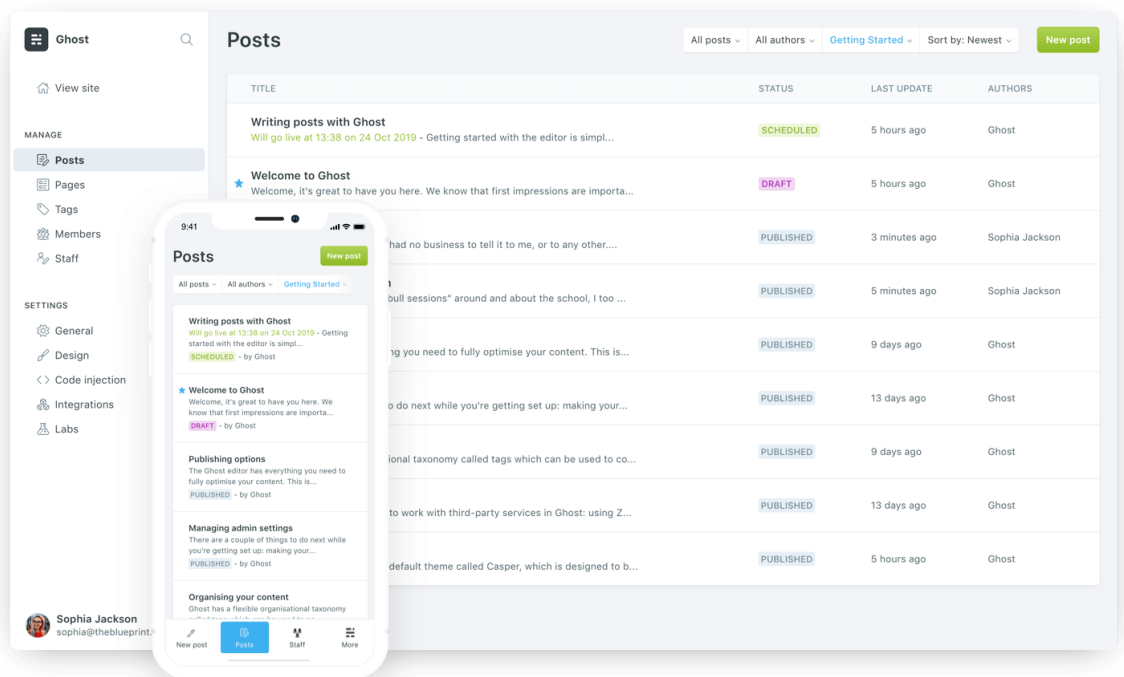


Рис 1.2. Інтерфейс платформи Ghost

Ще одним вагомим аналогом є Joomla, яка історично займає нішу між простим WordPress та складними корпоративними системами. З погляду інженерії ПЗ, Joomla пропонує більш просунуту ієрархію прав доступу та вбудовану підтримку багатомовності, проте її архітектура вважається однією з найскладніших серед open-source рішень. Використання Joomla для створення мереж є недоцільним через надмірну складність налаштування кожного окремого екземпляра та складної системи, яке створює високе навантаження на базу даних.

У контексті штучного інтелекту Joomla залишається найбільш інертною системою: екосистема плагінів для роботи з нейронними мережами розвинена слабо, а інтеграція власної LSTM-моделі в її запутану структуру об'єктно-орієнтованого коду вимагала б невиправдано великих трудовитрат. Інтерфейс платформи Joomla зображено на рис. 1.3.

Для обґрунтування розробки спеціалізованого програмного продукту було проведено аналіз найбільш розповсюджених систем управління контентом за критеріями, що є критичними для побудови автоматизованих мереж сайтів.

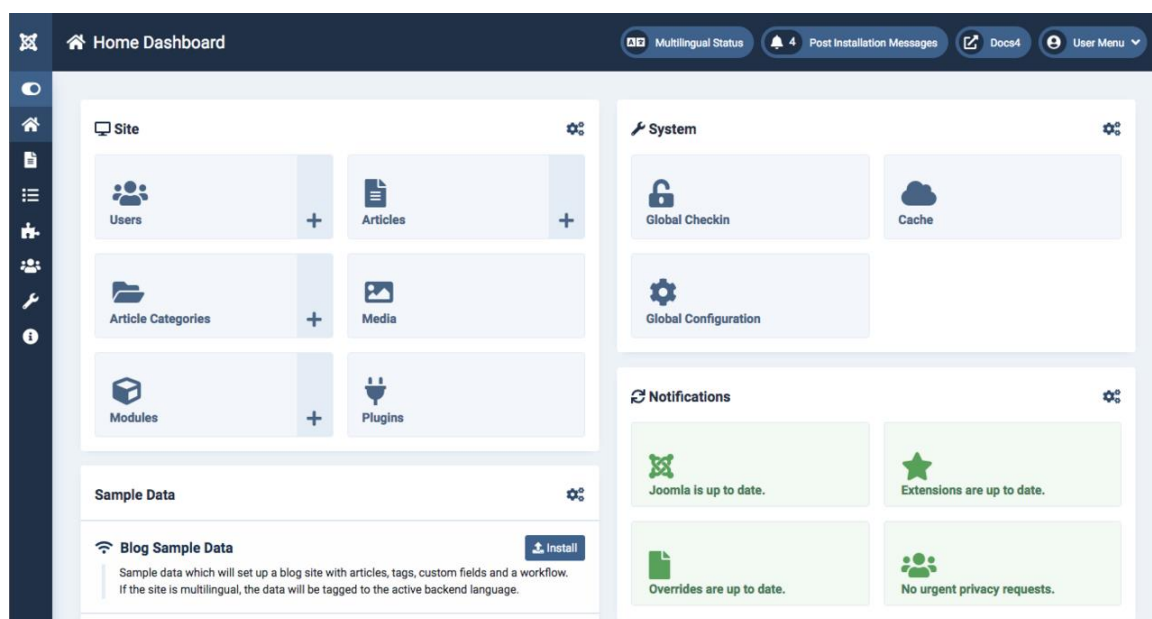


Рис 1.3. Інтерфейс платформи Joomla

Таблиця 1.1

Порівняльна характеристика програмних продуктів

	WordPress	Joomla	Ghost
Архітектурний паттерн	Моноліт (PHP)	Моноліт (PHP/MVC)	Headless/Decoupled (Node.js)
Споживання пам'яті (RAM)	Високе (через плагіни)	Високе (складна структура)	Середнє
Складність розгортання мережі	Складно (кожен сайт окремо)	Дуже складно	Помірно

	WordPress	Joomla	Ghost
Інтеграція ШІ (ML)	Тільки через зовнішні API	Відсутня (тільки плагіни)	Тільки через API
Гнучкість навчання моделі	Неможливо	Неможливо	Неможливо
Залежність від витрат	Висока (плата за API)	Середня	Середня

Аналіз, наведений у таблиці, демонструє, що традиційні CMS, такі як WordPress, Joomla, Ghost орієнтовані на створення універсальних ресурсів, що призводить до надмірності коду та високих вимог до серверної інфраструктури при масовому розгортанні. Ghost, хоча і є сучасним рішенням на базі Node.js, не забезпечує необхідного рівня автономності в питаннях генерації контенту, залишаючись залежним від сторонніх сервісів.

1.3 Огляд та обґрунтування вибору фреймворку для створення CRM-системи

Для ефективного управління мережею незалежних блог-сайтів необхідна розробка централізованої CRM-системи, що має відповідати високим архітектурним та функціональним вимогам. Головна вимога полягає у підтримці складної мультидоменної структури. Це означає, що єдиний бекенд повинен обслуговувати кілька сайтів, забезпечуючи при цьому повну ізоляцію даних, дозволяючи кожному домену мати власні налаштування, унікальний набір статей та індивідуальну ієрархію категорій, зберігаючи єдиний адміністративний інтерфейс для управління. З огляду на необхідність інтеграції зі складним обчислювальним модулем генерації контенту, система має бути спроектована на основі API-First підходу, а також мати вбудовану підтримку асинхронних операцій для ефективної обробки довготривалих завдань генерації.

1. Порівняльний аналіз фреймворків та вибір фреймворка для CRM-системи

Для реалізації бекенду CRM-системи, яка має керувати мережею блог-сайтів та інтегруватися з модулем штучного інтелекту, необхідно обрати надійний, гнучкий та масштабований веб-фреймворк. Було розглянуто три провідні технологічні платформи, які використовуються для розробки корпоративних рішень: Laravel (PHP), Django (Python) та Ruby on Rails (Ruby).

Django є фреймворком на Python, який використовує архітектуру, близьку до MVC (так звану MTV - Model-Template-View). Його сильна сторона полягає у філософії "батареї в комплекті" (batteries included), що забезпечує швидке розгортання проєкту. Крім того, його нативна інтеграція з екосистемою Python робить його привабливим для проєктів, що мають справу з Data Science та Machine Learning. Ruby on Rails відомий своєю швидкістю розробки, досягнутою завдяки принципу "Конвенція понад конфігурацією", що значно знижує кількість необхідного коду. Проте, складність його асинхронних операцій та повільніша динаміка розвитку спільноти останнім часом є певним обмеженням.

Laravel, як сучасний PHP-фреймворк, демонструє найкраще поєднання характеристик, які є критичними для гібридної архітектури. Він суворо дотримується патерну MVC (Model-View-Controller), що забезпечує чіткий поділ логіки, даних та інтерфейсу, що є критично важливим для масштабованості та підтримки такої складної CRM-системи. Його об'єктно-реляційна модель Eloquent ORM ефективно спрощує роботу зі складними реляційними базами даних, дозволяючи керувати необхідними для мультидоменної структури зв'язками між доменами, статтями та категоріями. На відміну від Django та Rails, екосистема Laravel пропонує потужні інструменти для роботи з чергами (Queue) та розробки REST API. Саме наявність вбудованої підтримки асинхронної обробки завдань через черги робить Laravel ідеальним кандидатом для безшовної інтеграції з обчислювальним ML-модулем на Python, оскільки дозволяє CRM-системі передавати завдання генерації у фоновому режимі, не блокуючи основний інтерфейс користувача. Виходячи з порівняльного аналізу (табл. 1.2), Laravel було обрано як основну технологію бекенду.

Для зберігання всієї конфігураційної інформації та контенту було обрано реляційну систему керування базами даних MySQL. Вибір саме реляційної моделі обґрунтовується необхідністю обробки складних зв'язків типу «один-до-багатьох» (наприклад, один сайт містить багато категорій, а одна категорія - безліч статей) та «багато-до-багатьох» (у випадку тегування контенту). Оскільки конфігурації сайтів, індивідуальні налаштування доменів та згенеровані тексти мають чітко визначену структуру, критично важливим є дотримання принципу цілісності даних (ACID). Це гарантує, що будь-яка транзакція - від оновлення налаштувань DNS до публікації згенерованої статті - буде виконана повністю або не виконана зовсім, запобігаючи появі «сирітських» записів або розсинхронізації контенту.

Особлива увага при виборі СУБД приділялася її взаємодії з Eloquent ORM, що є частиною фреймворку Laravel. Використання ORM дозволяє абстрагуватися від написання сирих SQL-запитів, представляючи таблиці бази даних як об'єкти мови PHP. Це не лише підвищує швидкість розробки та полегшує підтримку коду, але й забезпечує додатковий рівень безпеки, автоматично захищаючи систему від SQL-ін'єкцій. Крім того, реляційна структура MySQL дозволяє ефективно використовувати індексацію для повнотекстового пошуку по згенерованих статтях, що є необхідним для швидкої видачі контенту на фронтенд-частині мережі сайтів.

Таблиця 1.2

Порівняльна характеристика фреймворків

Критерій	Laravel (PHP)	Django (Python)	Ruby on Rails (Ruby)
Архітектурний патерн	MVC (Model-View-Controller)	MTV (Model-Template-View)	Convention Over Configuration
ORM	Eloquent (гнучка та інтуїтивна)	Django ORM (менш гнучка для складних запитів)	Active Record

Підтримка Асинхронності	Вбудована підтримка черг (Queue) через Redis/DB	Нативна підтримка Async/Await (версії 3+)	Потребує додаткових бібліотек
Швидкість розробки	Висока, завдяки Artisan CLI та великій кількості готових пакетів	Висока (вбудовано багато компонентів)	Висока (завдяки філософії Rails)

2. Аналіз інструментів для швидкої розробки адміністративної панелі. Розробка функціональної та інтуїтивно зрозумілої адміністративної панелі (адмін-панелі) є критично важливою для CRM-системи, оскільки саме тут контент-менеджери будуть керувати доменами, статтями та ініціювати генерацію контенту. Для прискорення цього процесу і забезпечення високої якості коду доцільно використовувати генератори boilerplate-коду. Серед таких інструментів для Laravel були розглянуті Laravel Breeze/Jetstream, Laravel Boilerplate та Craftable Laravel Generator.

Laravel Jetstream пропонує високоінтегроване рішення з повною авторизацією, але його фокус більше спрямований на автентифікацію, ніж на генерацію CRUD-інтерфейсів для управління даними. Laravel Boilerplate надає готову структуру проєкту, але часто вимагає значних доопрацювань для створення специфічних CRUD-функцій.

Натомість, для цього проєкту було обрано Craftable Laravel Generator. Його основна перевага полягає у фокусі на генерації повного CRUD-функціоналу (Create, Read, Update, Delete) для адміністративної частини. Craftable дозволяє автоматично створювати моделі, контролери, маршрути, міграції та Vue.js-інтерфейс для будь-якої сутності в системі (наприклад, "Домени", "Статті", "Категорії"). Це значно прискорює створення базового функціоналу адмін-панелі та дозволяє розробникам зосередитися на більш складних завданнях, таких як інтеграція з ML-модулем та реалізація мультидоменної логіки. Використання

Craftable Generator гарантує одноманітність коду, високу швидкість розробки та ефективне використання ресурсів.

3. Вибір реляційної бази даних для збереження конфігурації сайтів.

Для зберігання всієї конфігураційної інформації та контенту, включно з налаштуваннями доменів, інформацією про статті та ієрархіями категорій, було обрано використання реляційної бази даних (SQL). Це рішення обґрунтовується насамперед необхідністю управління структурованими зв'язками у мультидоменній системі. Конфігурації сайтів та контент мають чітко визначені залежності: наприклад, кожна стаття повинна бути однозначно прив'язана до конкретного домену та категорії, а конфігурація домену - до його ідентифікатора.

Вибір SQL-рішення, такого як PostgreSQL або MySQL, є переважним у порівнянні з нереляційними базами даних (NoSQL), такими як MongoDB або Cassandra. Хоча бази даних NoSQL забезпечують високу гнучкість схеми та горизонтальне масштабування, вони часто жертвують гарантіями цілісності даних. У контексті CRM-системи, де першочерговою є надійність зв'язку між ключовими сутностями, реляційна модель ідеально підходить для таких ієрархічних даних, оскільки гарантує цілісність даних відповідно до принципів ACID (Атомарність, Узгодженість, Ізольованість, Довговічність). Це є критично важливим для системи, де втрата або порушення зв'язку між доменом та його наповненням є неприпустимим. Використання SQL забезпечує надійність, передбачуваність та високу продуктивність при роботі з великою кількістю записів і, що особливо важливо, ідеально інтегрується з об'єктно-реляційною моделлю Eloquent ORM фреймворку Laravel.

1.4 Теоретичні основи машинного навчання для генерації тексту

Машинне навчання є галуззю штучного інтелекту, що займається створенням алгоритмів і моделей, які дозволяють комп'ютерам навчатися на основі даних. Це дозволяє розробляти алгоритмічну модель для прийняття рішень і прогнозування подій без прямого втручання людини шляхом програмування правил дій для

виконання певних процедур. Створення унікального контенту сьогодні має широке коло застосувань, оскільки освіта, комерція, медіа та творчість інтегрують ці технології у свою діяльність для ефективного керування часом, продуктивністю та розробкою продуктів. Підприємство може використовувати моделі для швидкого генерування описів, а преса відстежує тренди через аналіз даних і формує контент відповідно до інтересів споживачів. Важливо обрати модель, яка найкраще підходить для конкретних потреб, враховуючи типи навчання, такі як навчання з учителем, без учителя, напівкероване або підкріплене навчання. Система дозволяє автоматично формувати мета-теги та анонси для покращення індексації великої кількості сторінок. Завдяки налаштуванню параметрів генерації сервіс здатен адаптувати стилістику та термінологію під конкретні тематичні розділи мережі. З огляду на широкий спектр застосувань, генерація тексту стала активною галуззю досліджень, де важливою задачею є створення моделей, які генерують текст з високою якістю, враховуючи граматику, стиль та логічну послідовність. У контексті цієї роботи використання LSTM-моделей дозволяє побудувати ефективні рішення для автоматичного створення текстів, що застосовуються у системах підтримки та аналізі природної мови.

Генерація текстового контенту полягає у створенні послідовностей слів або цілих текстів, які мають сенс і відповідають поставленим вимогам. Існує кілька основних підходів до цієї задачі, серед яких виділяють метод правил та шаблонів, що базується на заздалегідь заданих структурах, проте має обмежену гнучкість. Статистичні моделі n -грам прогнозують наступне слово на основі попередніх елементів, але не здатні враховувати довготривалий контекст. Рекурентні нейронні мережі працюють із послідовностями змінної довжини, проте мають проблеми з навчанням через затухання або вибух градієнтів. Спеціалізованим типом RNN є мережі довгострокової короткострокової пам'яті, що включають механізми пам'яті для збереження інформації про довгий контекст і довели ефективність у перекладі та розпізнаванні мови. Найбільш технологічним рішенням на сьогодні є архітектури, що здатні аналізувати весь обсяг вхідного тексту одночасно, а не

послідовно слово за словом. Це дозволяє системі миттєво визначати найважливіші зв'язки між віддаленими частинами тексту, що забезпечує найвищу якість генерації, проте такі обчислення потребують надпотужного серверного обладнання

У даній роботі представлено підхід до навчання LSTM-моделі для генерації тексту на рівні символів, де основною метою є покращення якості шляхом оптимізації процесу навчання. Для цього текстовий корпус перетворюється у послідовність індексів, що відповідають унікальним символам словника. Використовується шар вбудовування для формування векторних представлень, а LSTM-комірка налаштовується для генерації наступних символів. Для оцінки точності роботи моделі використовується математична функція, яка обчислює різницю між згенерованим текстом та оригіналом, визначаючи ступінь відхилення. На основі цих даних застосовується спеціальний алгоритм оптимізації, який поступово корегує внутрішні налаштування нейромережі. Це дозволяє системі з кожним кроком навчання зменшувати кількість помилок і робити вихідний контент більш схожим на природну людську мову. Результати показують, що модель імітує структуру та стиль вихідного тексту, а оптимізація швидкості навчання та регуляризація покращують якість контенту. Підготовка якісних даних включає очищення тексту від HTML-тегів та шумів, нормалізацію через приведення до нижнього регістру та усунення зайвих пробілів. Токенізація розбиває текст на слова або символи для збереження семантики. Побудова словника має балансувати між охопленням лексики та обмеженням складності, оскільки великий словник потребує більше ресурсів. Для LSTM необхідно формувати вхідні та цільові послідовності, правильно встановлюючи їх довжину для балансу між контекстом і розміром пакетів. Ретельна підготовка даних створює основу для успішного навчання та стабільності роботи моделі.

Генерація тексту за допомогою LSTM стикається з низкою викликів, серед яких обмеження у захопленні дуже довгих залежностей, що може призводити до втрати контексту. Обчислювальна складність багатоваріантних моделей потребує значних ресурсів і часу, навіть з використанням GPU. При недостатньо великому

навчальному корпусі виникає проблема перевчання, коли модель запам'ятовує фрази замість узагальнення патернів. Параметр довжини послідовності обмежує кількість попередніх слів, які бачить модель, що впливає на зв'язність тексту. Ефективність навчання безпосередньо залежить від точного налаштування внутрішніх конфігурацій моделі, зокрема від обсягу цифрової пам'яті для зберігання значень слів та загальної кількості обчислювальних вузлів нейромережі. Неправильний вибір цих характеристик може призвести до того, що згенерований текст буде беззмістовним або міститиме велику кількість логічних помилок. Іноді LSTM генерують повторювані фрази, що виправляється налаштуванням температури вибору наступного слова. Також моделі складно захоплюють глибокий зміст і логічні зв'язки, що призводить до смислово порожніх фрагментів.

1.5. Обґрунтування вибору інструментальних засобів для розробки програмного комплексу

Реалізація архітектури системи автоматизованої генерації контенту вимагає поєднання потужних засобів управління веб-інтерфейсами та високоефективних інструментів для роботи з нейронними мережами.

Основним інструментом серверної розробки обрано мову PHP у поєднанні з фреймворком Laravel. Даний вибір зумовлений високим рівнем безпеки, наявністю вбудованих механізмів обробки асинхронних черг та потужною ORM-системою Eloquent для маніпуляції складними структурами даних. Використання даного стека дозволить забезпечити швидку обробку запитів користувача та надійну взаємодію з реляційною базою даних.

Візуальна частина адміністративної панелі та клієнтських сайтів базуватиметься на класичному поєднанні HTML5 та CSS3, що гарантує валідність розмітки та адаптивність інтерфейсу. Для забезпечення динамічної взаємодії користувача з елементами керування, зокрема для моніторингу статусу генерації статей у реальному часі, доцільно впровадити прогресивний JavaScript-фреймворк Vue.js. Використання його компонентного підходу та технології Virtual DOM

дозволить створити реактивний інтерфейс, що значно покращить користувацький досвід при роботі з великими масивами даних.

Ядром інтелектуальної обробки даних виступатиме мова Python, яка є галузевим стандартом у сфері машинного навчання. Вибір фреймворку PyTorch для реалізації LSTM-моделі обґрунтований його динамічним графом обчислень, що спрощує процес експериментування з архітектурою нейромережі. Додатково планується використання бібліотек NumPy для матричних операцій, NLTK для глибокої попередньої обробки лінгвістичних корпусів та Matplotlib для візуалізації процесу навчання і відстеження збіжності моделі. Використання мікрофреймворку Flask дозволить виділити ШІ-модуль у незалежний мікросервіс, ізолювавши важкі математичні обчислення від основного веб-сервера.

Надійне збереження структурованої інформації про користувачів, конфігурації вебресурсів та історію генерованого контенту забезпечуватиме реляційна система керування базами даних SQL. Даний вибір зумовлений її високою швидкістю обробки операцій читання та запису, що є критично важливим для стабільної роботи CRM-систем. Використання SQL у поєднанні з ORM Laravel дозволить реалізувати надійні транзакції та забезпечити цілісність даних на рівні зв'язків між таблицями. Крім того, наявність інструменту phpMyAdmin спростить процес адміністрування та візуальної перевірки структури БД на етапах розробки.

Для забезпечення безперебійного зв'язку між CRM-панеллю та LSTM-моделлю обрано протокол HTTP REST API із передачею даних у форматі JSON. Оскільки процес створення тексту нейромережею є ресурсомістким, для запобігання блокуванню інтерфейсу користувача буде впроваджено систему асинхронних черг на базі Redis. Вибір даного сховища зумовлений його високою швидкістю роботи в оперативній пам'яті, що дозволить миттєво реєструвати завдання та відстежувати їхній статус.

Для ефективного керування залежностями програмних модулів планується використання менеджерів пакетів Composer (для PHP) та Pip (для Python). Це гарантуватиме сумісність усіх бібліотек, зокрема бібліотеки Guzzle для здійснення

зовнішніх запитів. Безпосереднє написання коду здійснюватиметься в інтегрованих середовищах PhpStorm та PyCharm, які пропонують найкращі інструменти для статичного аналізу та дебагінгу.

Цілісність усього середовища розробки та легкість подальшого розгортання забезпечуватиметься використанням технології контейнеризації Docker. Це дозволить об'єднати PHP-контейнер з Laravel та Python-контейнер з нейронною мережею у єдину екосистему, де вони взаємодіятимуть через внутрішню мережу. Такий підхід гарантує ідентичність умов роботи системи на етапах розробки та фінального релізу. Для управління ресурсами у локальному середовищі використовуватиметься Docker Desktop, а контроль версій та паралельна розробка модулів здійснюватиметься за допомогою системи Git, що забезпечить надійне збереження історії змін та стабільність програмного комплексу.

Для забезпечення високої якості коду в процесі розробки планується застосування методів статичного аналізу, вбудованих у середовища PhpStorm та PyCharm. Ці інструменти дозволяють виявляти потенційні помилки ще на етапі написання скриптів. Для тестування взаємодії між модулями через API планується використання інструменту Postman, що дасть змогу імітувати запити до Python-мікросервісу та перевіряти коректність форматів передачі даних перед їх інтеграцією у фронтенд-частину.

Висновки до першого розділу

У першому розділі було проведено комплексний аналіз проблеми масштабування контент-менеджменту та обґрунтовано необхідність створення спеціалізованої гібридної системи, що поєднує можливості централізованого управління та інтелектуальної генерації тексту.

Проведений критичний огляд існуючих аналогів, таких як WordPress, Ghost та Joomla, дозволив виявити їхні основні недоліки в контексті поставленої задачі: високу архітектурну надмірність, значне споживання ресурсів при масовому розгортанні та складність інтеграції власних обчислювальних моделей.

Встановлено, що більшість сучасних CMS орієнтовані на використання платних зовнішніх API, що створює критичну залежність від провайдерів та обмежує можливості навчання моделей на локальних спеціалізованих наборах даних.

На основі аналізу архітектурних вимог для розробки CRM-системи було обрано PHP-фреймворк Laravel. Його вибір обґрунтовано наявністю потужної Eloquent ORM для управління складними реляційними зв'язками мультидоменної структури та вбудованою системою черг, що є критично важливою для асинхронної взаємодії з ML-модулем. Для забезпечення цілісності даних обрано реляційну СУБД MySQL, яка гарантує виконання принципу ACID при маніпуляціях з великими обсягами конфігураційної інформації.

Теоретичне дослідження методів машинного навчання підтвердило доцільність використання архітектури Long Short-Term Memory (LSTM) для генерації текстового контенту. Встановлено, що даний тип рекурентних нейронних мереж є найбільш збалансованим рішенням для прототипування систем генерації, оскільки дозволяє ефективно працювати з контекстними залежностями мови. Окрему увагу було приділено етапам препроцесингу даних (токенізації та очищенню корпусу), що безпосередньо впливають на якість фінального результату.

Підсумком розділу стало визначення інструментального стека, де для ізоляції важких математичних обчислень обрано мову Python з бібліотекою PyTorch, а для об'єднання компонентів у стабільне середовище технологію контейнеризації Docker. Таким чином, було сформовано теоретичний та технологічний фундамент для подальшої практичної реалізації системи.

РОЗДІЛ 2.ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО КОМПЛЕКСУ ГЕНЕРАЦІЇ КОНТЕНТУ ТА УПРАВЛІННЯ МЕРЕЖЕЮ САЙТІВ

2.1. Розробка нейромережевого модуля генерації тексту на основі архітектури LSTM

Одним із ключових етапів розробки інтелектуальної системи є підготовка якісного набору даних (dataset). Для рекурентних нейронних мереж типу LSTM обсяг та релевантність даних є критичними факторами: чим більшим і різноманітнішим є навчальний корпус, тим краще модель здатна виявляти закономірності у природній мові та генерувати змістовні тексти.

Для навчання моделей генерації та класифікації тексту необхідно було зібрати реальні текстові дані. В межах цієї роботи джерелом даних стала стаття з відкритого доступу на сайті artificialintelligence-news.com. Вибір зумовлений високою якістю англomовного технічного контенту, чіткою структурою статей та великою кількістю тематичного матеріалу, що дозволяє моделі вивчити специфічну термінологію та стилістику написання текстів про штучний інтелект.

Для автоматизації процесу збору даних було розроблено скрипт-парсер, який виконує ітеративний обхід сторінок ресурсу. Технічна реалізація базується на вилученні текстового вмісту з HTML-тегів <article> та блоків із класом entry-content. Це дозволяє відфільтрувати навігаційні елементи, рекламу та коментарі, залишаючи лише чистий інформаційний контент.

```
def download_article(url):  
    headers = {"User-Agent": "Mozilla/5.0"}  
    try:  
        r = requests.get(url, headers=headers, timeout=10)  
        r.raise_for_status()  
        soup = BeautifulSoup(r.content, "html.parser")  
        paragraphs = soup.find_all("p")
```

50)

```
text = "\n".join(p.get_text(strip=True) for p in paragraphs if len(p.get_text(strip=True)) >
if not text:
    text = soup.get_text(separator=' ', strip=True)
    text = text[text.find('article'):text.find('related posts')] # Обмеження
    text = text[:8000]
    text = re.sub(r'\s+', ' ', text).strip()
    return "Article Title", text
except requests.exceptions.RequestException as e:
    return "", ""
```

Після збору дані піддаються нормалізації: приведення до нижнього регістру та фільтрація не алфавітних символів. Далі виконується токенізація для побудови ЧИСЛОВОГО СЛОВНИКА.

```
def clean_text(text):
    text = text.lower()
    text = re.sub(r"[^a-z0-9\s.,!?'-]", "", text, flags=re.UNICODE)
    return text

def remove_repeats(words, max_repeats=2):
    result = []
    count = 1
    for i in range(len(words)):
        if i == 0:
            result.append(words[i])
            continue
        if words[i] == words[i - 1]:
            count += 1
        else:
            count = 1
        if count <= max_repeats:
            result.append(words[i])
    return result
```

Всі зібрані статті об'єднуються, і стають основою для подальшої токенізації та векторизації. Такий підхід дозволяє сформувати навчальне середовище обсягом

у сотні тисяч слів, що забезпечує високу контекстуальну глибину майбутньої LSTM-моделі та дозволяє системі генерувати SEO-оптимізований контент, який за своєю структурою наближений до професійної журналістики.

Після очищення текст розбивається на окремі слова (токени). Для роботи моделі необхідно кожне слово зіставити з унікальним числовим індексом - формується словник:

```
words = text.split()
vocab = sorted(set(words))
word2idx = {w: i for i, w in enumerate(vocab)}
idx2word = {i: w for w, i in word2idx.items()}
vocab_size = len(vocab)
print("Кількість слів у словнику:", vocab_size)
```

Це дозволяє перетворювати послідовності слів у числові послідовності, придатні для навчання нейромережі. Заданою довжиною `seq_length` формується навчальна вибірка з тексту. Кожен елемент містить вхідну послідовність слів довжиною `seq_length` та цільову послідовність - ті ж самі слова, зсунуті на один крок вперед.

```
class TextDataset(Dataset):
    def __init__(self, data, seq_len):
        self.data = data
        self.seq_len = seq_len
    def __len__(self):
        return len(self.data) - self.seq_len
    def __getitem__(self, idx):
        return (
            torch.tensor(self.data[idx:idx + self.seq_len]),
            torch.tensor(self.data[idx + 1:idx + self.seq_len + 1])
        )
```

Використання `DataLoader` дозволяє ефективно передавати дані в модель під час навчання пакетами (batch-wise):

```
dataset = TextDataset(data, seq_length)
```

```
loader = DataLoader(dataset, batch_size=16, shuffle=True)
```

Після виконання цих кроків текст готовий для подачі в модель. Отримано векторизовану версію тексту та словник розміром `vocab_size`, який використовується для побудови ембедінгів у LSTM-моделі. Генерація текстового контенту є послідовнісною задачею, для якої найкраще підходять рекурентні нейронні мережі (RNN). Проте класичні RNN мають обмежену здатність запам'ятовувати довгі залежності. Для подолання цієї проблеми було використано модифікацію RNN - LSTM (Long Short-Term Memory).

Основними компонентами моделі є Embedding-шар, LSTM-шар та Linear-шар. Embedding-шар перетворює індекси символів у вектори фіксованої розмірності, що дозволяє моделі краще навчати відношення між символами. LSTM-шар виступає основним рекурентним компонентом, який обробляє послідовність та «запам'ятовує» інформацію на довгих відстанях. Останній Linear-шар перетворює вихід LSTM у вектор ймовірностей для кожного можливого наступного символу.

Стан комірки та її регулятори є основними елементами традиційної архітектури LSTM. Одиницею пам'яті мережі є стан комірки, де зберігається інформація, яка може бути записана, прочитана або збережена з попереднього кроку за допомогою відкритих і закритих воріт. Інформація з попередніх етапів може нести відповідне навантаження протягом всієї обробки послідовності. Аналогові керуючі вентилі, що реалізуються через множення на тангенс або сигмоїдну функцію, функціонують як фільтри, визначаючи, яку саме інформацію дозволено пропускати далі.

Використовуючи рекурентний процес навчання нейронної мережі, ворота обирають, які дані запам'ятовувати, а які відкидати протягом навчання. Класична модель машинного навчання LSTM складається зі стану комірки та трьох воріт: воріт забуття, входу та виходу. Особливістю LSTM є техніка воріт у кожній комірці. Функція активації $\tanh$ використовується для обробки входу на кожному

часовому кроці та прихованого шару з попереднього часового кроку для створення нового прихованого стану та виходу в стандартній комірці RNN.

Воріт забуття визначає, що важливо зберегти з попереднього стану комірки. Простіше кажучи, він видаляє зі стану комірки інформацію, яка більше не є корисною. Це перший блок, представлений в архітектурі. Сигмоїдна функція активації використовується для передачі інформації з поточного входу і попереднього прихованого стану. Ворота отримують два входи - вхід у певний момент часу та попередній вихід комірки, які перемножуються з ваговими матрицями перед додаванням зсуву. Результат надсилається у функцію активації, яка видає двійкове значення. Якщо вихідне значення близьке до 0, це означає забути, а якщо близьке до 1 - зберегти.

Вхідний вентиль вирішує, яку частину інформації в поточному стані комірки LSTM блока слід оновити. Він слугує джерелом інформації про стан комірки. Він складається з двох частин: спочатку ми використовуємо сигмоїдну функцію, щоб визначити, які значення будуть оновлені на основі попереднього прихованого стану і поточного входу. Потім, щоб керувати мережею, подаємо ті самі два входи. Функція $\tanh$ виробляє вектор зі значеннями від -1 до +1, що містить всі можливі значення. Щоб витягти значущу інформацію, значення вектора і керованих значень перемножуються.

Вихідний вентиль визначає поточний прихований стан, який потім передається наступному LSTM-блоку. Прихований стан використовується для прогнозування і містить інформацію з попередніх входів. Поточний прихований стан регулюється вихідним вентилем. Сигмоїдна функція задається попереднім прихованим станом і поточним входом. Поточний прихований стан отримується множенням цього виходу на вихід функції тангенса. Значення вектора і регульовані значення перемножуються і подаються на вихід і вхід наступної комірки. Принцип роботи LSTM зображено на рис. 2.1.

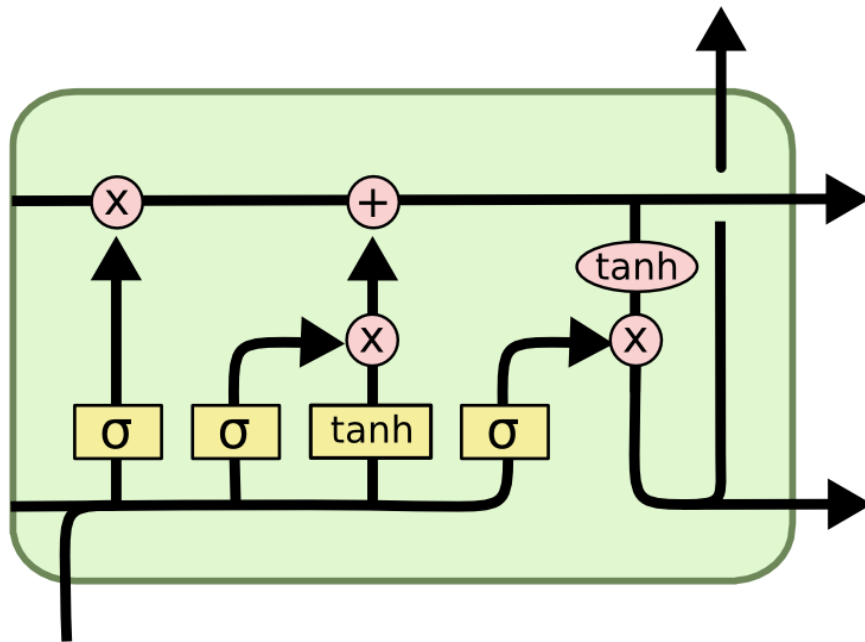


Рис.2.1. Принцип роботи LSTM

Лістинг класу LSTMModel

```
class LSTMModel(nn.Module):
    def __init__(self, vocab_size, embed_size=128, hidden_size=256, num_layers=3):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.lstm = nn.LSTM(embed_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, vocab_size)
    def forward(self, x, hidden=None):
        x = self.embedding(x)
        output, hidden = self.lstm(x, hidden)
        out = self.fc(output)
        return out, hidden
```

Пояснення параметрів:

vocab_size - кількість унікальних символів у тексті;

embedding_dim - розмірність векторного представлення символів;

hidden_dim - розмірність прихованого стану LSTM;

num_layers - кількість шарів LSTM (зазвичай 1 або 2 для простих моделей);

`batch_first=True` - забезпечує порядок розмірностей (batch, sequence, feature) у вхідному тензорі.

Після навчання моделі, текст генерується по символах: кожен символ подається в модель, яка на основі нього передбачає наступний. Потім цей передбачений символ знову подається як вхід, і процес повторюється. Таким чином, LSTM-модель здатна створювати осмислений текст, зберігаючи локальні та частково глобальні залежності, властиві оригінальному стилю.

Процес навчання моделі LSTM базується на ітеративному проходженні крізь весь підготовлений датасет. Кількість епох (повних циклів проходження через дані) була встановлена на рівні, що забезпечує мінімізацію помилки без ризику перенавчання (overfitting). На кожній епосі обчислюється функція втрат `CrossEntropyLoss`, яка порівнює передбачений моделлю токен з реальним наступним словом у послідовності. Оптимізація ваг відбувається за допомогою алгоритму, який адаптивно корегує швидкість навчання. Повний лістинг навчання моделі в додатку А.

```
model = LSTMModel(vocab_size).to(DEVICE)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
epochs = 300
for epoch in range(1, epochs + 1):
    model.train()
    total_loss = 0
    for x, y in loader:
        x, y = x.to(DEVICE), y.to(DEVICE)
        optimizer.zero_grad()
        out, _ = model(x)
        loss = criterion(out.view(-1, vocab_size), y.view(-1))
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    if epoch % 50 == 0 or epoch == 1:
```

```
print(f'Epoch {epoch}, Loss: {total_loss:.4f}')
```

Після завершення навчання критично важливо зберегти стан моделі та структуру словника. Словник (word2idx) необхідний для того, щоб при генерації нових статей вхідні слова перетворювалися на ті самі індекси, на яких навчалася мережа. Ваги моделі зберігаються у форматі .pth (стандарт PyTorch), що містить параметри всіх шарів: від embedding до рекурентних вентилів.

```
MODEL_PATH = "model_trained_en.pth"
VOCAB_PATH = "word2idx_en.pkl"
torch.save(model.state_dict(), MODEL_PATH)
with open(VOCAB_PATH, "wb") as f:
    pickle.dump(word2idx, f)
```

Було розроблено алгоритм генерації (Inference), який перетворює початкову фразу (seed text) у повноцінну статтю (повний лістинг коду генерації статей в додатку Б). Процес генерації базується на ітераційному прогнозуванні кожного наступного слова.

Особливістю реалізованого алгоритму є використання параметра температури (TS). Це дозволяє маніпулювати впевненістю моделі: при низьких значеннях TS вибір стає детермінованим (найбільш імовірне слово), а при вищих - варіативним. Після отримання ймовірностей з останнього шару моделі застосовується функція $\text{softmax}(\text{logits} / T)$, що дозволяє генерувати унікальні SEO-тексти, уникаючи повного копіювання навчального корпусу.

```
def generate_text(model, word2idx, idx2word, start_text, length=250,
seq_length=SEQ_LENGTH, temperature=0.8):
    model.eval()
    words_seq = clean_text(start_text).split()
    state = None
    device = next(model.parameters()).device
    for _ in range(length):
        indices = [word2idx.get(w, 0) for w in words_seq[-seq_length:]]
        if len(indices) < seq_length:
            indices = [0] * (seq_length - len(indices)) + indices
```

```

x = torch.tensor([indices], dtype=torch.long).to(device)
with torch.no_grad():
    out, state = model(x, state)
    last_word_logits = out[0, -1]
    p = torch.nn.functional.softmax(last_word_logits / temperature,
dim=0).detach().cpu().numpy()
    if len(p) > 0:
        next_idx = np.random.choice(len(p), p=p)
        next_word = idx2word.get(next_idx, "word")
        words_seq.append(next_word)
    else:
        break

```

Завдяки серіалізації навченої моделі та словника, ми отримуємо автономний модуль генерації. Це дозволяє використовувати модель у мікросервісному середовищі (наприклад, всередині Docker-контейнера з Flask API), де вона завантажується один раз при старті сервісу та миттєво відповідає на запити CRM-системи Laravel для створення нових статей. Такий підхід забезпечує високу продуктивність, оскільки ресурсомісткий процес навчання виконується одноразово, а в робочому режимі (inference) модель працює лише на прогнозування.

2.2. Проектування об'єктно-орієнтованої архітектури CRM системи

Проектування архітектури є критично важливим етапом розробки, оскільки система повинна забезпечувати стабільне та одночасне керування великою кількістю незалежних вебресурсів. Для реалізації цієї задачі було обрано фреймворк Laravel, який базується на фундаментальному архітектурному патерні MVC. Використання цього патерну дозволяє досягти високого рівня модульності, що є критичним при розробці складних систем, де бізнес-логіка повинна бути повністю відокремлена від інтерфейсу користувача.

Розподіл обов'язків у межах обраного архітектурного патерну реалізовано через чітку взаємодію трьох основних компонентів системи. Компонент моделі

(Model) безпосередньо відповідає за взаємодію з базою даних MySQL та втілює ключові бізнес-правила програмного комплексу. У межах проєкту розроблені моделі, зокрема такі як Site, Article та Category, не лише описують структуру даних, а й визначають складні логічні зв'язки між сутностями. Використання інструментарію Eloquent ORM дозволяє маніпулювати записами бази даних як об'єктами, що значно спрощує роботу з конфігураціями сайтів та забезпечує автоматичну валідацію типів даних на рівні коду.

Функції представлення (View) покладені на модулі, що відповідають за візуалізацію інформації та взаємодію з адміністратором системи. У програмному комплексі реалізовано два типи представлень, де адміністративна панель базується на шаблонізаторі Blade та компонентах Vue.js для реактивного відображення прогресу генерації контенту. Водночас фронтенд-частина керованих сайтів складається з динамічно сформованих сторінок, які відображають отриманий від нейромережі контент відповідно до заданих технічних конфігурацій.

Сполучною ланкою в цій архітектурі виступає контролер (Controller), який виконує роль посередника в інформаційному обміні. Він приймає вхідні HTTP-запити від користувача, ініціює взаємодію з відповідними моделями для отримання або модифікації даних у базі, після чого передає оброблений результат у представлення для відображення. Такий підхід дозволяє ізолювати логіку обробки даних від інтерфейсу, що підвищує стабільність системи та полегшує процес тестування окремих функціональних модулів.

Застосування архітектурного патерну MVC у межах даного проєкту дозволяє ефективно вирішити низку критичних інженерних задач, що виникають при розробці складних інтелектуальних систем. Перш за все, такий підхід забезпечує можливість паралельної розробки, коли команда або окремих розробників можуть одночасно працювати над реактивним інтерфейсом на базі Vue.js та внутрішньою логікою генерації контенту на Python або Laravel, не створюючи конфліктів у кодовій базі.

Крім того, чіткий розподіл обов'язків гарантує легкість масштабування програмного комплексу. Додавання нових функціональних можливостей, таких як аналітичні модулі для збору статистики відвідуваності або впровадження нових типів медіаконтенту, не вимагає глобального переписування існуючої логіки, а реалізується шляхом розширення системи новими моделями та відповідними контролерами. Важливим аспектом також є спрощення підтримки коду в довгостроковій перспективі. Завдяки детермінованій структурі патерну розробник може максимально швидко локалізувати та усувати технічні помилки, що є критично важливим для стабільного функціонування розгалуженої мережі сайтів та підтримки цілісності даних у централізованому сховищі.

Після визначення загальних архітектурних принципів та обрання патерну MVC, наступним етапом проектування стало формування логічної структури бази даних за допомогою інструменту SQL DBM. Проектування архітектури в спеціалізованих інструментах моделювання, таких як SQL DBM, є не просто допоміжним кроком, а фундаментальним етапом інженерії програмного забезпечення. Це дозволяє уникнути критичних помилок на ранніх стадіях, коли вартість їх виправлення є мінімальною. Було спроектовано логіку для адміністраторів, яка зберігає облікові дані, та надає систему ролей і дозволів, що забезпечує розмежований доступ до інструментів керування мережею. Структура бази даних адміністратора зображено на рис. 2.2.

Ключовим етапом проектування стало моделювання сутності конфігурації сайту. Оскільки кожен вебресурс у мережі повинен бути унікальним, модель бази даних була розроблена таким чином, щоб зберігати вичерпний набір параметрів для кожного домену. Це дозволяє централізовано керувати кожним сайтом як окремою одиницею, забезпечуючи при цьому цілісність зв'язків між сайтом, його категоріями та згенерованим контентом. Такий підхід до моделювання дозволяє CRM-системі виступати єдиною точкою істини для всієї інфраструктури, гарантуючи швидке розгортання нових ресурсів на основі вже існуючих

архітектурних шаблонів. Структура бази даних конфігурації сайтів зображено на рис. 2.3.

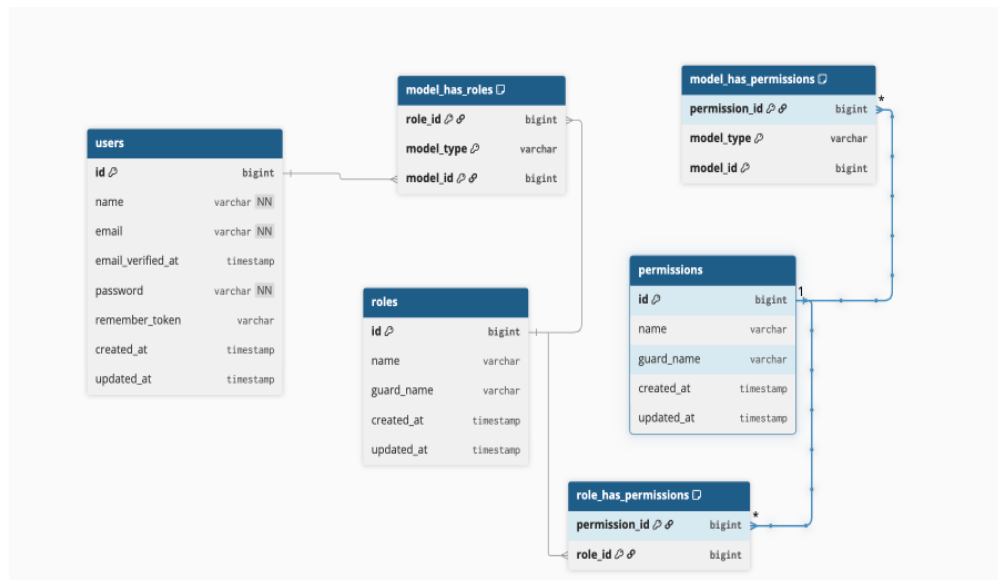


Рис. 2.2. Структура бази даних адміністраторів

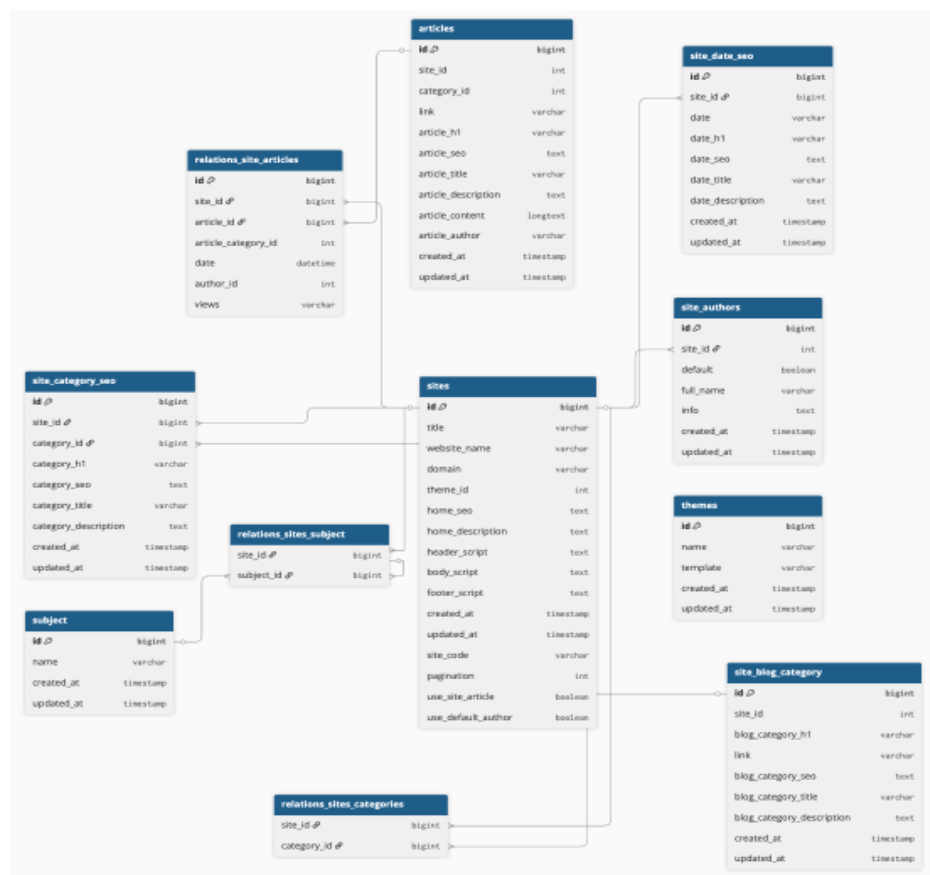


Рис. 2.3. Структура бази даних налаштувань сайтів

Спроектowana об'єктно-орієнтована модель стала основою для подальшої реалізації програмних модулів, дозволивши перейти від теоретичного планування до практичної розробки механізмів взаємодії між центральним вузлом управління та мережею сайтів.

2.3. Розробка CRM системи

Першочерговим завданням стала реалізація міграцій, які трансформують візуальну схему бази даних у фізичні таблиці СУБД MySQL. На відміну від прямого створення таблиць через SQL-запити, використання міграцій у Laravel забезпечує версійність структури даних. Це дозволяє синхронізувати стан бази даних та легко вертати зміни у разі виникнення помилок. Особлива увага приділялася таблиці `sites`, яка виступає головним репозиторієм налаштувань для кожного вузла мережі. Кожне поле в цій таблиці має чітке призначення. Міграції дозволяють не лише створювати таблиці, а й версіонувати структуру бази даних, що є стандартною практикою в інженерії ПЗ.

Приклад створення таблиці на моделі `sites`

```
Schema::create('sites', function (Blueprint $table) {  
    $table->id();  
    $table->string('title');  
    $table->string('website_name');  
    $table->string('domain');  
    $table->unsignedInteger('theme_id');  
    $table->text('home_seo')->nullable();  
    $table->text('home_description')->nullable();  
    $table->text('header_script')->nullable();  
    $table->text('body_script')->nullable();  
    $table->text('footer_script')->nullable();  
    $table->timestamps();  
});
```

На прикладі цієї міграції створюємо наступні таблиці зі зв'язками.

Для керування великою кількістю ресурсів була спроектована система маршрутів, що розділяє публічну частину та адміністративний інтерфейс. Безпека системи гарантується використанням middleware, які реалізують механізм автентифікації та авторизації. Це запобігає несанкціонованому доступу до функцій генерації контенту та налаштувань доменів.

Middleware - це механізм, який дозволяє фільтрувати та аналізувати HTTP-запити, що надходять до вашого додатка, ще до того, як вони потраплять до контролера. Якщо запит відповідає певній умові (наприклад, користувач авторизований), Middleware пропускає його далі. Якщо ні - він може перенаправити користувача або видати помилку (наприклад, 403 Forbidden).

```
Route::middleware(['auth:', config('admin-auth.defaults.guard'), 'admin'])->group(static function () {  
    Route::prefix('admin')->namespace('App\Http\Controllers\Admin')->name('admin')->group(sta  
tic function() {  
        Route::prefix('sites')->name('sites/')->group(static function() {  
            Route::get('/', 'SitesController@index')->name('index');  
            Route::get('/create', 'SitesController@create')->name('create');  
            Route::post('/store', 'SitesController@store')->name('store');  
            Route::get('/{site}/edit', 'SitesController@edit')->name('edit');  
            Route::post('/bulk-destroy', 'SitesController@bulkDestroy')->name('bulk-destroy');  
            Route::post('/{site}', 'SitesController@update')->name('update');  
            Route::get('/{site}', 'SitesController@view')->name('view');  
            Route::delete('/{site}', 'SitesController@destroy')->name('destroy');  
        });  
    });  
});
```

Для забезпечення ефективної роботи адміністратора з мережею сайтів було розроблено метод `index` у контролері `SiteController`. Головною інженерною особливістю цього методу є використання спеціалізованого сервісу `AdminListing`, який автоматизує процеси фільтрації, сортування та пагінації даних безпосередньо на рівні бази даних. Це дозволяє системі стабільно працювати навіть при наявності тисяч записів у таблиці `sites`.

Особлива увага приділена процесу збагачення даних (Data Enrichment). Оскільки в базі даних зберігаються лише ідентифікатори (ID), метод виконує ітеративну трансформацію отриманої колекції. За допомогою внутрішніх методів моделей Theme, Site та Group числові значення замінюються на зрозумілі для користувача назви категорій, тематик та груп. Це дозволяє винести складну логіку зіставлення даних у відповідні об'єкти, дотримуючись принципу єдиної відповідальності (Single Responsibility Principle). Лістинг коду методу index:

```
public function index(IndexSite $request)
{
    $data = AdminListing::create(Site::class)->processRequestAndGet(
        $request,
        ['id','title','use_default_author','seo_article_title','seo_article_description',
        'website_name','home_h1','version_img_gpt', 'pagination', 'domain', 'theme_id', 'home_description',
        'group','site_code','lack_count','default_count','pagination','use_site_article','version_chat_gpt','links'],
        ['id', 'title', 'site_code', 'lack_count', 'website_name', 'domain', 'home_seo',
        'home_description', 'header_script', 'body_script', 'footer_script','use_site_article','version_chat_gpt']
    );
    foreach ($data as &$item) {
        $item->theme_id = (new \App\Models\Theme)->getThemeById($item->theme_id);
        $item->subject = (new Site()->getCategoriesName($item);
        $item->category = (new Site()->getSubjectName($item);
        $item->group = (new Group()->getGroupById($item->group);
    }
    return view('admin.site.index', ['data' => $data]);
}
```

Аналогічною логікою було розроблено контролери для всіх ключових сутностей системи. Зокрема, було реалізовано повноцінні CRUD-інтерфейси (Create, Read, Update, Delete) для категорій, статей та інших моделей, що дозволяє адміністратору здійснювати повний контроль над контентом та конфігурацією сайта.

Після налаштування архітектури сайтів та підготовки нейромережевого модуля, було розроблено функціонал для запуску процесу створення статей безпосередньо з адміністративної панелі. Ключовим завданням було створення зручного інтерфейсу, який дозволяє адміністратору вводити "початкові фрази" (seed phrases), на основі яких модель LSTM буде розгорнути повноцінні тексти. Інтерфейс модального вікна в CRM зображено на рис. 2.4.

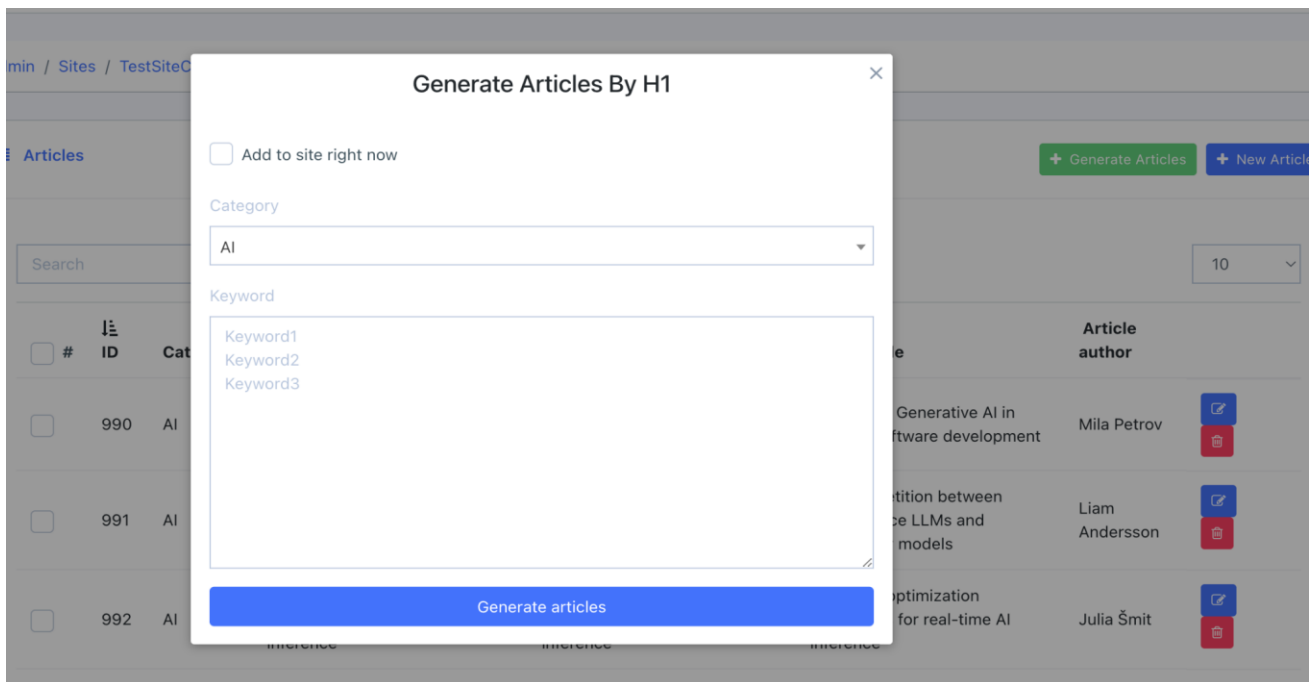


Рис. 2.4. Інтерфейс для створення статей через модель LSTM

У контролері `ArticleController` реалізовано метод `generateArticles`, який виступає точкою входу для процесу генерації. Його логіка побудована на валідації вхідних даних та підготовці масиву завдань. Особливістю реалізації є можливість пакетної обробки: адміністратор може ввести список заголовків, і система автоматично сформує чергу на генерацію для кожного з них. Лістинг методу `generateArticles`.

```
public function generateArticles(Request $request, Site $site)
{
    $h1Input = $request->input('h1');
```

```

$category = (int) $request->input('category');
$addRightNow = (bool) $request->input('add_right_now');
$startPhrases = array_filter(
    array_map('trim', explode("\n", $h1Input))
);
if (empty($startPhrases)) {
    return redirect('/admin')->with('error', '');
}
$count = (new ArticleGeneratorService())->dispatchGenerationJobs(
    $h1Input,
    $category,
    $site,
    $addRightNow
);
return redirect ('/admin/sites/' . $site->id . '/articles')
    ->with('success', "Add generate articles");

```

Впровадження системи асинхронної обробки даних стало необхідним кроком через специфіку роботи нейромережевого модуля, оскільки процес генерації тексту на базі архітектури LSTM вимагає значних часових витрат. Виконання таких запитів у межах стандартного життєвого циклу HTTP-запиту призвело б до блокування серверних ресурсів та неминучого виникнення таймаутів. Для розв'язання цієї інженерної задачі було застосовано механізм Laravel Jobs, який реалізує патерн черги повідомлень.

У межах розробленої архітектури контролер виконує роль ініціатора, який лише реєструє намір користувача згенерувати контент, формує відповідне завдання та миттєво повертає відповідь клієнту. Саме завдання переміщується у проміжне сховище, де воно очікує на обробку вільним процесом-воркером. Такий підхід гарантує високу відмовостійкість системи, адже навіть у разі тимчасової недоступності мікросервісу на Python, завдання не зникає, а залишається в черзі для повторної спроби виконання. Це забезпечує надійність збереження даних та

стабільність роботи адміністративної панелі незалежно від навантаження на нейронну мережу.

Особлива перевага такої реалізації полягає у можливості горизонтального масштабування. Система дозволяє запускати декілька паралельних обробників, що значно прискорює наповнення мережі сайтів контентом, оскільки запити до ШІ-моделі виконуються незалежно один від одного. Крім того, асинхронність позитивно впливає на користувацький досвід, надаючи адміністратору можливість продовжувати налаштування інших параметрів сайту, поки фонові процеси займаються інтелектуальною обробкою природної мови та збереженням результатів у базу даних. Лістинг handle методу класу GenerateSingleArticleJob.

```
public function handle(): void{
    $client = new Client();
    $hl_for_title = substr($this->startText, 0, 255);
    try {
        $response = $client->post('http://ml_generator:5000/generate', [
            'json' => [
                'start_text' => $this->startText,
                'length' => $this->length,
                'temperature' => $this->temperature],
            'timeout' => 50,]);
        $data = json_decode($response->getBody()->getContents(), true);
        if ($data['status'] === 'success') {
            $generatedContent = $data['generated_text'];
            $status = 'generated';
        } else {
            $errorMsg = 'ML Service Error: ' . ($data['message'] ?? 'Unknown error');
            throw new Exception($errorMsg);
        } catch (Exception $e) {
            $errorMsg = $e->getMessage();
        }
        $this->release(5);
    }
    return;
```

Для забезпечення функціонування мережі сайтів було розроблено архітектуру динамічних маршрутів. Головною особливістю цієї реалізації є те, що один набір маршрутів обслуговує сотні різних доменів, автоматично адаптуючи контент під конкретний HTTP_HOST.

У файлі web.php реалізовано логіку перевірки вхідного запиту. Якщо запит не стосується адміністративної панелі (/login), він проходить через групу маршрутів із посередником checksite. Це гарантує, що перед рендерингом сторінки система знайде відповідний сайт у базі даних CRM. Якщо домен не зареєстрований у системі, вона автоматично повертає помилку 404, що запобігає нецільовому використанню ресурсів сервера. Лістинг роутів для сайтів.

```
if (isset($_SERVER['REQUEST_URI'])) {  
    if (!str_contains($_SERVER['REQUEST_URI'], '/login')) {  
        Route::middleware(['checksite'])->group(static function () {  
            if (isset($_SERVER['HTTP_HOST'])) {  
                $site = (new Site)->getSiteByDomain($_SERVER['HTTP_HOST']);  
                if (empty($site)) {  
                    abort(404); }  
                Route::get('/', 'App\Http\Controllers\Front\HomeController@index')->name('home');  
                Route::get('/categories/{slug}',  
'App\Http\Controllers\Front\CATEGORIESController@index')->name('category');  
                Route::get('/{slug}',  
'App\Http\Controllers\Front\SinglePageController@index')->name('singlePage');  
                Route::get('/{year}/{month}',  
'App\Http\Controllers\Front\DatesController@index')->name('date');  
                Route::get('/category/{category_slug}/{slug}',  
'App\Http\Controllers\Front\ArticleController@index')->name('article');  
            });  
        }  
    }  
}
```

Структура кожного вебресурсу в межах мережі базується на наборі функціональних сторінок, що забезпечують повноцінну роботу сайту та його

ефективну індексацію. Головна сторінка відображає актуальний контент та ієрархію категорій, яка є специфічною для кожного обраного домену. Для покращення навігації та SEO-структури вебресурсу використовуються сторінки категорій, які забезпечують групування згенерованих статей за відповідними тематиками. Основною одиницею контенту є сторінка статті, де відображається повний текст, згенерований нейронною мережею, разом із метаданими, зображеннями та релевантними посиланнями. Додатково в архітектурі передбачені хронологічні архіви, що дозволяють фільтрувати контент за датою публікації у форматі року та місяця.

Для забезпечення коректної роботи розподіленої мережі сайтів було розроблено спеціалізований програмний клас `middleware` під назвою `CheckSite`. Його головна роль полягає в перехопленні кожного вхідного HTTP-запиту, ідентифікації цільового ресурсу за його доменним іменем та підготовці повного глобального контексту даних, необхідних для рендерингу сторінок. Логіка роботи цього посередника базується на декількох послідовних етапах, починаючи з ідентифікації домену через глобальний масив `$_SERVER['HTTP_HOST']`, що дозволяє системі точно визначити, до якого саме конфігу сайту звернувся користувач.

Наступним критично важливим етапом є забезпечення валідації та безпеки, під час якого `Middleware` перевіряє наявність відповідної конфігурації для даного домену в базі даних. У разі, якщо сайт не знайдено, система негайно перериває виконання запиту та повертає помилку 404, що надійно захищає серверні ресурси від обробки запитів на не налаштовані домени. Після успішної ідентифікації `Middleware` переходить до збору контекстних даних, виконуючи агрегацію всіх необхідних сутностей для конкретного вебресурсу. На цьому етапі завантажуються всі доступні тематичні розділи та категорії, формується список дат публікації статей для побудови навігації через архіви, а також визначається автор контенту за замовчуванням для підвищення рівня SEO-довіри до ресурсу.

Завершальним кроком є глобальна реєстрація отриманих даних, під час якої всі зібрані параметри реєструються в глобальному сховищі Config. Це стратегічне архітектурне рішення дозволяє будь-якому компоненту системи, включаючи контролери, Blade-шаблони або допоміжні сервіси, отримувати миттєвий доступ до всіх налаштувань сайту через фасад Config::get(). Такий підхід виключає необхідність виконання повторних запитів до бази даних у межах одного життєвого циклу запиту, що значно підвищує загальну продуктивність програмного комплексу. Нижче наведено лістинг класу CheckSite, який втілює описану логіку.

```
public function handle(Request $request, Closure $next)
{
    $site = (new \App\Models\Site)->getSiteByDomain($_SERVER['HTTP_HOST']);
    if (!empty($site) && $_SERVER['REQUEST_URI'] !== '/admin') {
        $categories = $site->allCategories;
        $categories = (new Site)->checkCategories($site, $categories);
        $dateArticles = (new Site)->getDateActualArticles($site);
        $defaultAuthor = $site->use_default_author != null ? (new
SiteAuthor)->getDefaultAuthorBySiteId($site->id) : null;
        $pagesLink = (new SitePage()->getAllPagesLinkBySiteId($site->id);
        Config::set('services.defaultAuthor', $defaultAuthor);
        Config::set('services.pagesLink', $pagesLink);
        Config::set('services.datesCategory', $dateArticles);
        Config::set('services.categories', $categories);
        Config::set('services.site', $site);
        return $next($request);
    } else {
        abort(404);}}}
```

Така гнучка система роутингу дозволяє централізовано вносити зміни в структуру всіх сайтів одночасно, зберігаючи при цьому унікальність URL-адрес та контенту для кожного окремого проєкту.

Для забезпечення візуальної різноманітності мережі сайтів було реалізовано систему шаблонізації, яка дозволяє кожному ресурсу мати унікальний зовнішній

вигляд. На поточному етапі розроблено два базові архітектурні шаблони, що відрізняються структурою розмітки та стилістичним оформленням. Така модульна побудова фронтенд-частини дозволяє в майбутньому безмежно розширювати бібліотеку тем, що є критично важливим для уникнення ідентифікації мережі пошуковими системами за ідентичними відбитками.

Практична реалізація виводу контенту на головну сторінку покладена на HomeController. Контролер отримує параметри конфігурації сайту, перелік категорій та масив згенерованих статей (включаючи медіафайли та баннери). На основі отриманого з налаштувань ідентифікатора теми система автоматично визначає шлях до відповідного Blade-шаблону. Це дозволяє використовувати єдину логіку підготовки даних для різних візуальних представлень, забезпечуючи повне відображення унікальної інформації про сайт - від назви та логотипа до специфічних SEO-налаштувань. Лістинг класу HomeController.

```
class HomeController
{
    public function index(Request $request)
    {
        $site = Config::get('services.site');
        $data = (new Site)->homeArticlesWithMedia($site);
        $template = $site->theme->template;
        $categories = Config::get('services.categories');
        $articles = $data['articles'];
        $articlesBanner = $data['firstThreeArticles'];
        $articles = (new ArticlesController)->checkDefaultAuthor($articles);
        return view('front.templates.' . $template . '.home', ['site' => $site, 'articles' => $articles,
        'articlesBanner' => $articlesBanner, 'categories' => $categories]);
    }
}
```

Такий підхід дозволяє адміністратору через CRM-панель миттєво змінювати зовнішній вигляд сайту, просто перемикаючи тему в налаштуваннях. При цьому вся структура посилань, контент та SEO-метадані залишаються стабільними, а

фронтенд-частина автоматично підлаштовується під обрану верстку, виводячи дані у відповідні блоки шаблону.

2.4. Розгортання проєкту та зв'язування LSTM моделі з CRM

Після завершення розробки окремих модулів системи, а саме нейронної мережі на базі Python та CRM-системи на Laravel, виникла необхідність їхньої інтеграції в єдине програмне середовище. Оскільки ці технології висувають різні вимоги до системних бібліотек та оточення, для забезпечення стабільності було обрано мікросервісний підхід із використанням технології контейнеризації. Для уніфікації процесу розгортання застосовано Docker, що дозволило повністю ізолювати Python-середовище з усіма специфічними залежностями, такими як PyTorch, Flask та NumPy, від PHP-оточення. Таке архітектурне рішення ефективно вирішує проблему конфлікту залежностей та дозволяє моделі використовувати ресурси сервера незалежно від основного вебдодатка. У конфігурації файлу `docker-compose.yml` нейромережевий модуль визначено як окремий сервіс `ai_service`, який взаємодіє з CRM через внутрішню віртуальну мережу Docker. Це додатково підвищує безпеку системи, оскільки API моделі залишається недоступним із зовнішньої мережі.

Окрім безпосередньо обчислювального модуля, було здійснено налаштування повного стека середовища, який включає кілька критично важливих компонентів. Роль централізованого сховища даних виконує база даних MySQL, де зберігаються всі налаштування мережі та результати генерації текстового контенту. Для візуального адміністрування та моніторингу стану бази даних на етапі розробки інтегровано інструмент phpMyAdmin. Ядром системи виступає CRM-панель на Laravel, яка виконує функції панелі керування всіма процесами. Окремі вебресурси розгорнуті у власних контейнеризованих середовищах і в реальному часі звертаються до CRM для отримання індивідуальних конфігурацій.

Архітектура дозволяє розгорнути всю інфраструктуру за допомогою однієї команди, що гарантує абсолютну ідентичність робочого середовища на будь-якому

сервері. Це забезпечує надійний та безперебійний зв'язок між LSTM-моделлю та CRM-системою, де остання виступає координаційним центром системи, контролюючи запити на генерацію та розподіляючи готовий контент між підключеними сайтами. На рисунку 2.5 продемонстровано поточний стан розгорнутої інфраструктури, що підтверджує успішний запуск усіх життєво важливих вузлів системи та їхню повну готовність до роботи.

Container Name	Status	ID	Image	Ports	Usage	Time	Actions
blog	Running	-	-	-	1.42%	5 hours ago	Stop, Restart, Delete
ml_test_servi	Running	054e49b01b62	blog-ml_ge	5001:5000	0.01%	6 hours ago	Stop, Restart, Delete
redis-1	Running	a5f608bf890c	redis:alpine	6379:6379	0.46%	6 hours ago	Stop, Restart, Delete
phpmyadmin	Running	2ce8b73463e2	phpmyadm	8081:80	0.01%	6 hours ago	Stop, Restart, Delete
nginx-1	Running	6aeb4392a9a7	blog-nginx	80:80	0%	6 hours ago	Stop, Restart, Delete
mysql-1	Running	1ac2642a9b70	mysql:8	3306:3306	0.94%	6 hours ago	Stop, Restart, Delete
backend-1	Running	098c3c33dd22	blog-backe	-	0%	5 hours ago	Stop, Restart, Delete

Рис. 2.5. Успішно запуснені контейнери в docker

Для того, щоб CRM-система могла надсилати команди на генерацію, нейромережевий модуль було обгорнуто у веб-сервер на базі фреймворку Flask. Він виконує роль шлюзу: приймає вхідні HTTP-запити з початковими фразами, запускає ітераційний процес генерації в моделі LSTM та повертає готовий результат у форматі JSON. Лістинг веб-серверу на python.

```
@app.route('/generate', methods=['POST'])
def api_generate():
    try:
        data = request.get_json()
        start_text = data.get('start_text', 'artificial intelligence')
        length = data.get('length', 250)
        temperature = data.get('temperature', 0.7)
        if length > 500: length = 500
        generated_text = generate_text(
            MODEL,
            WORD2IDX,
            IDX2WORD,
            start_text,
```

```

        length=length,
        temperature=temperature
    )
    return jsonify({
        'status': 'success',
        'generated_text': generated_text
    })
except Exception as e:
    return jsonify({
        'status': 'error',
        'message': f'Generation error: {e}'
    }), 500

```

На стороні Laravel за комунікацію відповідає сервіс `GenerateSingleArticleJob`. Використання бібліотеки `Guzzle` дозволяє організувати надійний обмін даними між контейнерами. Процес автоматизовано таким чином: фонове завдання (`Job`) бере заголовок із черги, надсилає його POST-запитом до Flask-сервісу, отримує відповідь та автоматично створює новий запис у таблиці `articles`, прив'язуючи його до відповідного сателіта. Лістинг коду відправки запиту на flask-сервіс.

```

$response = $client->post('http://ml_generator:5000/generate', [
    'json' => [
        'start_text' => $this->startText,
        'length' => $this->length,
        'temperature' => $this->temperature],
    'timeout' => 50,]);
$data = json_decode($response->getBody()->getContents(), true);
if ($data['status'] === 'success') {
    $generatedContent = $data['generated_text'];
    $status = 'generated';
} else {
    $errorMsg = 'ML Service Error: ' . ($data['message'] ?? 'Unknown error');
    throw new Exception($errorMsg);
}

```

Така архітектура зв'язування дозволяє системі працювати асинхронно: адміністратор лише натискає кнопку в CRM, а складна робота з обробки нейронною мережею відбувається у фоні. Це забезпечує безперебійне функціонування інтерфейсу та дозволяє масштабувати мереж, генеруючи тисячі унікальних статей без затримок у роботі основного сайту.

Висновки до другого розділу

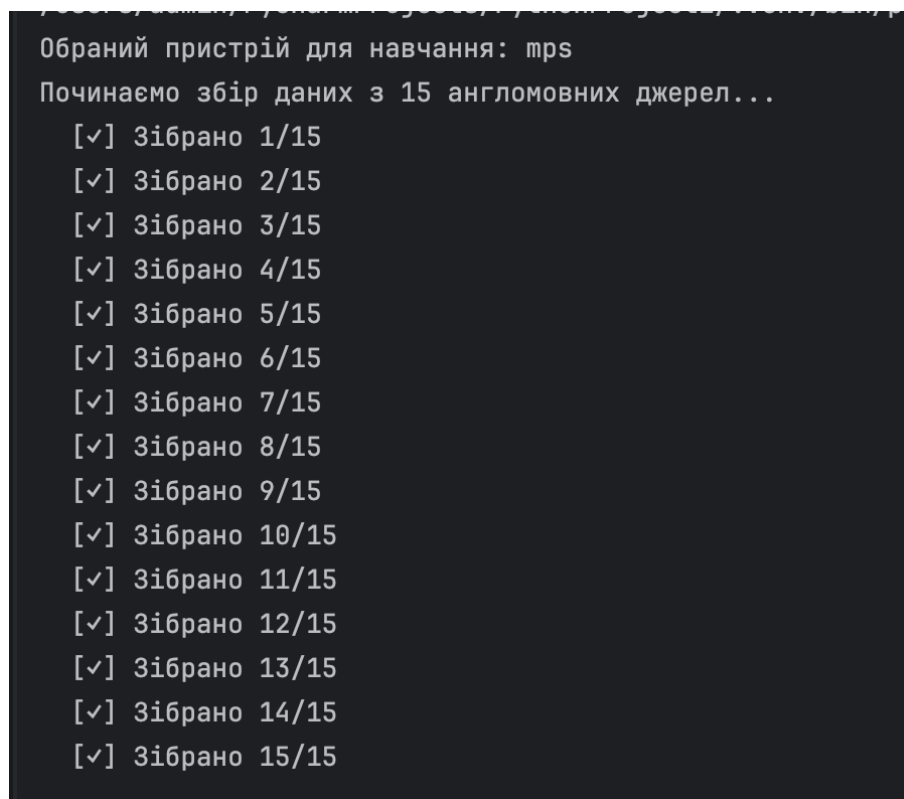
У ході розробки було впроваджено нейромережевий модуль на базі архітектури LSTM, який забезпечує автоматизоване створення тематичного контенту з високим рівнем унікальності та збереженням стилістичних особливостей оригінальних текстів. Система отримала здатність адаптуватися до різних мовних моделей, що є критично важливим для якісної SEO-оптимізації сайтів.

Централізована CRM-система, побудована на фреймворку Laravel, реалізує складну бізнес-логіку управління через патерн MVC та систему динамічної ініціалізації контексту. Використання Middleware дозволило відокремити ідентифікацію сайтів від основної логіки додатка, забезпечуючи роботу сотень незалежних доменів на єдиному програмному ядрі. Важливим інженерним досягненням стала реалізація асинхронної взаємодії через черги завдань, що дозволило перенести ресурсомісткі процеси генерації тексту у фоновий режим. Це гарантує стабільність адміністративного інтерфейсу та високу відмовостійкість системи навіть при одночасній обробці великої кількості запитів на створення контенту. Фінальний етап розгортання було здійснено за допомогою технології контейнеризації Docker, що дозволило об'єднати технологічні стеки PHP та Python у межах єдиної ізольованої інфраструктури. Налаштування віртуальної мережі між контейнерами забезпечило безпечну та швидку комунікацію між CRM та ШІ-модулем через внутрішні API-запити. Таким чином, розроблений програмний комплекс є цілісним та масштабованим рішенням, яке автоматизує повний цикл життєдіяльності мережі сайтів.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПРОГРАМНИМ КОМПЛЕКСОМ

3.1 Аналіз результатів навчання та генерації нейромережевого модуля

Процес тестування розпочався з етапу підготовки даних та формування лексичної бази моделі нейромережі. На першому етапі було проаналізовано результати токенизації та побудови словника. Система успішно обробила зібраний текстовий корпус, сформувавши унікальний набір tokenів. Скріншот консольного виводу підтверджує коректність індексування слів та формування структури word2idx, де кожне слово отримало свій числовий ідентифікатор для подальшої обробки нейронною мережею. Результат формування словника та індексації tokenів зображено на рис. 3.1.



```
Обраний пристрій для навчання: mps
Починаємо збір даних з 15 англomовних джерел...
[✓] Зібрано 1/15
[✓] Зібрано 2/15
[✓] Зібрано 3/15
[✓] Зібрано 4/15
[✓] Зібрано 5/15
[✓] Зібрано 6/15
[✓] Зібрано 7/15
[✓] Зібрано 8/15
[✓] Зібрано 9/15
[✓] Зібрано 10/15
[✓] Зібрано 11/15
[✓] Зібрано 12/15
[✓] Зібрано 13/15
[✓] Зібрано 14/15
[✓] Зібрано 15/15
```

Рис. 3.1. Результат формування словника та індексації tokenів

Важливим метричним показником на етапі підготовки став загальний розмір символів та слів у навчальній вибірці. Програма провела статистичний аналіз

корпусу, визначивши загальну довжину тексту та кількість унікальних символів. Ці дані є критичними для налаштування vocab_size в архітектурі LSTM, оскільки вони визначають розмірність вхідного та вихідного шарів нейронної мережі. Наявність достатнього обсягу даних (кількох сотень тисяч символів) є запорукою того, що модель зможе вивчити контекстуальні зв'язки. Результат збіру даних зображено на рис. 3.2.

```
✓ Збір даних завершено. Загальний розмір корпусу: 95,921 символів.
```

Рис. 3.2. Статистичні дані обробленого текстового корпусу

Наступним етапом стало безпосереднє навчання моделі. Тестування процесу навчання проводилося шляхом моніторингу зміни функції втрат (Loss) протягом встановленої кількості епох. На початкових етапах спостерігалось високе значення помилки, проте з кожною наступною епохою показник втрат стабільно знижувався. Це свідчить про те, що зменшення втрат ефективно коригує ваги моделі, і нейронна мережа поступово адаптується до структури природної мови. Результат навчання моделі зображено на рис. 3.3.

```
Кількість слів у словнику: 3934
Epoch 1, Loss: 5610.7533
Epoch 50, Loss: 235.0591
Epoch 100, Loss: 221.7843
Epoch 150, Loss: 221.3796
Epoch 200, Loss: 212.6609
Epoch 250, Loss: 213.5290
Epoch 300, Loss: 211.6792

✓ Навчання завершено. Зберігаємо модель та словник...
Файли model_trained_en.pth та word2idx_en.pkl успішно збережені.

Process finished with exit code 0
```

Рис. 3.3. Динаміка навчання моделі по епохах та показник Loss

Було проведено тестовий запуск генерації безпосередньо в інтегрованому середовищі розробки PyCharm. Цей метод дозволяє виключити вплив мережових протоколів та перевірити швидкість обробки даних самою моделлю LSTM на локальних обчислювальних потужностях. Під час тестування у терміналі було ініційовано виконання скрипту, який завантажує попередньо збережені ваги навченої моделі та словник токенів.

У ході експерименту моделі було задано вхідну фразу та параметри довжини й температури. Результат консольного виводу підтвердив, що модель коректно прогнозує наступні слова у реченні, вибудовуючи логічно завершений текст на основі вивчених раніше закономірностей. На скріншоті чітко видно процес ініціалізації моделі та отриманий у відповідь текстовий блок, що свідчить про успішне завершення етапу навчання та готовність алгоритму до інтеграції в загальну екосистему через API-шлюз. Тестовий вивід тексту на основі навченої моделі зображено на рис. 3.4.

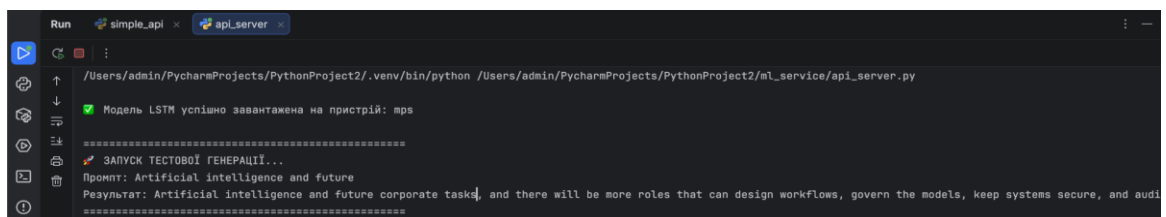
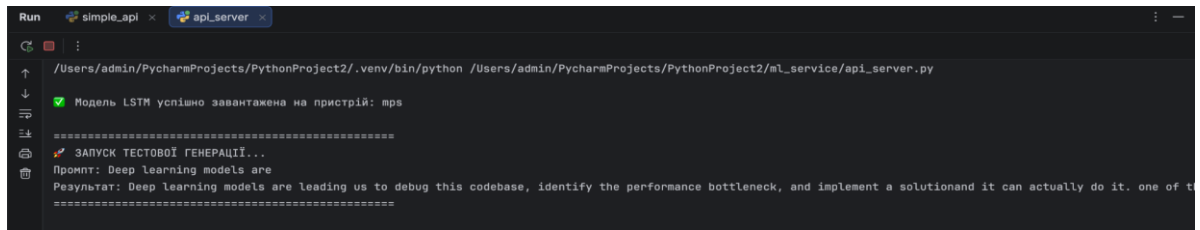


Рис. 3.4. Тестовий вивід згенерованого тексту на основі навченої моделі

Для підтвердження високої якості та варіативності контенту було проведено повторну генерацію тексту з використанням відмінної початкової фрази. Результати контрольних запусків продемонстрували здатність LSTM-моделі адаптувати лексичний склад та граматичні конструкції залежно від заданого контексту. Навіть при роботі з близькими за тематикою промптами, алгоритм формує унікальні послідовності токенів завдяки встановленому параметру температури генерації, що вносить елемент імовірнісного вибору та запобігає детермінованому повторенню фраз. Повторний тестовий вивід тексту на основі навченої моделі зображено на рис. 3.5.

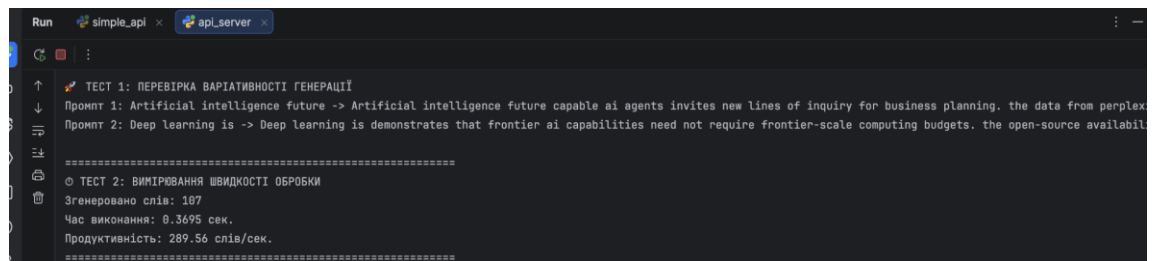


```
Run simple_api x api_server x
/Users/admin/PycharmProjects/PythonProject2/.venv/bin/python /Users/admin/PycharmProjects/PythonProject2/ml_service/api_server.py
✓ Модель LSTM успішно завантажена на пристрій: mps
=====
✚ ЗАПУСК ТЕСТОВОЇ ГЕНЕРАЦІЇ...
Промпт: Deep learning models are
Результат: Deep learning models are leading us to debug this codebase, identify the performance bottleneck, and implement a solutionand it can actually do it. one of t
=====
```

Рис. 3.5. Тестовий вивід згенерованого тексту на основі навченої моделі

Порівняльний аналіз отриманих результатів (Рис. 3.4 та Рис. 3.5) підтверджує, що система забезпечує високий рівень унікальності текстових блоків, що є критично важливою умовою для успішної індексації сайтів пошуковими системами. Кожен згенерований фрагмент має власну структуру та змістовне наповнення, зберігаючи при цьому задану технічну стилістику, що доводить ефективність проведеного навчання на зібраному корпусі даних.

Важливим критерієм оцінки ефективності розробленого нейромережевого модуля стала швидкість генерації та віддачі текстового контенту. Для об'єктивного аналізу продуктивності в програмний код було впроваджено інструменти точного вимірювання часу виконання операцій. Отримані результати (Рис. 3.6) продемонстрували високу продуктивність системи: генерація текстового блоку обсягом 107 слів зайняла лише 0.3695 секунди. Показник питомої швидкості на рівні 289.56 слів/сек підтверджує, що обрана архітектура LSTM у поєднанні з апаратним прискоренням Apple Silicon (MPS) здатна забезпечувати миттєву віддачу контенту. Така швидкодія дозволяє CRM-системі обробляти масові запити на створення статей без затримок, що є ключовим фактором при масштабуванні мережі сайтів. Тестування швидкості обробки статей зображено на рис. 3.5.

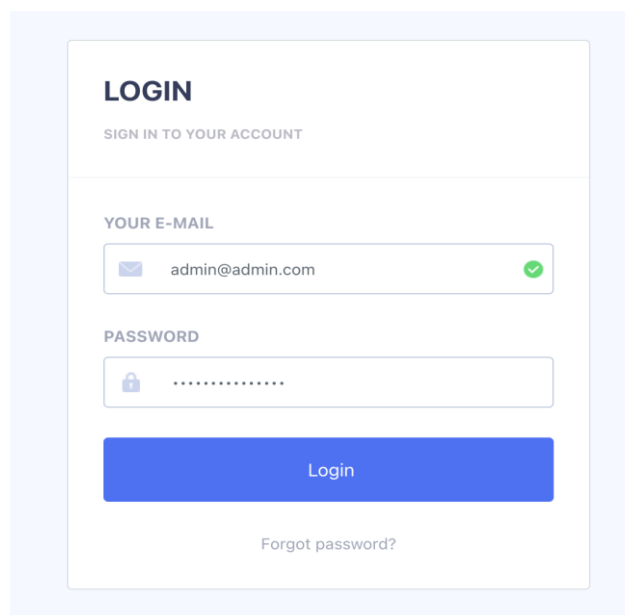


```
Run simple_api x api_server x
TEST 1: ПЕРЕВІРКА ВАРІАТИВНОСТІ ГЕНЕРАЦІЇ
Промпт 1: Artificial intelligence future -> Artificial intelligence future capable ai agents invites new lines of inquiry for business planning. the data from perplex
Промпт 2: Deep learning is -> Deep learning is demonstrates that frontier ai capabilities need not require frontier-scale computing budgets. the open-source availabil
=====
© TEST 2: ВИМІРЮВАННЯ ШВИДКОСТІ ОБРОБКИ
Згенеровано слів: 107
Час виконання: 0.3695 сек.
Продуктивність: 289.56 слів/сек.
=====
```

Рис. 3.6. Тестування швидкості обробки та генерації статей

3.2. Тестування інтерфейсу адміністратора та системи управління конфігураціями

Процес перевірки користувацької частини системи розпочався з етапу авторизації. Для гарантування безпеки даних та розмежування прав доступу було реалізовано захищений вхід у систему. На рівні бази даних та програмної логіки створено роль адміністратора, який володіє повними правами на перегляд, редагування та видалення будь-яких сутностей у мережі сайтів. Тестування вікна логіну підтвердило коректну роботу механізмів сесій та захисту від несанкціонованого доступу до панелі керування зображено на рис. 3.7. Після успішного входу адміністратор потрапляє до основного розділу керування сайтами. Сторінка списку конфігурацій сайтів була спроектована за стандартами сучасних систем для зручної роботи з великими масивами даних. Під час тестування було перевірено роботу ключових компонентів інтерфейсу: системи динамічного пошуку за доменом або назвою, механізму пагінації та блоку вибору кількості елементів на одній сторінці. Використання асинхронних запитів дозволяє оновлювати список без повного перезавантаження вікна, що значно прискорює адміністрування мережі, зображено на рис. 3.8.



The image shows a login form titled "LOGIN" with the subtitle "SIGN IN TO YOUR ACCOUNT". It contains two input fields: "YOUR E-MAIL" with the value "admin@admin.com" and a green checkmark icon, and "PASSWORD" with masked characters ".....". Below the fields is a blue "Login" button and a link "Forgot password?".

Рис. 3.7. Інтерфейс авторизації адміністратора в CRM-системі

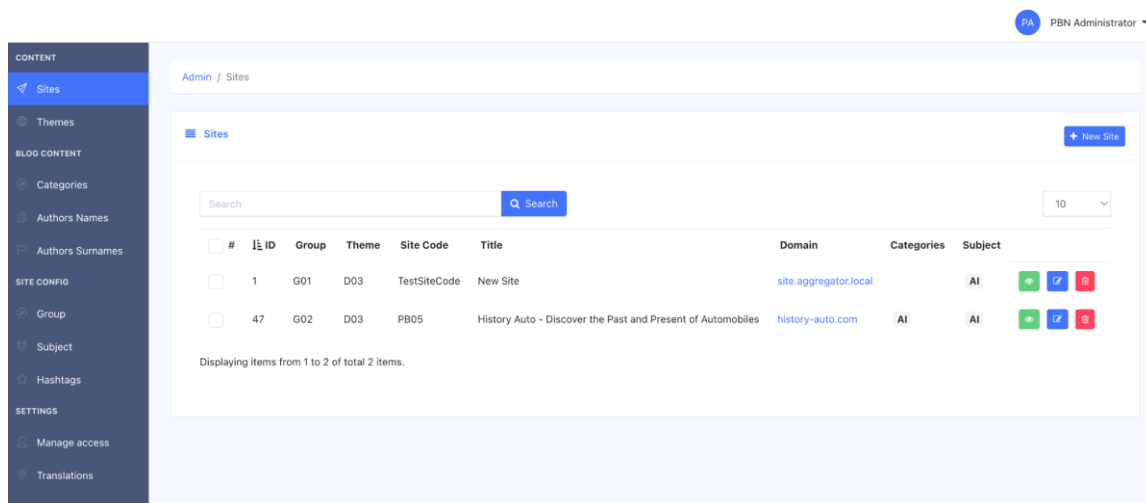


Рис. 3.8. Сторінка списку конфігурацій сайтів із елементами фільтрації та пагінації

Після перевірки лістингів було протестовано функціонал детального редагування конфігурації сайту. Цей розділ є критично важливим, оскільки саме тут формується індивідуальність кожного сателіта. В інтерфейсі редагування адміністратор має можливість гнучко налаштовувати технічні та візуальні параметри: вказувати доменне ім'я для коректної ідентифікації через Middleware, завантажувати унікальний логотип та обирати тематичні категорії, які будуть доступні на конкретному ресурсі. Інтерфейс редагування зображений на рис. 3.9.

Рис. 3.9. Інтерфейс редагування параметрів та метаданих сайту

Окрім глобальних параметрів сайту, було протестовано роботу модуля Categories SEO, який дозволяє здійснювати точкове налаштування метаданих для кожного тематичного розділу. В інтерфейсі (рис. 3.10) реалізовано механізм перевірки наявності специфічних заголовків та описів для конкретної категорії. Якщо адміністратор визначає унікальні параметри для розділу, система пріоритетно використовує їх при рендерингу сторінки. У випадку, коли поля залишаються порожніми, програмна логіка автоматично підставляє дефолтні значення, визначені в загальних налаштуваннях конфігу.

Такий підхід дозволяє швидко запускати нові сателіти з базовими налаштуваннями, одночасно надаючи можливість глибокої оптимізації найбільш важливих розділів без необхідності редагування коду шаблонів. Тестування підтвердило, що система коректно ідентифікує відсутність даних та забезпечує безперервність відображення метатегів на фронтенд-частині, що є критичним для стабільної індексації сайту пошуковими роботами.

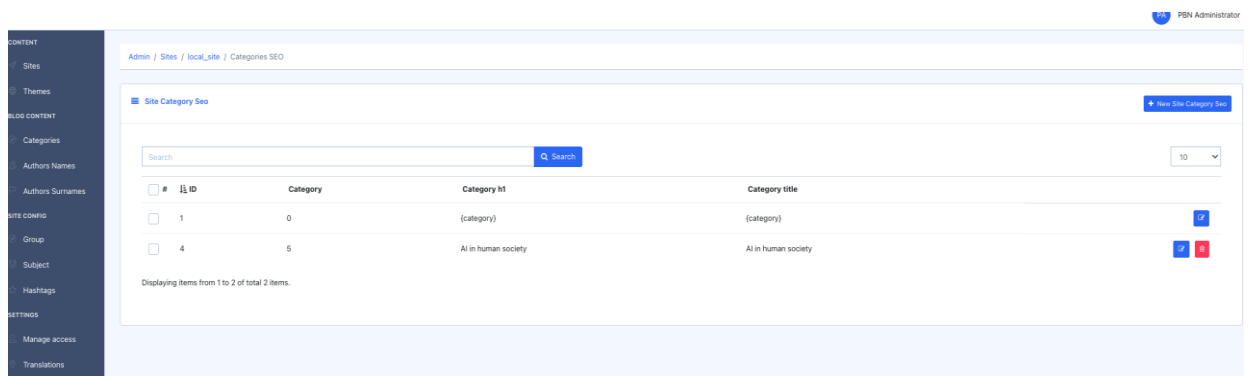


Рис. 3.10. Інтерфейс налаштування метатегів для категорій та логіка дефолтних значень

Окрему увагу під час тестування приділено взаємозв'язкам між сутностями. В інтерфейсі (рис. 3.11) реалізовано переходи до вкладених лістингів, які безпосередньо відносяться до конкретного сайту. Зокрема, було перевірено роботу розділів керування списку статей, що належать обраній конфігурації. Система коректно фільтрує записи, відображаючи лише тих авторів та статті, які

ідентифіковані за унікальним `site_id`, що підтверджує цілісність реляційних зв'язків на рівні бази даних.

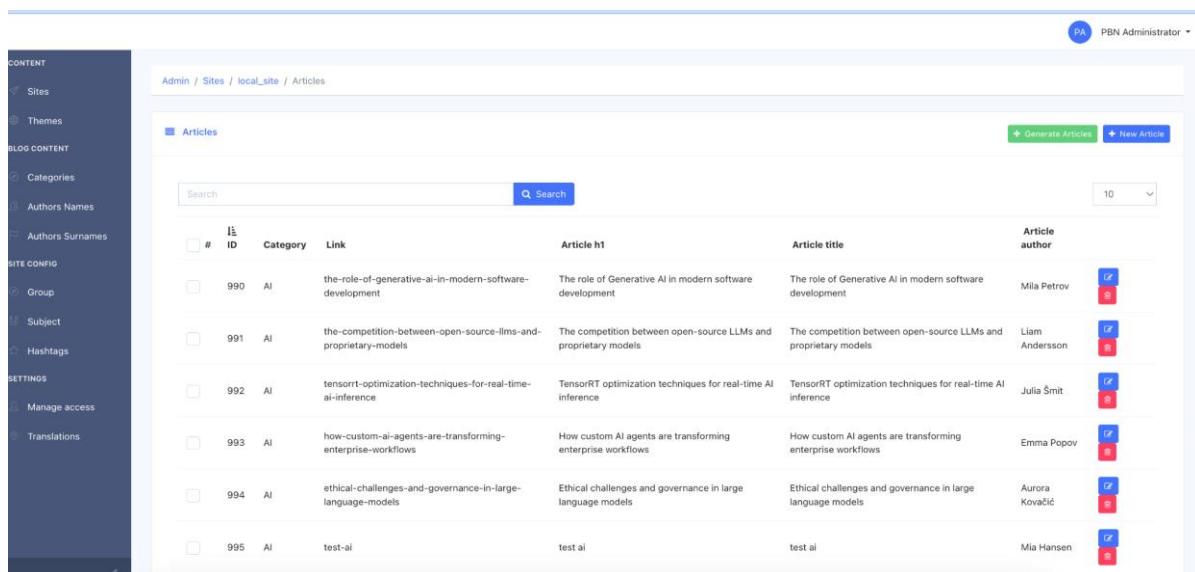


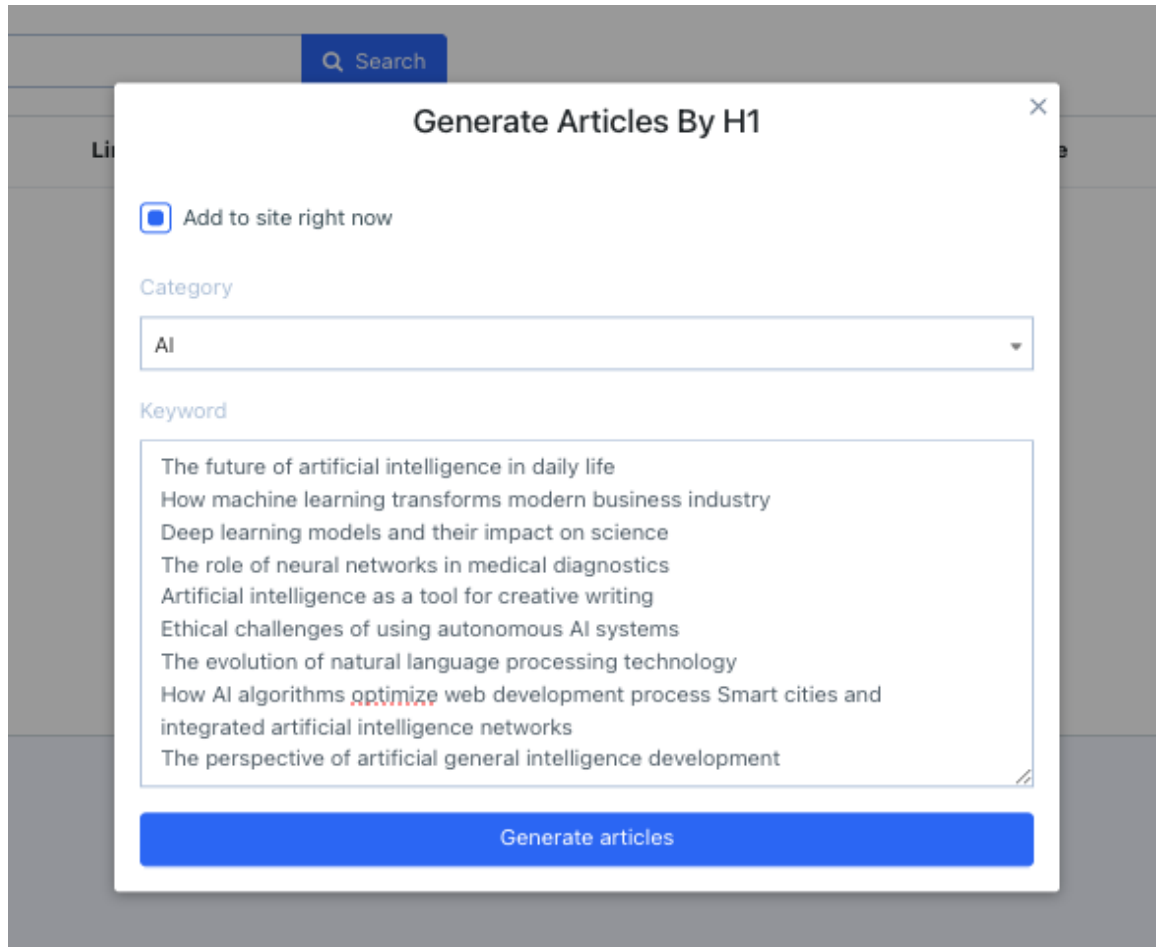
Рис. 3.11. Лістинг статей, прикріплених до обраного конфігу сайту

Така побудова інтерфейсу дозволяє адміністратору мати повний доступ до всієї мережі проєктів. Можливість швидко перемикатися між загальним списком сайтів та детальним переглядом їхнього вмісту (статей, категорій, авторів) робить систему масштабованою та придатною для керування сотнями ресурсів одночасно. Коректна робота всіх кнопок управління, швидкий пошук та стабільна пагінація доводять високу якість програмної реалізації фронтенд-частини CRM.

3.3. Тестування механізмів асинхронної взаємодії та черг генерації контенту між CRM та LSTM моделлю

Процес тестування створення статей під сайт, розпочався з модального вікна (рис. 3.12) в адміністративній панелі, яке слугує точкою входу для пакетного створення статей. Під час експерименту було продемонстровано введення масиву заголовків (H1) та параметрів тематичної прив'язки, після чого система ініціювала механізм розподіленої обробки даних. Замість прямого очікування відповіді від ШІ-сервісу, CRM миттєво додав кожен запит у відокремлений об'єкт завдання, що

дозволило зберегти повну інтерактивність користувацького інтерфейсу навіть під час інтенсивних обчислень.



The screenshot shows a modal window titled "Generate Articles By H1". It features a search bar at the top with a magnifying glass icon and the word "Search". Below the title bar, there is a checkbox labeled "Add to site right now" which is checked. Underneath the checkbox is a "Category" dropdown menu with "AI" selected. Below the category dropdown is a "Keyword" text area containing a list of article topics related to artificial intelligence. At the bottom of the modal is a blue button labeled "Generate articles".

Рис. 3.12. Інтерфейс модального вікна для пакетного введення заголовків статей

Після відправки форми (рис. 3.12), перевіряємо чи додалися задачі для створення статей в таблицю jobs. Таблиця jobs зображена на рис. 3.13.

Для переведення завдань зі стану очікування у стан виконання було задіяно воркер у середовищі Docker за допомогою команди `php artisan queue:work --queue=ai_generation,default --tries=3 --timeout=60`. Моніторинг терміналу в режимі реального часу дозволив відстежити життєвий цикл кожного завдання: від моменту захоплення воркером до фінального підтвердження успішної генерації та збереження тексту в реляційну таблицю статей. Встановлений параметр тайм-ауту забезпечив коректне завершення процесів у випадку тривалої генерації складних

текстів, а механізм повторних спроб (tries) мінімізував ризик втрати контенту через потенційні мережеві колізії між контейнерами. Процес виконання черг зображено на рис. 3.14.

	id	queue	payload	attempts	reserved_at	available_at	created_at
☐ Изменить ☐ Копировать ☐ Удалить	3	ai_generation	("uuid":"58f00ebd-02a3-4592-a29b-894f82b63d91","di...	0	NULL	1766008323	1766008323
☐ Изменить ☐ Копировать ☐ Удалить	4	ai_generation	("uuid":"b36710a1-4d3f-417f-a389-ad0579dd1859","di...	0	NULL	1766008323	1766008323
☐ Изменить ☐ Копировать ☐ Удалить	5	ai_generation	("uuid":"60655eb7-13f2-4ebe-8765-1d53f3f8d763","di...	0	NULL	1766008323	1766008323
☐ Изменить ☐ Копировать ☐ Удалить	6	ai_generation	("uuid":"ecd760e8-73d2-4271-9f0c-e9861429afa6","di...	0	NULL	1766008323	1766008323
☐ Изменить ☐ Копировать ☐ Удалить	7	ai_generation	("uuid":"c5f41b59-d4de-4912-a6ca-9da1a675c282","di...	0	NULL	1766008323	1766008323
☐ Изменить ☐ Копировать ☐ Удалить	8	ai_generation	("uuid":"02acf010-eb84-4352-b11e-5b3638490f87","di...	0	NULL	1766008323	1766008323
☐ Изменить ☐ Копировать ☐ Удалить	9	ai_generation	("uuid":"d9966512-245b-44da-81ad-00edd99076ca","di...	0	NULL	1766008323	1766008323
☐ Изменить ☐ Копировать ☐ Удалить	10	ai_generation	("uuid":"50987ed9-b9dc-466d-8b34-e12e17eca701","di...	0	NULL	1766008323	1766008323
☐ Изменить ☐ Копировать ☐ Удалить	11	ai_generation	("uuid":"0cac85e7-0409-44c4-bfa7-1c7b77ccb7a6","di...	0	NULL	1766008323	1766008323

Рис. 3.13. Стан таблиці jobs у базі даних після ініціації генерації

```

^[[A^[[B^Z[1]+  Stopped                  php artisan queue:work --queue=ai_generation,default --tries=3 --timeout=60
/app $ php artisan queue:work --queue=ai_generation,default --tries=3 --timeout=60

INFO Processing jobs from the [ai_generation,default] queues.
2025-12-17 21:53:36 App\Jobs\GenerateSingleArticleJob ..... RUNNING
2025-12-17 21:53:38 App\Jobs\GenerateSingleArticleJob ..... 1s DONE
2025-12-17 21:53:38 App\Jobs\GenerateSingleArticleJob ..... RUNNING
2025-12-17 21:53:39 App\Jobs\GenerateSingleArticleJob ..... 978.56ms DONE
2025-12-17 21:53:39 App\Jobs\GenerateSingleArticleJob ..... RUNNING
2025-12-17 21:53:40 App\Jobs\GenerateSingleArticleJob ..... 1s DONE
2025-12-17 21:53:40 App\Jobs\GenerateSingleArticleJob ..... RUNNING
2025-12-17 21:53:41 App\Jobs\GenerateSingleArticleJob ..... 1s DONE
2025-12-17 21:53:41 App\Jobs\GenerateSingleArticleJob ..... RUNNING
2025-12-17 21:53:42 App\Jobs\GenerateSingleArticleJob ..... 991.11ms DONE
2025-12-17 21:53:42 App\Jobs\GenerateSingleArticleJob ..... RUNNING
2025-12-17 21:53:43 App\Jobs\GenerateSingleArticleJob ..... 974.74ms DONE
2025-12-17 21:53:43 App\Jobs\GenerateSingleArticleJob ..... RUNNING
2025-12-17 21:53:44 App\Jobs\GenerateSingleArticleJob ..... 975.42ms DONE
2025-12-17 21:53:44 App\Jobs\GenerateSingleArticleJob ..... RUNNING
2025-12-17 21:53:45 App\Jobs\GenerateSingleArticleJob ..... 998.73ms DONE
2025-12-17 21:53:45 App\Jobs\GenerateSingleArticleJob ..... RUNNING
2025-12-17 21:53:46 App\Jobs\GenerateSingleArticleJob ..... 1s DONE
  
```

Рис. 3.14. Процес виконання черг у терміналі Docker-контейнера

Завершальним етапом тестування стала перевірка оновленого лістингу статей у CRM-панелі. Після відпрацювання всіх запланованих завдань у таблиці контенту автоматично з'явилися нові записи (рис. 3.15), що повністю відповідають заданим раніше заголовкам та структурі. Аналіз згенерованих текстів підтвердив успішну передачу даних від LSTM-моделі через API безпосередньо у базу даних CRM.

☐	ID	Category	Link	Article title	Article author
☐	997	AI	the-future-of-artificial-intelligence-in-daily-life	The future of artificial intelligence in daily life	Vincent Krüger
☐	998	AI	how-machine-learning-transforms-modern-business-industry	How machine learning transforms modern business industry	Clara Russo
☐	999	AI	deep-learning-models-and-their-impact-on-science	Deep learning models and their impact on science	Nina Lefebvre
☐	1000	AI	the-role-of-neural-networks-in-medical-diagnostics	The role of neural networks in medical diagnostics	Marie Monneau
☐	1001	AI	artificial-intelligence-as-a-tool-for-creative-writing	Artificial intelligence as a tool for creative writing	Emilia Klein
☐	1002	AI	ethical-challenges-of-using-autonomous-ai-systems	Ethical challenges of using autonomous AI systems	Emma Conti
☐	1003	AI	the-evolution-of-natural-language-processing-technology	The evolution of natural language processing technology	Aurora Krüger
☐	1004	AI	how-ai-algorithms-optimize-web-development-process-smart-cities-and-integrated-artificial-intelligence-networks	How AI algorithms optimize web development process Smart cities and integrated artificial intelligence networks	Oscar Russo

Рис. 3.15. Відображення результатів генерації у загальному списку статей

CRM

Таким чином, успішна перевірка асинхронної взаємодії довела надійність інтеграційної логіки та здатність системи до стабільної роботи під навантаженням.

3.4. Валідація динамічного відображення контенту та структурних елементів сателіта

Завершальним етапом тестування стала перевірка коректності виводу згенерованих даних та налаштованих параметрів на фронтенд-частині вебресурсу. Після успішного виконання черг генерації було здійснено перехід на головну сторінку сайту, де механізм Middleware CheckSite забезпечив автоматичну ідентифікацію домену та підвантаження відповідної конфігурації. На головній сторінці було зафіксовано коректне відображення статей. Основна секція сторінки заповнилася списком останніх статей, де кожна публікація містить заголовок H1, отриманий від нейромережі, короткий анонс та метадані, що підтверджує цілісність зв'язку між базою даних та фронтенд-контролерами. Відображення головної сторінки сайту зображено на рис. 3.16.

У ході подальшої перевірки було проаналізовано роботу сторінки лістингу категорій (рис. 3.17). Система продемонструвала здатність до динамічного формування контентної сітки на основі тематичної прив'язки статей. При переході

до конкретного розділу було зафіксовано вивід лише тих матеріалів, що належать до обраної категорії, а також відображення специфічних SEO-параметрів (Title та Description), налаштованих у модулі Categories SEO адмін-панелі. Це доводить ефективність розробленої логіки пріоритетності метаданих, де кастомні налаштування розділу успішно замінюють дефолтні значення сайту, забезпечуючи гнучкість пошукової оптимізації кожної сторінки окремо.

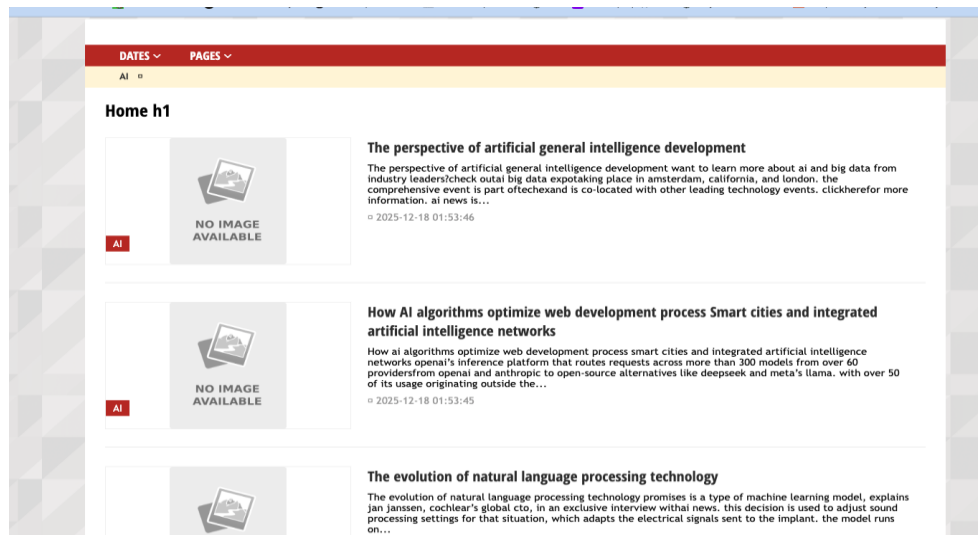


Рис. 3.16. Відображення головної сторінки сайту з динамічним контентом та брендуванням

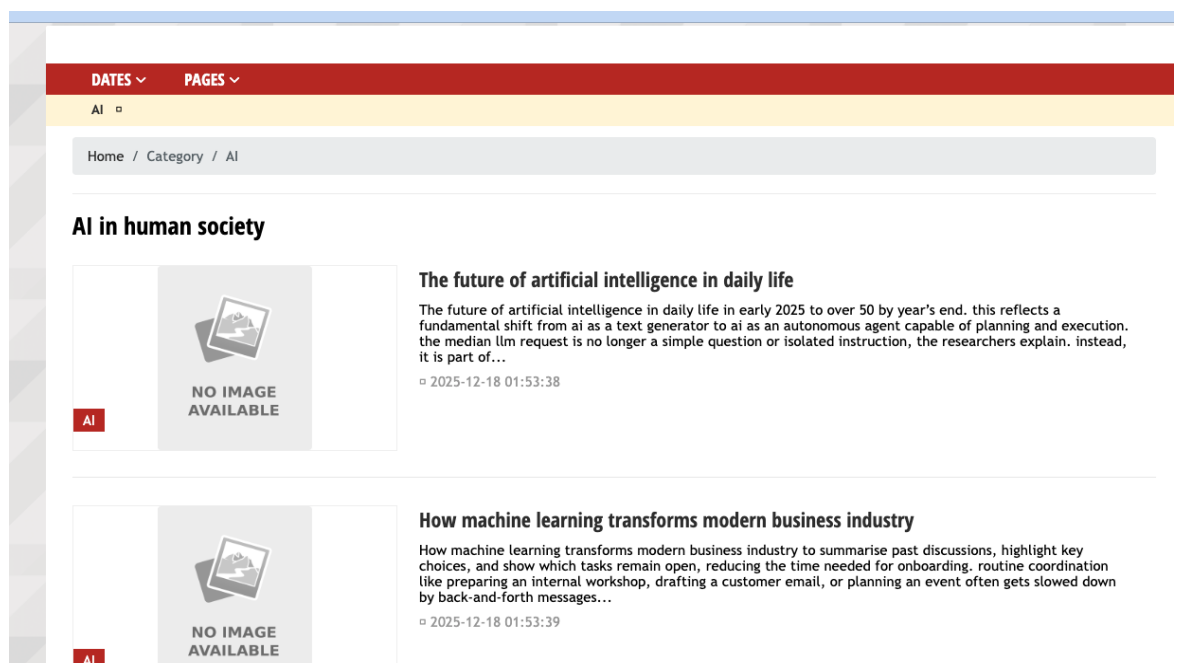


Рис. 3.17. Відображення сторінки лістингу статей по категорії

Протестуємо сторінку статі, яка є кінцевою точкою споживання контенту (рис. 3.18). Тестування показало, що повний текст, згенерований LSTM-моделлю, виводиться без втрати форматування та з дотриманням ієрархії заголовків. Сторінка містить усі необхідні атрибути: повне ім'я призначеного автора, дату публікації та тематичні теги. Коректний рендеринг сторінки статті підтверджує, що розроблений програмний комплекс успішно виконує повний цикл автоматизації - від інтелектуальної генерації тексту мовою оригіналу до його структурованого представлення кінцевому користувачу. Таким чином, результати валідації фронтенд-частини повністю підтверджують працездатність та готовність системи до експлуатації в реальних умовах.

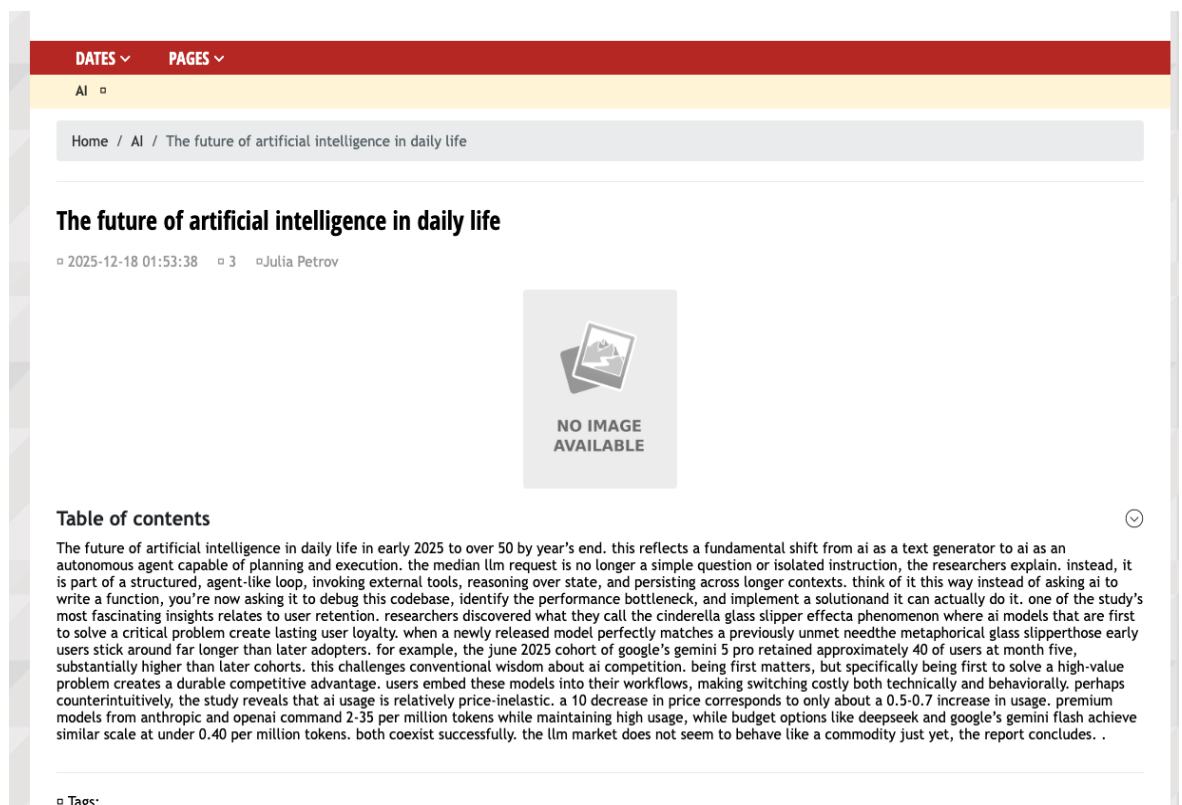


Рис. 3.18. Сторінка детального перегляду статті із виводом згенерованого тексту

Також змінємо тему сайту та протестуємо коректну роботу функціоналу. Порівняльний аналіз відображення головної сторінки та карток статей при зміні теми продемонстрував повну адаптивність системи. У новому візуальному

представленні всі раніше згенеровані статті та метатеги залишилися актуальними, але отримали нову стилістичну обробку, що відповідає обраному дизайну. Сайт з іншою обраною темою конфігурації зображено на рис. 3.19 та рис. 3.20.

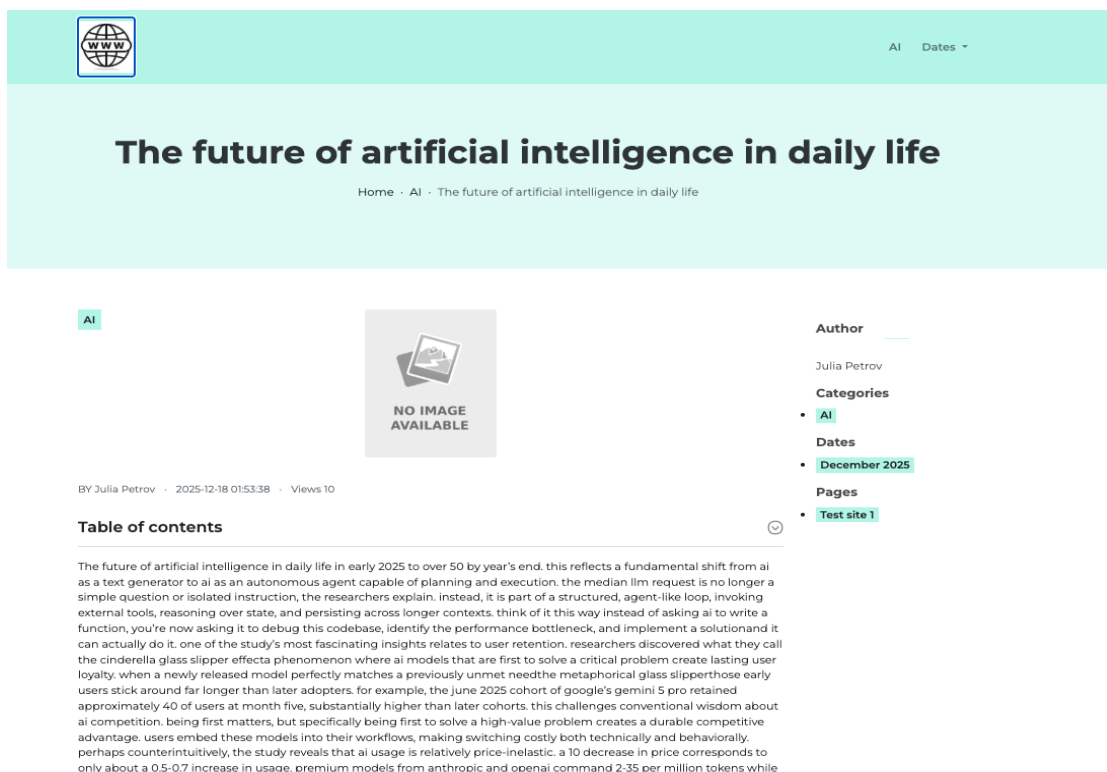


Рис. 3.19. Сторінка детального перегляду статті із виводом згенерованого тексту з іншою конфігурацією сайту

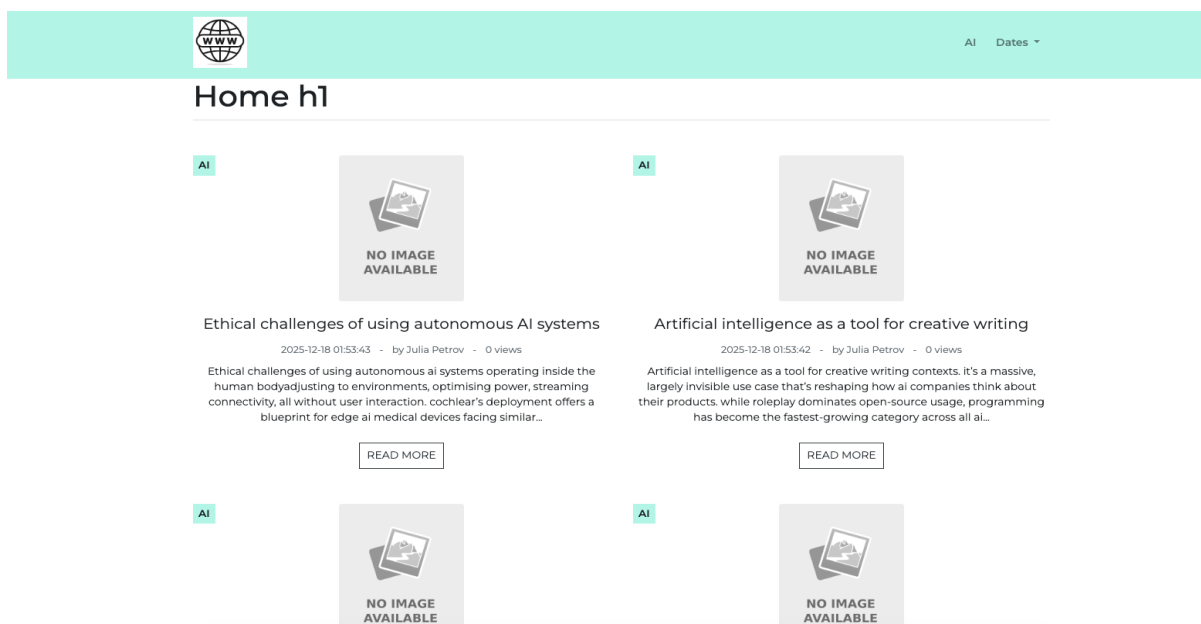


Рис. 3.20. Головна сторінка сайту

Такий функціонал є критично важливим для швидкого розгортання унікальних за виглядом мереж сайтів, дозволяючи адміністратору уникати візуального дублювання ресурсів без необхідності залучення розробників для створення нових сайтів.

Висновки до третього розділу

Результати проведеного тестування підтвердили повну працездатність обраної архітектури та ефективність інтеграції всіх компонентів системи. У ході верифікації нейромережевого модуля було встановлено, що розроблена модель LSTM демонструє високу якість генерації англomовного контенту, зберігаючи логічну структуру та контекстуальну унікальність текстів. Важливим досягненням стала висока швидкість обробки даних, яка завдяки апаратному прискоренню та оптимізації алгоритмів досягла показника понад 280 слів за секунду, що дозволяє використовувати систему в умовах інтенсивного навантаження.

Перевірка адміністративної панелі та механізмів управління сайтами довела надійність асинхронної моделі взаємодії. Використання черг завдань у поєднанні з реляційною базою даних забезпечило стабільність процесу створення статей, унеможливлюючи втрату даних навіть при масовій генерації контенту. Валідація фронтенд-частини продемонструвала гнучкість системи у відображенні індивідуальних налаштувань, включаючи брендування, SEO-метатеги та адаптивну зміну візуальних тем. Це підтверджує, що розроблений інструментарій дозволяє автоматизувати повний життєвий цикл управління мережею вебресурсів - від інтелектуального наповнення до фінальної презентації контенту користувачеві. Узагальнюючи результати тестування, можна стверджувати, що система відповідає всім поставленим технічним вимогам і готова до розгортання в реальних умовах експлуатації для вирішення завдань пошукової оптимізації та масового створення тематичних сателітів.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було досягнуто поставленої мети - розроблено та протестовано комплексну систему управління мережею сателітів з інтегрованим модулем автоматичної генерації контенту на основі рекурентних нейронних мереж. Дослідження поєднало в собі проектування складних мікросервісних архітектур, впровадження методів глибокого навчання та автоматизацію SEO-процесів. Аналіз предметної області підтвердив, що існуючі монолітні системи управління контентом не забезпечують необхідного рівня гнучкості та автономності при масштабуванні великих мереж ресурсів, що обґрунтувало доцільність розробки власного програмного продукту з гібридною структурою.

Розроблена архітектура на базі фреймворку Laravel та мови програмування Python продемонструвала високу ефективність у розподілі обчислювальних ресурсів. Використання моделі LSTM дозволило реалізувати стабільну генерацію тематичного англомовного контенту, який за показниками унікальності та логічної зв'язності відповідає вимогам сучасних пошукових систем. Практична реалізація підтвердила високу продуктивність системи: завдяки оптимізації алгоритмів та використанню апаратного прискорення швидкість генерації перевищила 280 слів за секунду, що є вагомим показником для нейромережових моделей рекурентного типу.

Особливу увагу в роботі приділено надійності обміну даними та асинхронності процесів. Впровадження механізму черг повідомлень та використання Docker-контейнеризації забезпечило стабільну роботу CRM-панелі незалежно від інтенсивності обчислювальних процесів у ML-модулі. Реалізована мультидоменна логіка дозволяє централізовано керувати великою кількістю незалежних сайтів, забезпечуючи для кожного з них унікальне брендування, індивідуальну ієрархію категорій та специфічні SEO-налаштування.

Наукова та практична цінність роботи полягає у створенні відкритої та масштабованої системи, яка мінімізує використання ручної праці при створенні та

підтримці мереж інформаційних ресурсів. Отримані результати можуть бути використані як база для подальших досліджень у сфері вдосконалення генеративних моделей, зокрема переходу до комерційного впровадження в галузі автоматизованого контент-маркетингу. Таким чином, успішне виконання всіх етапів розробки, тестування та валідації підтверджує досягнення мети дослідження та готовність системи до експлуатації в реальних умовах.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Климко І. М. Optimization of LSTM model. ELARTU – Інституційний репозитарій ТНТУ імені Івана Пулюя. 2024. URL: https://elartu.tntu.edu.ua/bitstream/lib/47938/2/MNPK_2024_Klymko_I_M-Optimization_of_LSTM_model_460-461.pdf (дата звернення: 18.12.2025).
2. Huyen C. Designing Machine Learning Systems: An Iterative Process for Production-Ready Applications. O'Reilly Media, 2022. 386 p.
3. Chollet F. Deep Learning with Python. 2nd ed. Manning Publications, 2021. 504 p.
4. Stauffer M. Laravel: Up & Running: A Framework for Building Modern PHP Apps. 3rd ed. O'Reilly Media, 2023. 560 p.
5. Allen A. Battle Ready Laravel: A guide to auditing, testing, fixing, and improving your Laravel applications. 2024. 280 p.
6. Winand M. SQL Performance Explained. 2012. 204 p.
7. Laravel - The PHP Framework For Web Artisans. URL: <https://laravel.com/> (дата звернення: 15.12.2025).
8. Flask documentation. Pallets Projects. URL: <https://flask.palletsprojects.com/> (дата звернення: 14.12.2025).
9. Craftable - Administration toolkit for Laravel. URL: <https://getcraftable.com/> (дата звернення: 15.12.2025).
10. Documentation for Craftable. Craftable - Laravel CRUD generator. URL: <https://getcraftable.com/docs/8.0/overview> (дата звернення: 12.12.2025).
11. Довідник по Machine Learning – Long Short-Term Memory (LSTM). База знань IT. URL: <https://itwiki.dev/data-science/ml-reference/ml-glossary/long-short-term-memory-lstm> (дата звернення: 11.06.2025).
12. Long Short Term Memory (LSTM) Networks using PyTorch. GeeksforGeeks. URL: <https://www.geeksforgeeks.org/deep-learning/long-short-term-memory-networks-using-pytorch/> (дата звернення: 11.12.2025).

13. Long Short-Term Memory (LSTM). NVIDIA Developer. URL: <https://developer.nvidia.com/discover/lstm> (дата звернення: 03.12.2025).
14. DeepLearning.AI. Sequence Models (LSTM) Overview. URL: <https://www.deeplearning.ai/> (дата звернення: 10.12.2025).
15. Queues - Laravel 5.8 documentation. URL: <https://laravel.com/docs/5.8/queues> (дата звернення: 01.12.2025).
16. Docker: accelerated container application development. URL: <https://www.docker.com/> (дата звернення: 02.12.2025).
17. PyCharm Documentation. JetBrains. URL: <https://www.jetbrains.com/help/pycharm/> (дата звернення: 05.12.2025).
18. MySQL Documentation. Oracle Corporation. URL: <https://dev.mysql.com/doc/> (дата звернення: 19.12.2025).
19. DbForge Studio for SQL Server - SQL Relationships. Devart Software. URL: <https://www.devart.com/dbforge/sql/studio/sql-relationships.html> (дата звернення: 18.12.2025).
20. Database Design and Entity-Relationship Diagrams (ERD). Visual Paradigm. URL: <https://www.visual-paradigm.com/guide/data-modeling/what-is-erd/> (дата звернення: 19.12.2025).
21. Scikit-learn: Machine Learning in Python. URL: <https://scikit-learn.org/stable/> (дата звернення: 19.12.2025).
22. Tutorials – Matplotlib 3.10.3 documentation. URL: <https://matplotlib.org/stable/tutorials/index> (дата звернення: 12.06.2025).
23. Gunicorn - Python WSGI HTTP Server for UNIX. URL: <https://gunicorn.org/> (дата звернення: 19.12.2025).
24. Nginx Documentation. URL: <https://nginx.org/en/docs/> (дата звернення: 19.12.2025).
25. Cloudflare. HTTP/2 Performance Benefits Explained. URL: <https://www.cloudflare.com/learning/performance/http2/> (дата звернення: 05.12.2025).

26. Medium Engineering. Machine Learning and SEO. URL: <https://medium.com/engineering> (дата звернення: 15.12.2025).
27. What Is GitHub? A Beginner's Introduction. Kinsta. URL: <https://kinsta.com/blog/what-is-github/> (дата звернення: 03.12.2025).
28. What is WordPress | CityHost. URL: <https://cityhost.ua/en/blog/wordpress-cto-eto-takoe-i-s-chem-ego-edyat.html> (дата звернення: 04.12.2025).
29. Joomla Content Management System (CMS). URL: <https://www.joomla.org/> (дата звернення: 18.12.2025).
30. Лабораторна робота №8. Learn ZTU. URL: https://learn.ztu.edu.ua/pluginfile.php/337170/mod_resource/content/1/Лабораторна%20робота%208.pdf (дата звернення: 18.12.2025).
31. Як штучний інтелект впливає на довкілля? Екодія. URL: <https://ecoaction.org.ua/iak-ai-vplyvaie-na-dovkillia.html> (дата звернення: 12.06.2025).

ДОДАТКИ

Код тренування моделі LSTM

```

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
import random, pickle, re
import numpy as np
import requests
from bs4 import BeautifulSoup
SEQ_LENGTH = 10
MODEL_PATH = "model_trained_en.pth"
VOCAB_PATH = "word2idx_en.pkl"
if torch.backends.mps.is_available():
    DEVICE = torch.device("mps")
elif torch.cuda.is_available():
    DEVICE = torch.device("cuda")
else:
    DEVICE = torch.device("cpu")
def download_article(url):
    headers = {"User-Agent": "Mozilla/5.0"}
    try:
        r = requests.get(url, headers=headers, timeout=10)
        r.raise_for_status()
        soup = BeautifulSoup(r.content, "html.parser")
        paragraphs = soup.find_all("p")
        text = "\n".join(p.get_text(strip=True) for p in paragraphs if len(p.get_text(strip=True)) > 50)
        if not text:
            text = soup.get_text(separator=' ', strip=True)
            text = text[text.find('article'):text.find('related posts')] # Limits
            text = text[:8000]
        text = re.sub(r'\s+', ' ', text).strip()
        return "Article Title", text
    except requests.exceptions.RequestException as e:

```

```

        return "", ""
def clean_text(text):
    text = text.lower()
    text = re.sub(r"[^a-z0-9\s,!?''-]", "", text, flags=re.UNICODE)
    return text
def remove_repeats(words, max_repeats=2):
    result = []
    count = 1
    for i in range(len(words)):
        if i == 0:
            result.append(words[i])
            continue
        if words[i] == words[i - 1]:
            count += 1
        else:
            count = 1
        if count <= max_repeats:
            result.append(words[i])
    return result
class TextDataset(Dataset):
    def __init__(self, data, seq_len):
        self.data = data
        self.seq_len = seq_len
    def __len__(self):
        return len(self.data) - self.seq_len
    def __getitem__(self, idx):
        return (
            torch.tensor(self.data[idx:idx + self.seq_len]),
            torch.tensor(self.data[idx + 1:idx + self.seq_len + 1])
        )
class LSTMMModel(nn.Module):
    def __init__(self, vocab_size, embed_size=128, hidden_size=256, num_layers=3):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_size)

```

```

self.lstm = nn.LSTM(embed_size, hidden_size, num_layers, batch_first=True)
self.fc = nn.Linear(hidden_size, vocab_size)

def forward(self, x, hidden=None):
    x = self.embedding(x)
    output, hidden = self.lstm(x, hidden)
    out = self.fc(output)
    return out, hidden

def generate_text(model, word2idx, idx2word, start_text, length=250, seq_length=SEQ_LENGTH,
temperature=0.8):
    model.eval()
    words_seq = clean_text(start_text).split()
    state = None
    device = next(model.parameters()).device
    for _ in range(length):
        indices = [word2idx.get(w, 0) for w in words_seq[-seq_length:]]
        if len(indices) < seq_length:
            indices = [0] * (seq_length - len(indices)) + indices
        x = torch.tensor([indices], dtype=torch.long).to(device)
        with torch.no_grad():
            out, state = model(x, state)
        last_word_logits = out[0, -1]
        p = torch.nn.functional.softmax(last_word_logits / temperature, dim=0).detach().cpu().numpy()
        if len(p) > 0:
            next_idx = np.random.choice(len(p), p=p)
            next_word = idx2word.get(next_idx, "word")
            words_seq.append(next_word)
        else:
            break
    if words_seq:
        words_seq[0] = words_seq[0].capitalize()
    words_seq = remove_repeats(words_seq)
    for i in range(1, len(words_seq)):
        if words_seq[i - 1] in ['.', '!', '?']:

```

```

        words_seq[i] = words_seq[i].capitalize()
    if words_seq and words_seq[-1] not in ['.', '!', '?']:
        words_seq.append('.')

    return ''.join(words_seq)

def collect_data_from_urls(file_path):
    all_text = []
    try:
        with open(file_path, 'r', encoding='utf-8') as f:
            urls = [line.strip() for line in f if line.strip() and not line.strip().startswith('#')]
    except FileNotFoundError:
        print(f' Error: URL list file not found: {file_path}')
        return ""

    print(f'Starting data collection from {len(urls)} English-language sources.')
    for i, url in enumerate(urls):
        _, text = download_article(url)
        if text and len(text) > 500:
            all_text.append(text)
            print(f' [✓] Collected {i + 1}/{len(urls)}')
        else:
            print(f' [X] Skipped {i + 1}/{len(urls)} (insufficient text content)')

    return "\n\n".join(all_text)

if __name__ == "__main__":
    print(f'Selected device for training: {DEVICE}')
    # --- 1. DATA COLLECTION FROM ALL URLS ---
    URL_FILE = "article_urls.txt"
    initial_text = collect_data_from_urls(URL_FILE)
    if not initial_text:
        print(" Fatal Error: No articles collected. Terminating process.")
        exit()

    print(f'\n Data collection complete. Total corpus size: {len(initial_text):,} characters.')
    text = clean_text(initial_text)
    words = text.split()

```

```

vocab = sorted(set(words))
word2idx = {w: i for i, w in enumerate(vocab)}
idx2word = {i: w for w, i in word2idx.items()}
vocab_size = len(vocab)
print("Vocabulary size:", vocab_size)
seq_length = SEQ_LENGTH
data = [word2idx[w] for w in words]
dataset = TextDataset(data, seq_length)
loader = DataLoader(dataset, batch_size=16, shuffle=True)
# --- TRAINING ---
model = LSTMModel(vocab_size).to(DEVICE)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
print("\nStarting model training...")
epochs = 300
for epoch in range(1, epochs + 1):
    model.train()
    total_loss = 0
    for x, y in loader:
        x, y = x.to(DEVICE), y.to(DEVICE)
        optimizer.zero_grad()
        out, _ = model(x)
        loss = criterion(out.view(-1, vocab_size), y.view(-1))
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    if epoch % 50 == 0 or epoch == 1:
        print(f'Epoch {epoch}/{epochs}, Loss: {total_loss:.4f}')
# --- SAVING RESULTS ---
print("\n Training complete. Saving model and vocabulary...")

torch.save(model.state_dict(), MODEL_PATH)
with open(VOCAB_PATH, "wb") as f:

```

```
pickle.dump(word2idx, f)
print(f'Files '{MODEL_PATH}' and '{VOCAB_PATH}' saved successfully.")
```

Код генерації статей на основі навченої моделі з використанням арі

```
import torch
import torch.nn as nn
import numpy as np
import pickle
import re
from flask import Flask, request, jsonify
import time
import sys
```

```
SEQ_LENGTH = 10
```

```
if torch.backends.mps.is_available():
    DEVICE = torch.device("mps")
elif torch.cuda.is_available():
    DEVICE = torch.device("cuda")
else:
    DEVICE = torch.device("cpu")
```

```
MODEL_PATH = "model_trained_en.pth"
VOCAB_PATH = "word2idx_en.pkl"
```

```
class LSTMModel(nn.Module):
    def __init__(self, vocab_size, embed_size=128, hidden_size=256, num_layers=3):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.lstm = nn.LSTM(embed_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, vocab_size)

    def forward(self, x, hidden=None):
        x = self.embedding(x)
```

```

output, hidden = self.lstm(x, hidden)
out = self.fc(output)
return out, hidden

```

```

def clean_text(text):
    text = text.lower()
    text = re.sub(r"^[a-z0-9\s,!?''-]", "", text, flags=re.UNICODE)
    return text

```

```

def remove_repeats(words, max_repeats=2):
    result = []
    count = 1
    for i in range(len(words)):
        if i == 0:
            result.append(words[i])
            continue
        if words[i] == words[i - 1]:
            count += 1
        else:
            count = 1
        if count <= max_repeats:
            result.append(words[i])
    return result

```

```

def generate_text(model, word2idx, idx2word, start_text, length=250, seq_length=SEQ_LENGTH,
temperature=0.7):
    model.eval()
    words_seq = clean_text(start_text).split()
    state = None
    device = next(model.parameters()).device

```

```

for _ in range(length):
    indices = [word2idx.get(w, 0) for w in words_seq[-seq_length:]]
    if len(indices) < seq_length:
        indices = [0] * (seq_length - len(indices)) + indices

    x = torch.tensor([indices], dtype=torch.long).to(device)

    with torch.no_grad():
        out, state = model(x, state)

    last_word_logits = out[0, -1]
    p = torch.nn.functional.softmax(last_word_logits / temperature, dim=0).detach().cpu().numpy()

    if len(p) > 0:
        next_idx = np.random.choice(len(p), p=p)
        next_word = idx2word.get(next_idx, "word")
        words_seq.append(next_word)
    else:
        break

if words_seq:
    words_seq[0] = words_seq[0].capitalize()

words_seq = remove_repeats(words_seq)

for i in range(1, len(words_seq)):
    if words_seq[i - 1] in ['.', '!', '?']:
        words_seq[i] = words_seq[i].capitalize()

if words_seq and words_seq[-1] not in ['.', '!', '?']:
    words_seq.append('.')

return ''.join(words_seq)

```

```
# --- INITIALIZATION AND TESTING ---
```

```
app = Flask(__name__)
```

```
try:
```

```
    with open(VOCAB_PATH, "rb") as f:
```

```
        WORD2IDX = pickle.load(f)
```

```
    IDX2WORD = {i: w for w, i in WORD2IDX.items()}
```

```
    VOCAB_SIZE = len(WORD2IDX)
```

```
    MODEL = LSTMMModel(VOCAB_SIZE).to(DEVICE)
```

```
    MODEL.load_state_dict(torch.load(MODEL_PATH, map_location=DEVICE))
```

```
    MODEL.eval()
```

```
    print(f"\n Model successfully loaded on: {DEVICE}", flush=True)
```

```
    print("\n" + "=" * 60, flush=True)
```

```
    print(" TEST 1: CHECKING GENERATION VARIABILITY", flush=True)
```

```
    prompts = ["Artificial intelligence future", "Deep learning is"]
```

```
    for i, p_text in enumerate(prompts, 1):
```

```
        res = generate_text(MODEL, WORD2IDX, IDX2WORD, p_text, length=30)
```

```
        print(f'Prompt {i}: {p_text} -> {res}', flush=True)
```

```
    print("\n" + "=" * 60, flush=True)
```

```
    print("🕒 TEST 2: MEASURING PROCESSING SPEED", flush=True)
```

```
    speed_prompt = "The future of technology depends on"
```

```
    t_start = time.time()
```

```
    speed_result = generate_text(MODEL, WORD2IDX, IDX2WORD, speed_prompt, length=100)
```

```
    t_end = time.time()
```

```

duration = t_end - t_start
w_count = len(speed_result.split())

print(f'Generated words: {w_count}', flush=True)
print(f'Execution time: {duration:.4f} sec.', flush=True)
print(f'Performance: {w_count / duration:.2f} words/sec.', flush=True)
print("=" * 60 + "\n", flush=True)

time.sleep(1)

except Exception as e:
    print(f' ERROR: {e}', flush=True)
    sys.exit(1)

@app.route('/generate', methods=['POST'])
def api_generate():
    try:
        data = request.get_json()
        start_text = data.get('start_text', 'artificial intelligence')
        length = data.get('length', 250)
        temp = data.get('temperature', 0.7)

        generated_text = generate_text(MODEL, WORD2IDX, IDX2WORD, start_text, length=length,
temperature=temp)

        return jsonify({
            'status': 'success',
            'generated_text': generated_text
        })
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500

```

```
if __name__ == '__main__':  
    print(f' API server is ready on port 5001", flush=True)  
    app.run(host='0.0.0.0', port=5001, debug=False)
```