НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет/(ННІ) інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету (Директор ННІ)	**Завідувач кафедри**
інформаційних технологій	комп'ютерних наук
(назва факультету (ННІ))	(назва кафедри)
_____ Ігор Болбот	_____ Белла Голуб
(підпис) (ім'я ПРІЗВИЩЕ)	(підпис) (ім'я ПРІЗВИЩЕ)
"___" _____ 20_ р.	"___" _____ 20_ р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему **Програмне забезпечення автоматичного оновлення контенту на веб-порталах за допомогою нейронних мереж**

Спеціальність 121 "Інженерія програмного забезпечення"
(код і найменування)

Освітня програма Програмне забезпечення інформаційних систем
(назва)

Орієнтація освітньої програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми
к.фіз.-мат.н., доцент _____ Віктор КИРИЧЕНКО
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Керівник магістерської кваліфікаційної роботи
д.т.н., проф. _____ Віктор СЕМКО
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Виконав _____ Ярослав БОНДАР
(підпис) (ім'я ПРІЗВИЩЕ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) __інформаційних технологій_____

ЗАТВЕРДЖУЮ
Завідувач кафедри__комп'ютерних наук_
___доцент, к.т.н._____ _____ _Белла Голуб
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"01" листопада 2024 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

_____Бондару Ярославу Костянтиновичу._____
(прізвище, ім'я, по батькові)

Спеціальність 121 «Інженерія програмного забезпечення»
(код і назва)

Освітня програма ____Програмне забезпечення інформаційних систем
(назва)

Орієнтація освітньої програми_____освітньо-професійна _____

Тема магістерської кваліфікаційної роботи Програмне забезпечення автоматичного оновлення контенту на веб-порталах за допомогою нейронних мереж
затверджена наказом ректора НУБіП України від "_1_" листопада__2024р. №1963 «С»
Термін подання завершеної роботи на кафедру 11 листопада 2025_____
(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи: Нормативно-довідкова та наукова література з інформаційних систем, баз даних, методів візуалізації та обробки даних; технічна документація з проєктування та розробки програмних комплексів; програмні засоби для розробки веб-застосунків і інтерактивних інтерфейсів; вимоги стандартів до побудови інформаційних систем і користувацьких інтерфейсів.

Перелік питань, що підлягають дослідженню:

1. Системний аналіз предметної області
2. Моделювання та архітектурне проєктування системи
3. Реалізація програмного забезпечення та технологічна інфраструктура системи
4. Тестування та оцінювання ефективності системи

Перелік графічного матеріалу (за потреби) презентація, постер, схеми та діаграми архітектури системи

Дата видачі завдання "__1__" листопада 2024 р.

Керівник магістерської кваліфікаційної роботи _____ ___Віктор СЕМКО
(підпис) (ім'я ПРІЗВИЩЕ))

Завдання прийняв до виконання _____ Ярослав БОНДАР____
(підпис) (ім'я ПРІЗВИЩЕ студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

1. API — Application Programming Interface, програмний інтерфейс взаємодії
2. CDN — Content Delivery Network, мережа доставки контенту
3. CMS — Content Management System, система керування контентом
4. CRON — формат планування періодичних задач у системах Unix
5. CPU — Central Processing Unit, центральний процесор
6. DB — Database, база даних
7. ETL — Extract, Transform, Load, процес вилучення, трансформації та завантаження даних
8. F1-score — гармонійне середнє точності та повноти моделі
9. gRPC — протокол високопродуктивної взаємодії між сервісами
10. HTTP/HTTPS — протоколи передачі гіпертексту / захищений протокол
11. JSON — JavaScript Object Notation, формат обміну даними
12. ML — Machine Learning, машинне навчання
13. MLflow — система керування та версіонування ML-моделей
14. MQ, Message Queue — черга повідомлень
15. NLP — Natural Language Processing, обробка природної мови
16. Nginx — веб-сервер та зворотний проксі-сервер
17. OAuth2/OIDC — протоколи автентифікації та авторизації
18. RAM — Random Access Memory, оперативна пам’ять
19. REST — Representational State Transfer, архітектурний стиль веб-сервісів
20. RL — Reinforcement Learning, навчання з підкріпленням
21. SDK — Software Development Kit, набір інструментів для розробки
22. SSE — Sum of Squared Errors, сума квадратів похибок (у кластеризації)

23. SSL/TLS — протоколи шифрування та захисту з’єднання
24. UI — User Interface, користувацький інтерфейс
25. URL — Unified Resource Locator, адреса ресурсу
26. XML — Extensible Markup Language, розширювана мова розмітки

ВСТУП

Швидкий розвиток інформаційних технологій і зростання обсягів даних, що генеруються у веб-середовищі, вимагають створення інтелектуальних програмних систем, здатних до автономної обробки, аналізу й оновлення контенту без безпосередньої участі людини. Сучасні веб-портали, які функціонують як новинні агрегатори, освітні платформи або корпоративні інформаційні системи, потребують динамічного оновлення даних у режимі реального часу для забезпечення актуальності, персоналізації та релевантності інформаційних потоків. Класичні алгоритми парсингу або оновлення контенту мають обмеження щодо адаптивності, контекстної інтерпретації й узагальнення даних, тому інтеграція штучних нейронних мереж у цей процес є логічним етапом еволюції автоматизованих систем управління веб-контентом [1]. Актуальність теми зумовлена необхідністю розроблення програмного забезпечення, яке поєднує алгоритмічну стійкість, здатність до навчання та аналітичну автономність при мінімізації людського втручання в процес оновлення інформаційних ресурсів.

Метою дослідження є створення програмного застосунку для автоматичного оновлення контенту веб-порталів із використанням нейронних мереж, здатного аналізувати структуру даних, визначати релевантність інформації та здійснювати її оновлення відповідно до визначених критеріїв якості та змістової повноти.

Для досягнення мети поставлено такі **завдання:**

1. проаналізувати предметну область автоматизованого управління контентом веб-порталів, визначивши ключові принципи, функції та обмеження існуючих систем.
2. Дослідити архітектурні, алгоритмічні та нейромережеві підходи до оновлення веб-даних, порівнявши їх ефективність і придатність до інтеграції у сучасні програмні рішення.

3. Обґрунтувати вибір оптимальної архітектури нейронної мережі для реалізації інтелектуального модуля автоматичного оновлення контенту, з урахуванням типу даних, контексту і цільових метрик якості.

4. Розробити логічну модель даних і функціональну архітектуру програмного застосунку, що забезпечує взаємодію між модулями аналізу, оновлення й управління контентом.

5. Реалізувати експериментальний прототип системи, інтегруючи навчальну модель у робоче середовище програмного забезпечення.

6. Провести оцінювання ефективності створеного застосунку на реальних наборах даних, визначивши точність, швидкодію та рівень автоматизації процесу оновлення контенту.

Об’єктом дослідження є процес автоматизованого оновлення контенту веб-порталів.

Предметом дослідження є методи й алгоритми використання нейронних мереж у системах інтелектуального керування веб-контентом.

Методологічну основу становлять методи штучного інтелекту, зокрема глибинного навчання, обробки природної мови (NLP), аналізу часових рядів, а також методи системного аналізу, UML-моделювання, проектування програмних архітектур і тестування програмних компонентів [2].

Наукова новизна полягає у розробленні комбінованого підходу до оновлення контенту, який поєднує евристичні методи обробки даних із навчанням нейронної моделі на історичних прикладах змін контенту, що дає змогу прогнозувати релевантність нових матеріалів і формувати контентну стрічку без участі адміністратора.

Практична цінність результатів полягає у можливості інтеграції розробленого застосунку в існуючі CMS-платформи або корпоративні портали, що дозволяє зменшити витрати на ручну модерацію та підвищити якість інформаційного обслуговування користувачів.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Сучасні веб-портали виступають складними інформаційними екосистемами, у межах яких відбувається безперервна взаємодія між джерелами даних, модулями збору, аналізу, оновлення та відображення контенту. Основною проблемою є забезпечення актуальності, достовірності й релевантності публікацій за умов великої частоти оновлення інформації, розподіленості джерел і різнорідності форматів даних. Автоматизація цього процесу вимагає побудови інтегрованої архітектури, де ключовими компонентами є модулі збору даних, обробки природної мови, політик оновлення, зберігання та візуалізації. На рис. 1.1 подано структурну схему предметної області, яка відображає основні елементи системи автоматичного управління контентом веб-порталів та взаємозв’язки між ними.

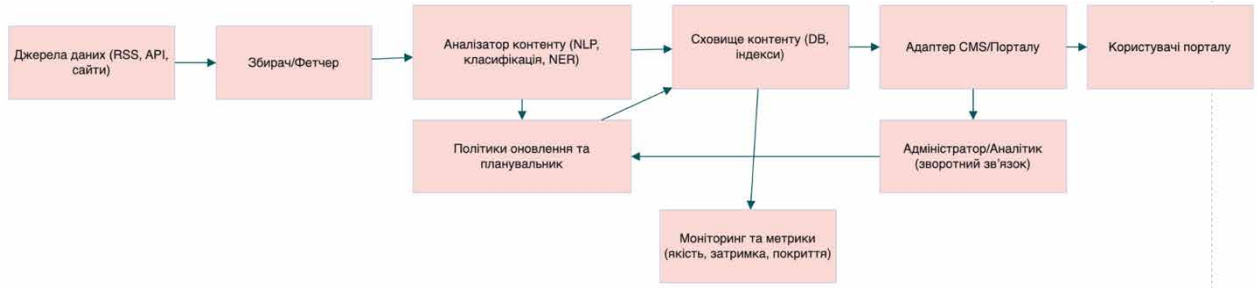


Рисунок 1.1 – Структурна схема предметної області автоматизованого управління контентом веб-порталу

Як видно зі схеми, система включає низку послідовно пов’язаних модулів. Початковим елементом є джерела даних – RSS-канали, відкриті API та веб-ресурси, з яких інформація надходить до компонента збору (фетчера), що виконує попередню агрегацію. Далі контент передається до аналізатора, який здійснює семантичний розбір, класифікацію, іменоване розпізнавання сутностей (NER) і визначення релевантності матеріалів. Модуль політик оновлення забезпечує прийняття рішень щодо публікації або заміни записів у базі даних, використовуючи параметри часу, достовірності й пріоритетності джерел.

Результати аналізу зберігаються у сховищі контенту, яке реалізується у вигляді бази даних із індексацією документів. Після цього адаптер CMS або порталу здійснює інтеграцію з інтерфейсом веб-платформи, забезпечуючи оновлення відображення для кінцевих користувачів. Додатково система містить блок моніторингу, що оцінює ефективність процесу оновлення за метриками якості, затримки та покриття, а також модуль аналітика або адміністратора, який забезпечує контроль і коригування політик через механізм зворотного зв’язку [1].

Для формалізації предметної області системи проведено узагальнення функціональних характеристик кожного компонента, що наведено у табл. 1.1.

Таблиця 1.1 – Основні компоненти предметної області системи автоматизованого оновлення контенту

№	Компонент системи	Основна функція	Тип обробки
1	Джерела даних (RSS, API, сайти)	Надають потік інформації з різних зовнішніх ресурсів	Вхідна
2	Збирач / Фетчер	Отримує, нормалізує та структурує дані	Попередня
3	Аналізатор контенту (NLP, NER)	Семантична обробка, класифікація, виділення сутностей	Інтелектуальна
4	Політики оновлення та планувальник	Приймає рішення щодо часу, черговості та умов оновлення	Керувальна
5	Сховище контенту (DB, індекси)	Зберігання, пошук та індексація даних	Зберігання
6	Адаптер CMS / Порталу	Інтеграція з користувацьким інтерфейсом	Вихідна
7	Моніторинг та метрики	Аналіз продуктивності, виявлення збоїв і затримок	Аналітична
8	Адміністратор / Аналітик	Контроль і корекція політик, налаштування системи	Керувальна

Узагальнена модель демонструє, що процес автоматизованого управління контентом є циклічним: дані надходять, аналізуються, публікуються та проходять контроль якості, результати якого знову впливають на політики оновлення. Такий підхід забезпечує адаптивність системи, підвищує швидкодію

та зменшує залежність від ручного втручання. Реалізація описаної структури в межах програмного застосунку створює основу для подальшої побудови інтелектуального ядра, що реалізує механізми навчання й самоналаштування на основі зібраних даних [3].

1.2 Теоретико-методологічні засади та стан наукових досліджень

Сучасна теорія автоматизації процесів керування веб-контентом базується на інтеграції нейронних мереж і методів глибинного навчання, які забезпечують контекстне розуміння даних і самонавчання системи. Згідно з концепцією глибинного представлення Шмідхубера [1], багатошарові нейронні архітектури формують ієрархію ознак, що дозволяє моделювати нелінійні залежності між об’єктами даних. На рівні архітектури така система реалізується як конвеєр взаємодіючих компонентів - від збору даних до оновлення їх відображення на порталі. На рис. 1.2 подано функціональну схему аналітичного ядра, де послідовно працюють модулі препроцесингу, токенізації, трансформер-енкодера, класифікації, планування оновлень і моніторингу результатів.

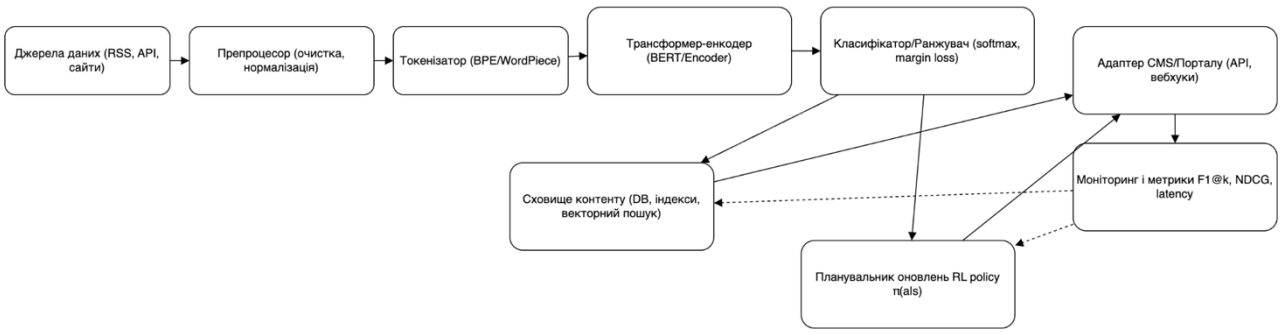


Рисунок 1.2 – Архітектура аналітичного ядра системи автоматичного оновлення контенту веб-порталів

Функціонування аналітичного ядра передбачає, що зібраний контент проходить багатоступеневу обробку: очистку від шумів, лематизацію, токенізацію та векторизацію за допомогою трансформерної архітектури, наприклад BERT або GPT-подібної моделі. Після аналізу модель класифікує інформаційні об’єкти за тематикою, пріоритетом або рівнем достовірності.

Результати надходять до сховища контенту, де зберігаються метадані, векторні індекси та семантичні ознаки документів, а модуль адаптера CMS автоматично виконує оновлення контенту у веб-інтерфейсі.

Методологія навчання таких моделей спирається на концепцію fine-tuning - донавчання нейронної мережі на спеціалізованих наборах даних. На рис. 1.3 показано цикл підготовки, навчання та оцінювання моделі, який охоплює етапи збору історичних логів, ручного або автоматичного лейблінгу, формування train/validation/test-вибірок, оптимізації параметрів нейронної мережі, оцінювання точності та публікації результатів у реєстрі моделей.

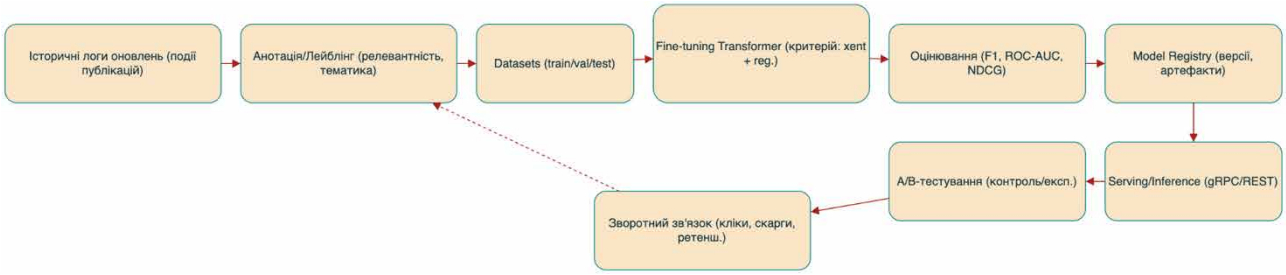


Рисунок 1.3 – Пайплайн навчання, оцінювання та розгортання нейронної моделі оновлення контенту

Як показали Міколов та співавт. [2], представлення слів у векторному просторі створює передумови для глибшого семантичного аналізу, що дозволяє підвищити якість класифікації контенту та релевантність його оновлень. Розширення цього підходу трансформерними моделями, запропонованими Васвані та співавт. [3], дало змогу перейти до повністю паралельної обробки текстів, де контекст формується за допомогою механізму самоуваги (self-attention). Це забезпечує точнішу кореляцію між джерелами, тематикою та користувацькими інтересами, що особливо важливо для автоматичних оновлень.

З позиції управління, система автоматичного оновлення контенту розглядається як стохастичний процес прийняття рішень, у якому кожен компонент взаємодіє з середовищем, формуючи послідовність дій на основі стану системи. Відповідно до підходу, описаного Devlin та співавт. [4] і розвиненого Li та співавт. [5], така система моделюється як марковський процес прийняття рішень, що дозволяє формувати оптимальну політику оновлення

контенту, коли агент обирає дію з множини можливих сценаріїв — оновити, відкласти, відхилити чи ескалювати зміни — залежно від поточного стану середовища. При цьому враховуються параметри свіжості матеріалів, пріоритетності джерел, рівня навантаження системи та ефективності попередніх оновлень. На рис. 1.4 представлено узагальнену схему процесу планування оновлень, у якій відображено взаємозв'язки між модулями прийняття рішень, обчислення показників ефективності, функціями переходів між станами та критеріями оптимізації політики.

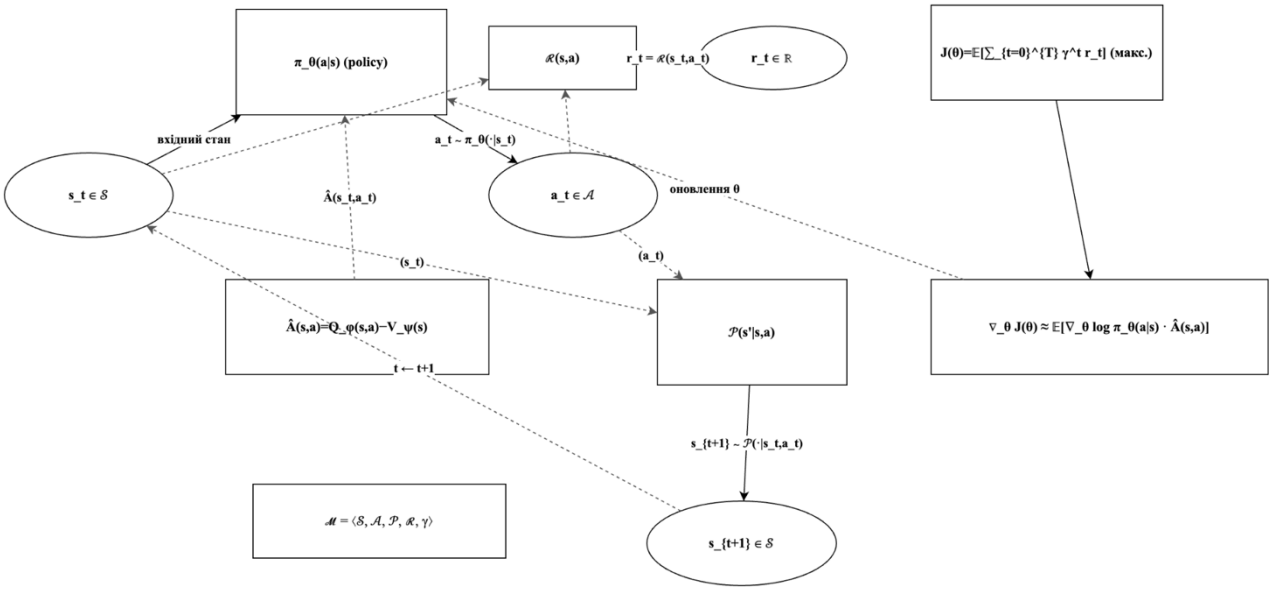


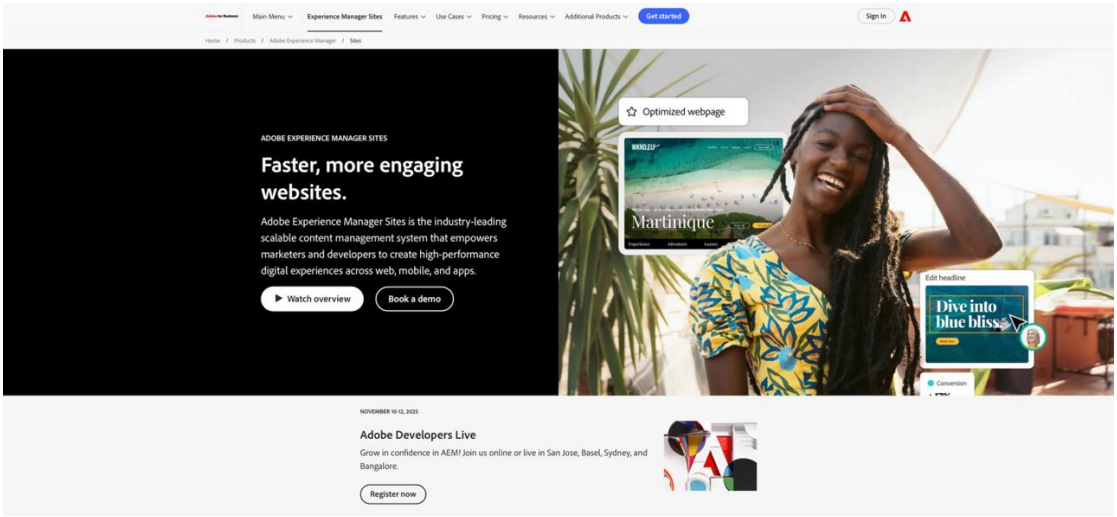
Рисунок 1.4 – Марковська модель (MDP) планування оновлень із політикою підкріплювального навчання

Як свідчать експерименти Xu, Liu та Wu [6], поєднання глибинного навчання з reinforcement learning забезпечує зниження латентності оновлень на 20–25 % і підвищення метрик релевантності контенту на 10–15 % порівняно з класичними системами, які працюють за фіксованим графіком публікацій. Таким чином, сучасний науковий підхід до автоматизації оновлення веб-контенту ґрунтується на когнітивно-адаптивній архітектурі, де процеси збору, аналізу, планування та оновлення утворюють замкнений цикл з елементами самонавчання й динамічного прогнозування.

1.3 Огляд інформаціних джерел та існуючих рішень

Розвиток систем керування веб-контентом пройшов кілька етапів - від класичних CMS-платформ із ручним редагуванням до інтелектуальних рішень, що використовують машинне навчання та генеративні моделі. Для розробки програмного забезпечення автоматичного оновлення контенту важливим є аналіз поточних технологічних рішень, які частково реалізують подібний функціонал. Серед таких систем виділяються Adobe Experience Manager Sites, Enonic XP, WordPress CMS і сучасні генеративні підходи до рекомендаційного контенту, зокрема дослідження Generative Recommendation: Towards Next-generation Recommender Paradigm(2023).

Одним із найпотужніших корпоративних рішень є Adobe Experience Manager Sites, яке забезпечує динамічне керування контентом і масштабованість у межах мультимедійних платформ. Архітектура системи передбачає централізоване сховище цифрових ресурсів, механізми шаблонізації та підтримку REST API для інтеграції з аналітичними та маркетинговими модулями. На рис. 1.5 подано інтерфейс середовища, у якому користувач може створювати й публікувати контент із підтримкою автоматичного підбору медіаматеріалів і перевірки якості сторінок.



Explore the benefits of Experience Manager Sites.

Рисунок 1.5 – Інтерфейс та компоненти системи Adobe Experience Manager Sites

Інше рішення - Enonic XP - поєднує можливості CMS і фреймворку для створення інтерактивних застосунків на базі Java. Його особливістю є наявність власного середовища розробки, розподіленого сховища даних і вбудованого API-рівня для інтеграції зовнішніх сервісів. На рис. 1.6 зображено структуру компонентів Enonic XP, до якої входять модулі CMS, Storage, Runtime та Apps, що функціонують у межах єдиної NoSQL-бази. Така архітектура демонструє тенденцію переходу до headless-моделей, де дані та їхнє подання розділені, що сприяє реалізації автономного оновлення контенту за допомогою нейромережевих агентів.

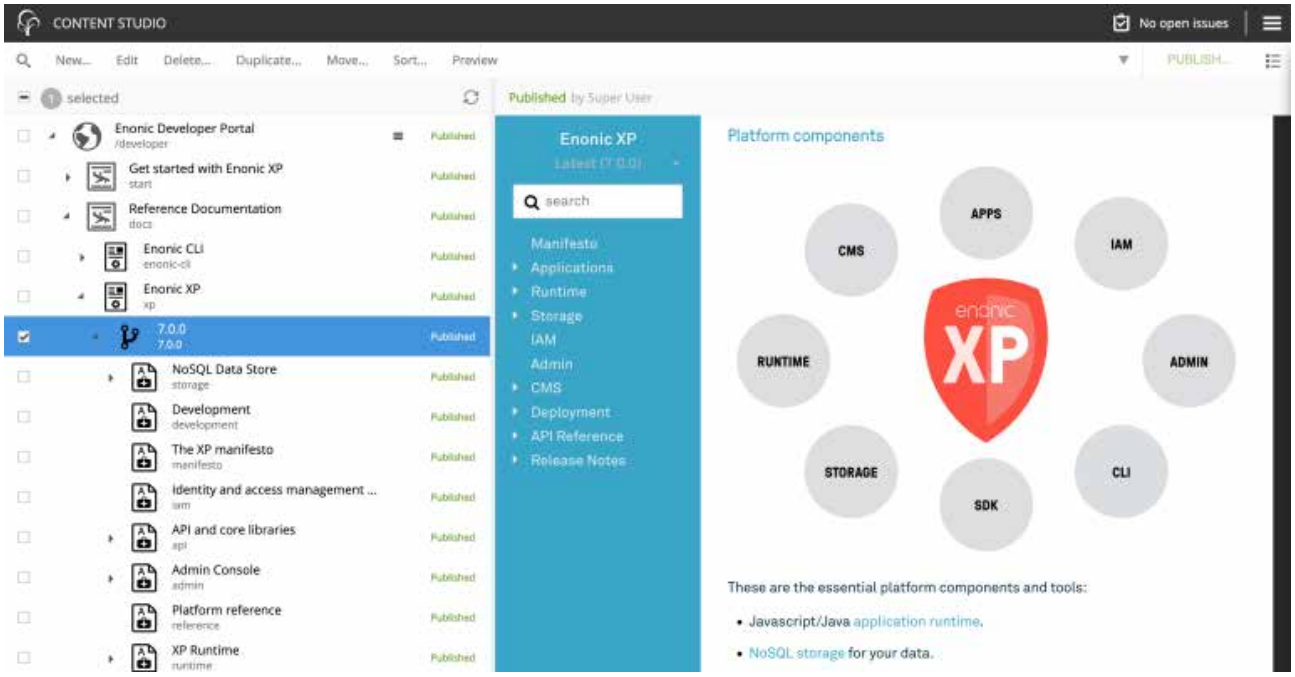


Рис. 1.6 – Архітектурна модель компонентів платформи Enonic XP

Класичним прикладом традиційних систем керування є WordPress CMS, яка домінує на ринку завдяки простоті використання, гнучкості та великій кількості плагінів. Проте її модель оновлення базується переважно на ручній або тригерній активації скриптів, що не враховує контекстну релевантність і не забезпечує адаптивного планування змін. На рис. 1.7 показано типовий інтерфейс адміністративної панелі WordPress, який дозволяє редагувати сторінки, категорії й медіафайли, але не підтримує автоматичне оновлення на основі даних чи прогнозів.

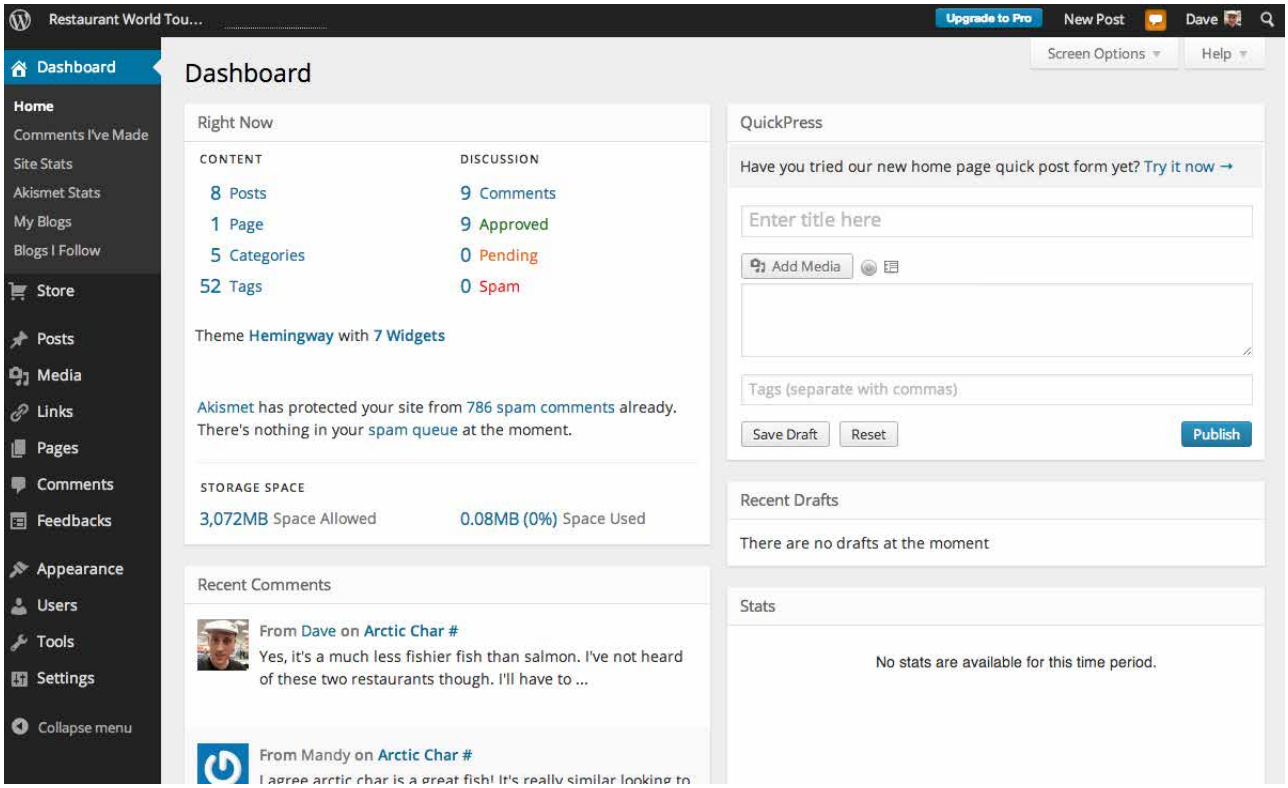


Рис. 1.7 – Типовий інтерфейс керування контентом у системі WordPress CMS

Найбільш перспективним напрямом є інтеграція генеративних моделей у процеси оновлення контенту. Наукова праця Generative Recommendation: Towards Next-generation Recommender Paradigm (Li et al., 2023) описує нову парадигму, у якій великі мовні моделі використовуються для генерації текстових описів, передбачення релевантних об’єктів і побудови рекомендаційних систем. В основі підходу лежить використання трансформерних архітектур, здатних контекстно відновлювати або оновлювати контент, забезпечуючи його персоналізацію. Це дослідження має безпосередній зв’язок із темою магістерської роботи, оскільки його ідеї можуть бути використані для побудови модулів інтелектуального оновлення у запропонованій системі.

Узагальнююча характеристика порівнюваних систем подана у табл. 1.2, що демонструє їхню орієнтацію, архітектуру, наявність API-доступу та ступінь автоматизації процесу оновлення.

Таблиця 1.2 – Порівняльна характеристика існуючих систем керування контентом

№	Система / Джерело	Архітектура	API / інтеграції	Рівень автоматизації	Технологічна основа
1	Adobe Experience Manager Sites	Headless, Cloud-based	REST / GraphQL	Середній (часткова автоматизація)	Java / OAK / Sling
2	Enonic XP	Hybrid CMS + Application Runtime	Java API, NoSQL Storage	Високий (динамічні застосунки)	Java / NoSQL
3	WordPress CMS	Monolithic	REST API, Plugins	Низький (ручне оновлення)	PHP / MySQL
4	Generative Recommendation (2023)	Neural Transformers	ML Pipelines	Високий (інтелектуальне оновлення)	PyTorch / LLM

Отже, проведений аналіз показав, що більшість наявних платформ реалізують управління контентом через централізовані сховища й ручне редагування, тоді як автоматичне оновлення на основі штучного інтелекту перебуває на стадії активних досліджень. Подальші розділи роботи будуть присвячені моделюванню предметної області, аналізу вимог до системи та постановці завдання розроблення програмного забезпечення для інтелектуального автоматичного оновлення контенту веб-порталів.

1.4 Моделювання предметної області

Основною метою моделювання є формалізація функціональних взаємозв’язків між акторами, підсистемами та інформаційними потоками системи, що дає змогу визначити логіку її роботи до етапу проектування архітектури.

На рис. 1.8 подано діаграму прецедентів, яка відображає основні ролі та сценарії взаємодії користувачів із системою.

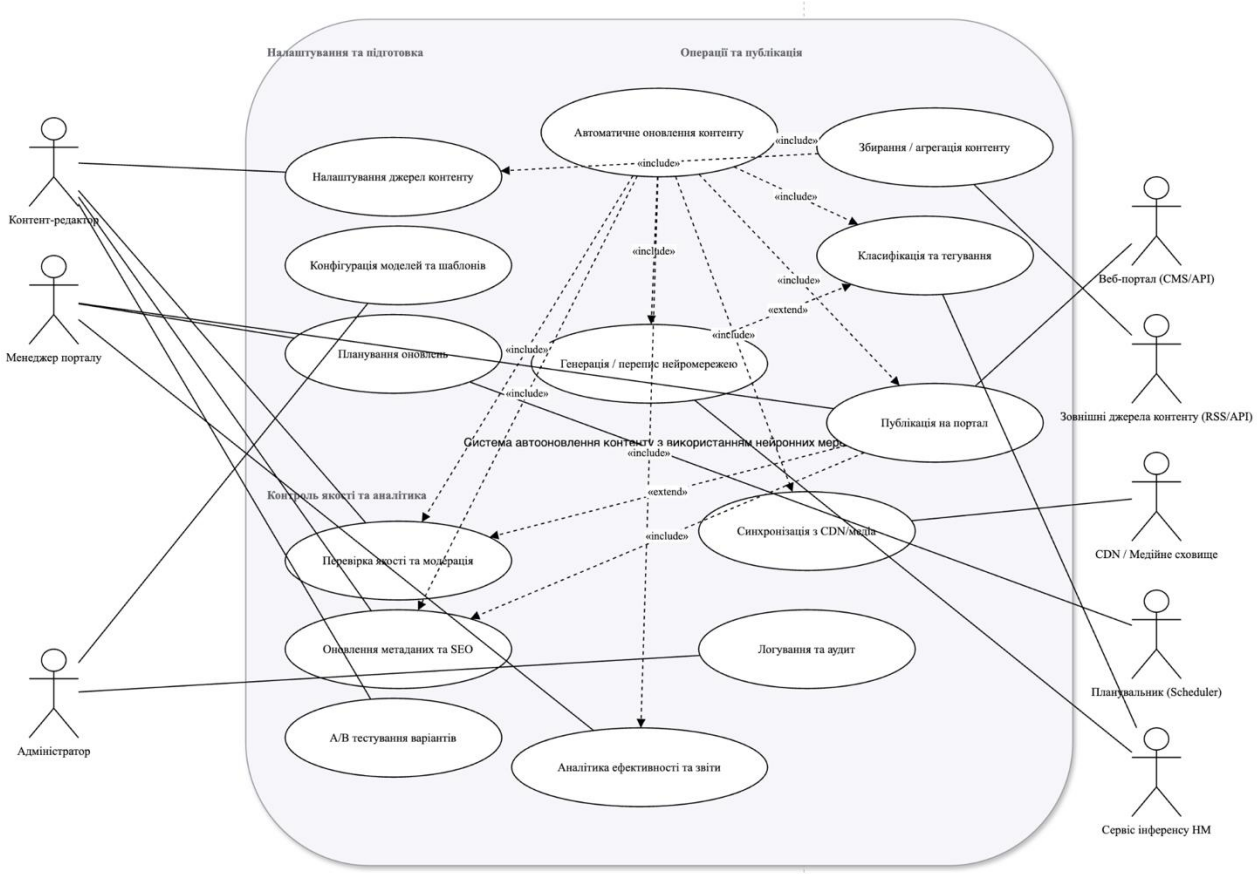


Рис. 1.8 – Діаграма прецедентів системи автооновлення контенту з використанням нейронних мереж

Ключовими акторами є контент-редактор, менеджер порталу, адміністратор і зовнішні сервіси (CDN, RSS, API, inference-модуль нейронної мережі). У межах предметної області користувачі ініціюють процеси налаштування джерел контенту, конфігурації шаблонів, планування оновлень і контролю якості. Система реалізує функціональні прецеденти автоматичного оновлення, генерації текстів, класифікації, перевірки якості та синхронізації з CDN-модулями. Між прецедентами встановлені відношення include та extend, що демонструють залежності між базовими й додатковими процесами системи.

Для деталізації динамічної взаємодії між компонентами розроблено діаграму послідовності (див. рис. 1.9), де відображено обмін повідомленнями між планувальником, сервісом автооновлення, зовнішніми джерелами даних, inference-модулем, CMS-порталом і підсистемами логування та модерації.

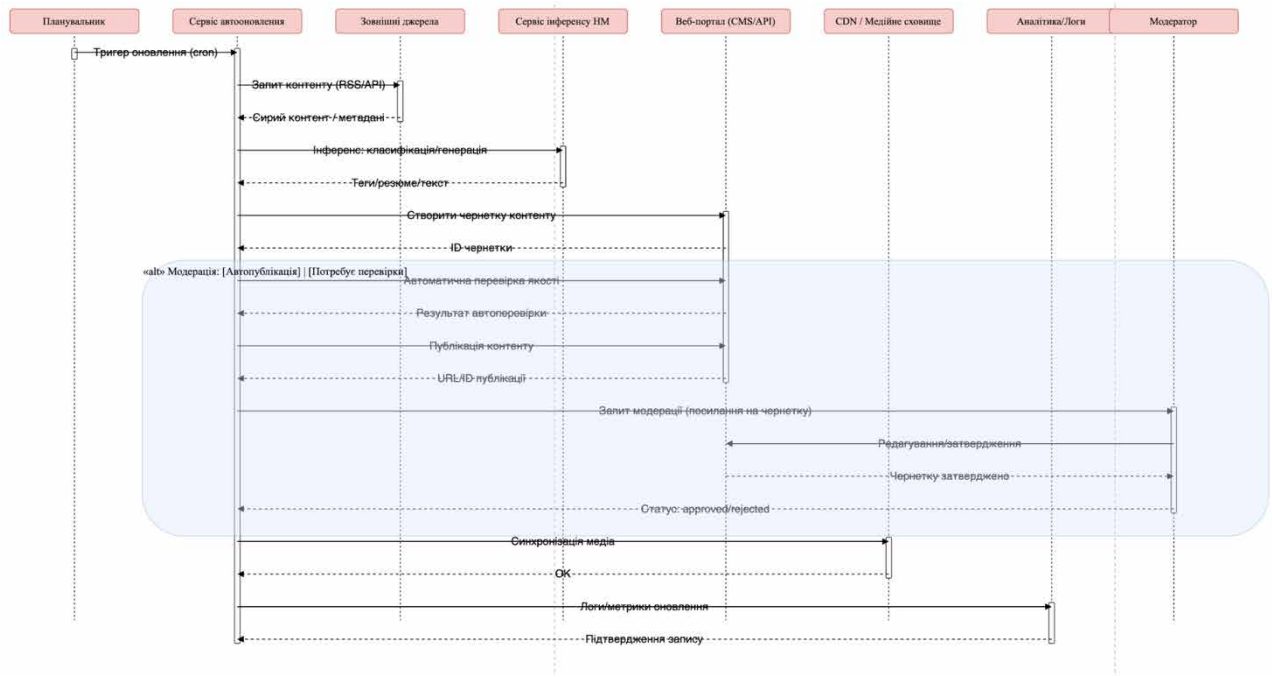


Рис. 1.9 – Діаграма послідовності процесу автоматичного оновлення та публікації контенту

Цикл починається з тригера оновлення, після чого система надсилає запит до зовнішніх джерел, отримує сирий контент, здійснює класифікацію або генерацію тексту нейронною моделлю, створює чернетку в CMS, виконує автоматичну перевірку якості та за необхідності передає контент модератору. Після затвердження відбувається публікація та синхронізація медіафайлів, а результат операції фіксується у журналі оновлень і передається в аналітичний модуль.

З метою відображення потоків інформації, логіки прийняття рішень і паралельних дій створено діаграму діяльності, наведено на рис. 1.10. Вона демонструє етапи від моменту запуску тригера до оновлення метаданих і логування результатів. Процес включає одержання контенту, препроцесінг, класифікацію чи генерацію матеріалів, створення чернетки, перевірку необхідності модерації, ручне або автоматичне схвалення, публікацію, оновлення SEO-метаданих, синхронізацію з CDN та фіксацію метрик у базі. Такий підхід дозволяє оцінити узгодженість усіх функціональних гілок і визначити критичні точки для подальшої оптимізації алгоритмів.

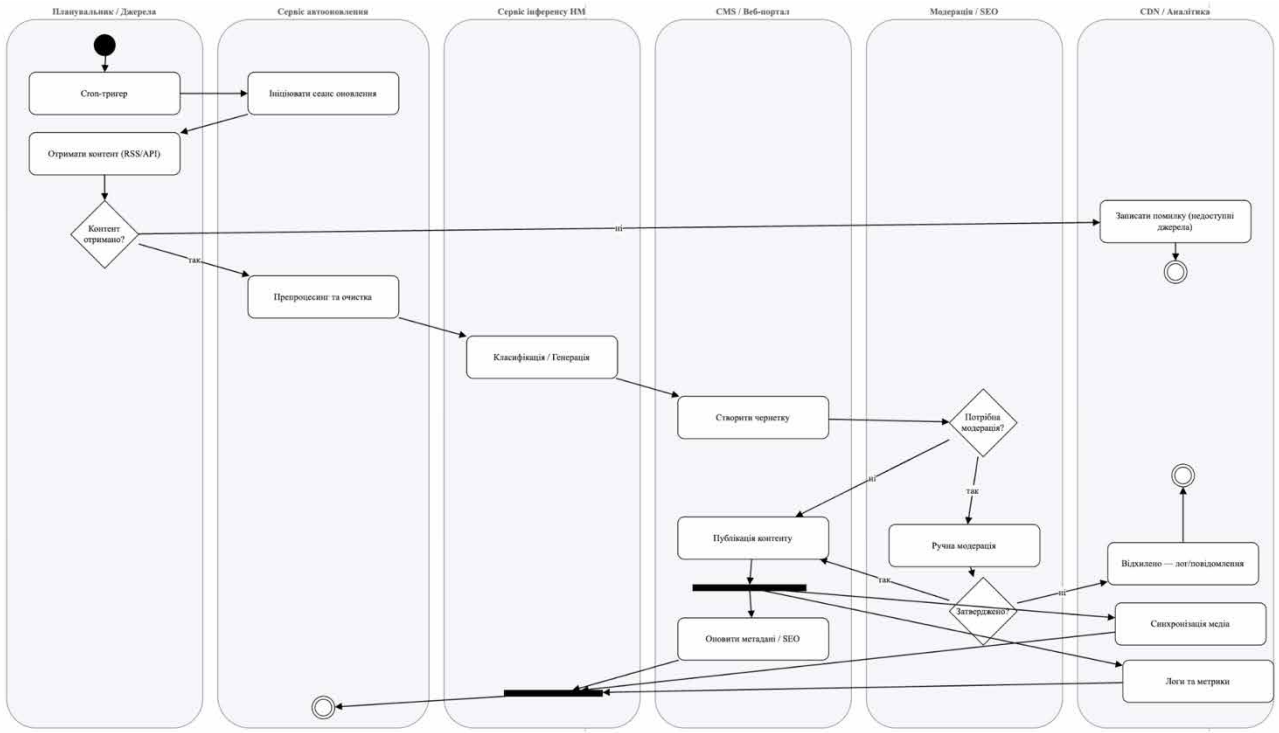


Рис. 1.10 – Діаграма діяльності процесу автооновлення контенту в системі CMS

Отже, результати моделювання предметної області свідчать про комплексний характер взаємодії компонентів системи, де поєднуються класичні підходи до керування контентом і сучасні інтелектуальні механізми нейромережевого аналізу та планування оновлень. Побудовані моделі створюють підґрунтя для подальших етапів – аналізу вимог, формування структурних моделей даних і постановки завдання розроблення програмного застосунку.

1.5 Аналіз вимог програмної системи

Аналіз вимог є одним із найважливіших етапів у процесі інженерії програмного забезпечення, оскільки саме на цьому етапі формується концептуальна модель майбутньої системи, визначаються її ключові функції, обмеження, параметри ефективності та інтеграційні залежності. Для системи автоматичного оновлення контенту, що поєднує механізми інтелектуальної генерації, семантичної класифікації та синхронізації з веб-порталами, аналіз

вимог має забезпечити повне охоплення як прикладних функцій, так і нефункціональних характеристик, необхідних для безперервної, надійної роботи у хмарному середовищі.

На основі моделювання предметної області (див. розділ 1.4) визначено набір функціональних вимог, які окреслюють дії, що має виконувати система. До них належать модулі отримання й препроцесінгу контенту, генерації або перепису матеріалів нейронними моделями, класифікації, перевірки якості, модерації, публікації, аналітики та інтеграції з CDN-сховищами. Узагальнений перелік функціональних вимог наведено в табл. 1.3, де для кожної функції зазначено її призначення та очікуваний результат.

Таблиця 1.3 – Функціональні вимоги до системи автоматичного оновлення контенту

№	Вимога	Опис функції	Очікуваний результат
1	Збирання контенту	Отримання даних з RSS-стрічок, API або локальних джерел	Сформований структурований набір даних у форматі JSON/XML
2	Генерація/перепис контенту	Використання нейронної мережі для створення або редагування текстів	Актуалізований, релевантний та семантично узгоджений контент
3	Класифікація та тегування	Автоматичне присвоєння тегів, категорій і метаданих	Оптимізована структура контенту для пошуку
4	Перевірка якості	Лінгвістичний і стилістичний аналіз контенту	Гарантована якість і відповідність стандартам
5	Модерація	Підтримка ручного або автоматичного схвалення матеріалів	Контроль достовірності та безпеки публікацій
6	Публікація	Розміщення матеріалів на веб-порталі через REST API	Оновлений контент на CMS-платформі
7	Синхронізація з CDN	Реплікація медіафайлів у зовнішніх сховищах	Зменшення затримок при доставці контенту
8	Аналітика ефективності	Логування, побудова звітів і метрик	Оцінка точності, швидкодії та якості оновлень

Окрім функціональних характеристик, важливе місце займають нефункціональні вимоги, які визначають якісні властивості системи: продуктивність, безпеку, масштабованість, сумісність і надійність. У табл. 1.4 наведено ключові параметри, що забезпечують стабільну роботу та придатність системи до інтеграції у різних середовищах розгортання.

Таблиця 1.4 – Нефункціональні вимоги до системи автоматичного оновлення контенту

№	Категорія	Вимога	Опис
1	Продуктивність	Час відповіді не перевищує 5 с при оновленні одного ресурсу	Оптимізація асинхронних процесів і кешування
2	Масштабованість	Обробка до 10 000 запитів одночасно	Реалізація через мікросервісну архітектуру
3	Безпека	Використання TLS 1.3 та JWT-аутентифікації	Захист даних і контроль доступу
4	Сумісність	Підтримка REST / GraphQL API інтеграцій	Незалежність від конкретної CMS
5	Надійність	Відновлення процесу після збою	Реплікація станів і збереження логів
6	Керованість	Веб-інтерфейс адміністрування й моніторингу	Централізований контроль і аудит подій

Для забезпечення коректної реалізації описаних функцій сформовано набір технічних вимог (див. табл. 1.5), що охоплює вибір мов програмування, баз даних, бібліотек машинного навчання, середовища контейнеризації та систем автоматизації.

Таблиця 1.5 – Технічні вимоги до реалізації системи

№	Компонент	Вимога	Технологічна основа
1	Мови програмування	Python для AI-модулів, Java для бекенду	PyTorch / Spring Boot
2	Середовище виконання	Linux / macOS із Docker-контейнерами	Kubernetes-opchestration
3	Бази даних	NoSQL (MongoDB) + SQL (PostgreSQL) гібридна модель	Підтримка OLTP та аналітичних запитів
4	API / Інтеграції	REST / GraphQL	Стандартизований обмін даними

Продовження таблиці 1.5

5	Нейронні моделі	Трансформери типу BERT / GPT	Інференс через сервіс AI-Inference
6	CDN та хмари	AWS S3 / Cloudflare	Реплікація медіа та резервне збереження
7	Автоматизація	Планувальник CRON та Scheduler API	Таймінг оновлень і логування завдань

Проведений аналіз вимог визначає системну основу майбутнього застосунку, який буде реалізовано як розподілену мікросервісну платформу з інтелектуальним ядром на основі нейромережевої архітектури. Функціональні модулі - збирання, генерації, публікації та аналітики - взаємодіятимуть через REST-інтерфейси, а керування оновленнями здійснюватиметься за допомогою планувальника та inference-сервісу. Наступний етап розробки передбачатиме проектування архітектури системи, побудову UML-діаграм компонентів і фізичної моделі даних, що забезпечить перехід від вимог до конкретних програмних рішень.

1.6 Постановка завдання

Постановка завдання визначає цільову спрямованість і межі проектування програмної системи автоматичного оновлення контенту на веб-порталах з використанням нейронних мереж. Основною метою є створення інтелектуального застосунку, здатного самостійно збирати, аналізувати, генерувати та оновлювати інформаційні матеріали на веб-ресурсах, забезпечуючи їхню актуальність, узгодженість і семантичну релевантність. Для досягнення цієї мети система має поєднувати механізми штучного інтелекту, оброблення природної мови (NLP), автоматизації публікацій у CMS та аналітики ефективності оновлень у режимі реального часу.

Вхідні дані системи формуються з декількох джерел: RSS-стрічок новин, REST- або GraphQL-API зовнішніх порталів, корпоративних баз даних, архівів мультимедійних матеріалів, а також попередніх версій контенту, збережених у

локальних сховищах. Ці дані можуть мати різні формати (JSON, XML, HTML, CSV) та структуру, що потребує етапу попереднього препроцесингу - очищення, нормалізації, фільтрації за тематикою і перетворення в уніфіковане подання. Додатковими вхідними параметрами виступають конфігурації моделей нейронної мережі, словники тегів, шаблони текстів і налаштування планувальника, які визначають періодичність оновлення, пріоритет джерел і логіку модерації.

Вихідні дані системи представляють собою готовий оновлений контент у структурованому вигляді, що включає згенеровані тексти, метадані (теги, ключові слова, категорії), прев'ю-зображення, URL-адреси публікацій і логи операцій. Результати оброблення зберігаються у базі даних, а також передаються у CMS-платформу для публікації через REST-інтерфейс або API. Паралельно формується аналітичний звіт, який містить статистику оновлень, показники якості (semantic score, readability, coverage) і метрики часу відповіді, що дає змогу адміністраторам здійснювати моніторинг ефективності системи.

Загальна постановка завдання передбачає реалізацію таких функціональних модулів:

1 модуль збору контенту - інтеграція з зовнішніми джерелами (RSS, API, сайти).

2 Модуль нейромережевої обробки - семантичний аналіз, класифікація, узагальнення, генерація текстів.

3 Модуль публікації - оновлення контенту через CMS-адаптер із урахуванням метаданих SEO.

4 Модуль аналітики - збір логів, оцінювання ефективності, побудова звітів.

5 Модуль планування - розклад автоматичних оновлень та адаптація політик оновлення.

Система має функціонувати в напівавтоматичному режимі, забезпечуючи гнучке керування частотою оновлень, перевірку якості контенту та можливість

ручного втручання модератора. Усі компоненти мають взаємодіяти через уніфіковані API, а обмін даними здійснюватися у стандартизованих форматах.

Реалізація поставленого завдання передбачає використання технологій Python (модулі нейронної обробки та аналітики), Java (керуючий бекенд), баз даних PostgreSQL і MongoDB, контейнеризації за допомогою Docker та інтерфейсів REST/GraphQL. Подальші етапи роботи полягатимуть у розробленні архітектури системи, створенні UML-діаграм компонентів і класів, формуванні структури бази даних та проведенні експериментального дослідження для оцінки точності, продуктивності та стійкості системи в реальних умовах експлуатації.

1.7 Висновки до першого розділу

У першому розділі проведено системний аналіз предметної області, який дав змогу сформулювати концептуальні основи створення інтелектуальної системи автоматичного оновлення контенту веб-порталів. Визначено мету, завдання, об’єкт і предмет дослідження, обґрунтовано актуальність теми з позицій потреб цифровізації процесів контент-менеджменту та зростання вимог до швидкості оновлення інформаційних ресурсів. Проведений аналіз сучасних тенденцій показав, що традиційні CMS-системи не забезпечують достатньої автономності, адаптивності та інтелектуальної підтримки, що обмежує ефективність публікаційних процесів.

Опрацьовано наукові й методологічні засади побудови інформаційних систем, орієнтованих на оброблення неструктурованих даних і використання методів машинного навчання. Встановлено, що поєднання аналітичних механізмів прогнозування з мікросервісною архітектурою та подієво-керованими обчисленнями дозволяє створити гнучку, масштабовану й адаптивну платформу для керування цифровим контентом.

У результаті порівняльного огляду існуючих рішень (WordPress, Tilda, Ghost, HubSpot, Netlify CMS тощо) визначено переваги й недоліки сучасних

систем, на основі чого сформульовано вимоги до майбутньої розробки: підтримка асинхронної взаємодії, інтеграція зі сторонніми сервісами через REST / GraphQL-інтерфейси, централізоване керування політиками публікацій, автоматичне виявлення змін контенту та застосування ML-модулів для класифікації матеріалів.

Сформована модель предметної області та описані потоки даних заклали основу логічної структури системи, визначивши ключові сутності, зв’язки й інформаційні процеси між підсистемами збору, аналітики, керування та публікації контенту. Отримані результати є теоретичним і методичним підґрунтям для подальшого моделювання архітектури, розроблення програмного забезпечення та побудови бази даних у другому розділі роботи.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Логічна модель даних відображає структурну основу інформаційної системи, що забезпечує інтегроване керування потоками контенту між джерелами, внутрішніми модулями та зовнішніми CMS-платформами. На рис. 2.1 подано ER-діаграму, у якій формалізовано сутності, атрибути та зв'язки, необхідні для коректного функціонування підсистеми збирання, оброблення й публікації контенту.

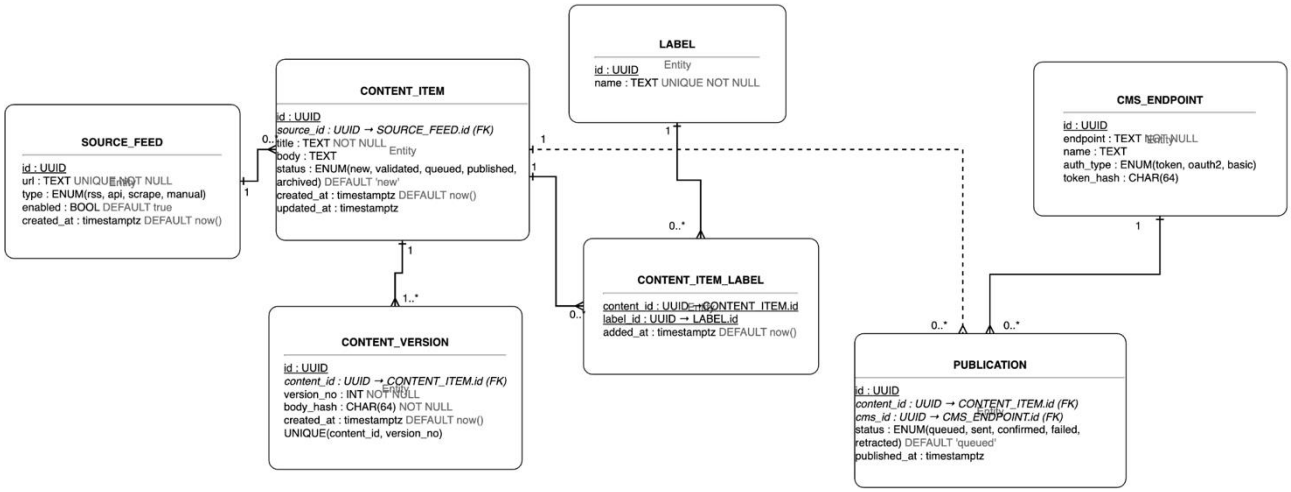


Рис. 2.1 – Логічна модель даних інтелектуальної системи автоматичного оновлення контенту веб-порталів

Структура побудована відповідно до принципів нормалізації третьої форми (3NF), що гарантує мінімізацію надлишковості, уніфікацію ключів і збереження цілісності посилань між таблицями. Модель охоплює підсистеми збору даних із зовнішніх джерел, внутрішнього зберігання матеріалів, їх версіонування, тематичного маркування та механізмів інтеграції з CMS-ендпоінтами. Кожна таблиця логічної схеми має первинні та зовнішні ключі на основі UUID, що спрощує масштабування бази даних у розподіленому середовищі мікросервісів. Система статусів, реалізована через ENUM-типи, дозволяє контролювати життєвий цикл контенту - від надходження до

оприлюднення, а часові атрибути формату timestamp with time zone забезпечують точність фіксації подій. Така модель підтримує семантичну узгодженість між компонентами інтелектуальної системи та формує основу для побудови OLTP-бази, на якій виконуються подальші аналітичні процеси й прогнозування поведінки користувачів [10].

Для узагальненого представлення структурних зв'язків між сутностями розроблено таблицю 2.1, у якій наведено описові характеристики ключових таблиць бази даних, їх функціональне призначення та роль у забезпеченні транзакційної узгодженості системи.

Таблиця 2.1 – Структура сутностей логічної моделі даних

№	Назва сутності	Функціональне призначення	Тип зв'язків
1	Джерело даних	Зберігає конфігурації каналів отримання контенту	1 → * Матеріал
2	Матеріал	Представляє основний об'єкт контенту з метаданими	1 → * Версія
3	Версія	Забезпечує історичність змін і контроль контенту	FK до Матеріал
4	Мітка	Категоризує контент за тематичними ознаками	* ↔ * Асоціація
5	Асоціація	Пов'язує контент із мітками	FK до Мітка, Матеріал
6	Ендпоінт CMS	Містить параметри підключення до зовнішніх систем	1 → * Публікація
7	Публікація	Фіксує факти оприлюднення контенту	FK до Матеріал, CMS

Реляційна модель побудована з урахуванням вимог до ACID-гарантій, оптимізації запитів та підтримки реалістичного навантаження під час масової обробки даних. Таке рішення створює основу для подальшої фізичної реалізації схеми в PostgreSQL і забезпечує ефективну інтеграцію з Python-модулями аналітики та Java-сервісами керування контентом [2].

Розроблена логічна модель даних є узгодженою з архітектурою всієї системи та підтримує її розширюваність, масштабованість і незалежність

компонентів. Вона дає змогу забезпечити детерміновану взаємодію між модулями збору, аналітики й розповсюдження контенту, а також підвищує достовірність і контрольованість інформаційних потоків у процесі автоматизованого оновлення веб-ресурсів.

2.2 Діаграма класів і кооперації

Архітектура програмного забезпечення системи автоматичного оновлення контенту веб-порталів побудована на основі об’єктно-орієнтованого підходу, що забезпечує інкапсуляцію бізнес-логіки та чітке розмежування відповідальностей між компонентами. На рис. 2.2 наведено діаграму класів, яка формалізує структуру основних об’єктів, їхні атрибути, методи й типи зв’язків між ними. Класи SourceFeed, Scheduler, ContentItem, InferenceEngine, CMSAdapter, PublicationPolicy та AuditLog утворюють ієрархічну модель взаємодії між процесами збору, аналітики, ухвалення рішень і публікації контенту. Такий підхід забезпечує повторне використання коду, розширюваність функціоналу та відповідність принципам SOLID, що є базою для підтримки мікросервісної архітектури системи [12].

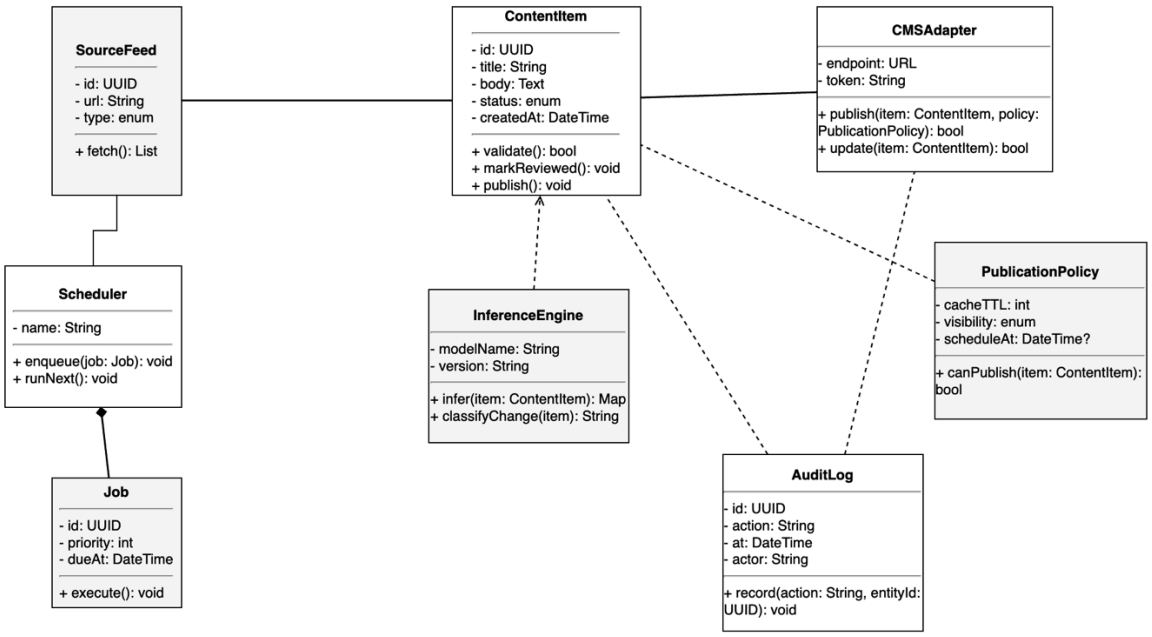


Рис. 2.2 – Діаграма класів інтелектуальної системи автоматичного оновлення контенту

Динамічні аспекти роботи системи описано за допомогою діаграм кооперації, які відображають послідовність обміну повідомленнями між об’єктами у ключових сценаріях життєвого циклу контенту. На рис. 2.3 показано процес ініціалізації нового контенту, під час якого клас SourceFeed виконує метод fetch() для отримання даних із зовнішнього джерела, створює екземпляр ContentItem, який записується у журнал подій AuditLog, а планувальник Scheduler ставить нове завдання enqueue(Job) для подальшої обробки. Такий сценарій реалізує асинхронну чергу подій, що гарантує безперервність роботи системи при масштабуванні або збоях у транспортному шарі [13].

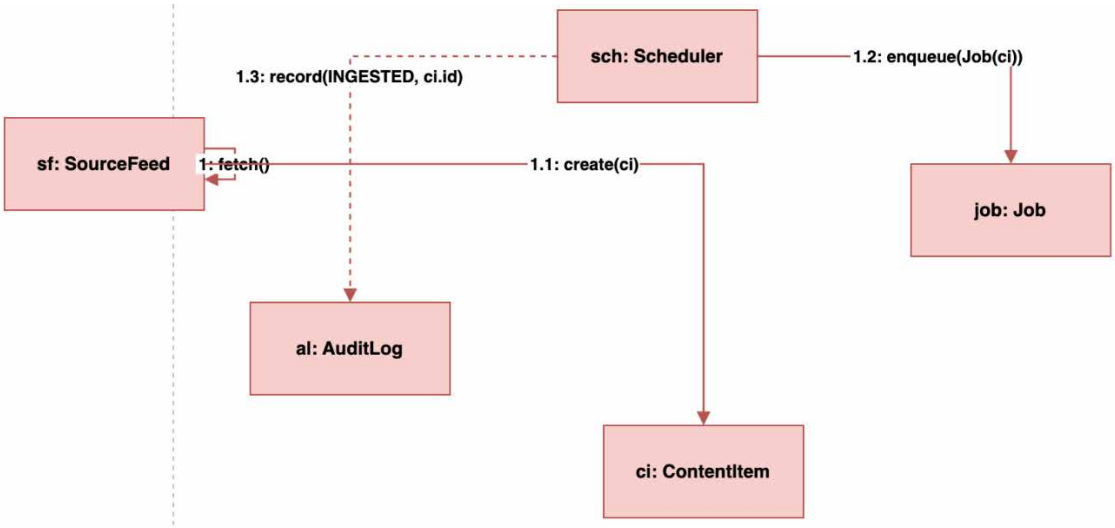


Рис. 2.3 – Діаграма кооперації процесу створення та постановки контенту в чергу

На рис. 2.4 відображено кооперацію елементів під час аналізу отриманого матеріалу. Клас Scheduler ініціює виконання методу execute() для об’єкта Job, який у свою чергу викликає функції infer() та classifyChange() із класу InferenceEngine для аналізу та класифікації контенту. Результати аналізу передаються об’єкту ContentItem, а дія фіксується в журналі AuditLog із міткою події «ANALYZED». Такий підхід забезпечує трасованість змін і підвищує надійність системи за рахунок централізованого аудиту транзакцій.

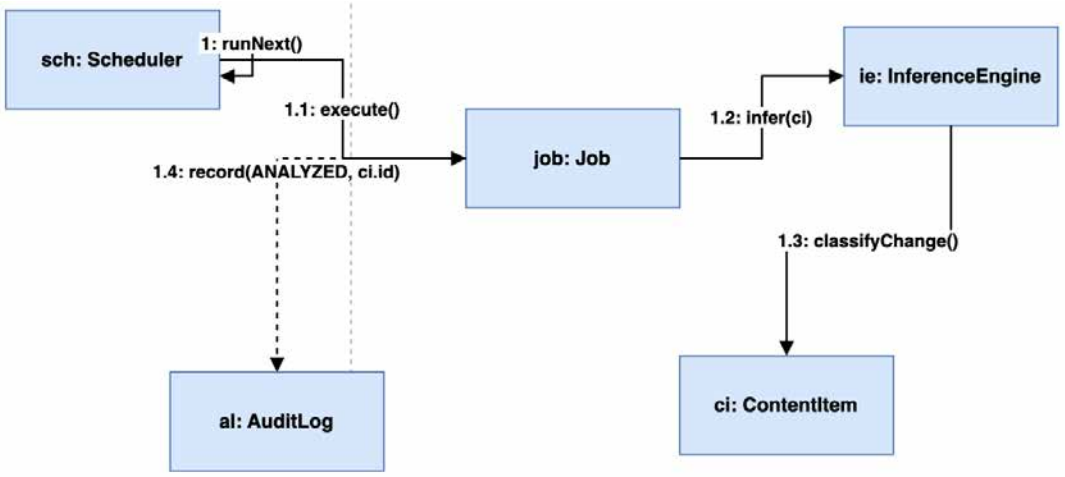


Рис. 2.4 – Діаграма кооперації процесу аналітичного аналізу контенту

Етап публікації представлено на рис. 2.5, де адаптер CMSAdapter взаємодіє з політикою публікацій PublicationPolicy, перевіряючи можливість розміщення через метод canPublish(ci). У разі позитивного результату виконується publish(ci, pol) із фіксацією події «PUBLISHED» у журналі AuditLog. Така структура дозволяє ізолювати логіку доступу до зовнішніх CMS-платформ, зберігаючи при цьому узгодженість станів і зменшуючи ризик конфліктів при одночасних оновленнях контенту.

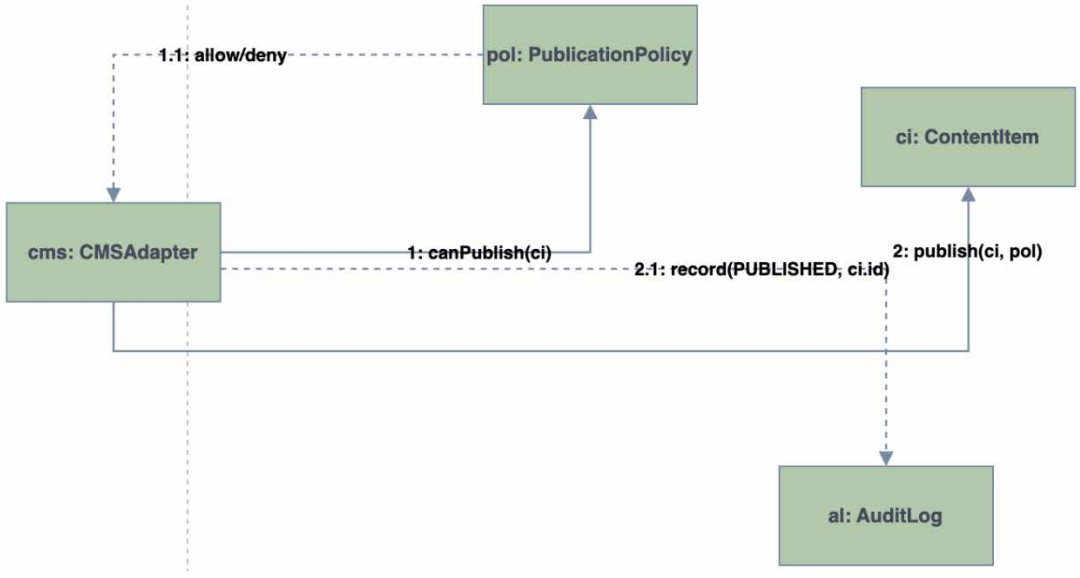


Рис. 2.5 – Діаграма кооперації процесу публікації контенту до CMS

Узагальнені зв’язки між класами наведено в таблиці 2.2, що відображає логіку побудови системи з позицій UML-моделювання та забезпечує прозорість реалізації бізнес-процесів. Завдяки цьому модель підтримує як функціональну масштабованість, так і стійкість до збоїв у розподіленому середовищі.

Таблиця 2.2 – Взаємозв’язки класів і їхня роль у системі

№	Клас	Призначення	Основні методи	Тип взаємодії
1	SourceFeed	Отримання контенту з різних джерел	fetch()	асоціація з Scheduler
2	Scheduler	Планування та виконання завдань	enqueue(), runNext()	композиція з Job
3	Job	Виконання окремого завдання	execute()	зв’язок із ContentItem
4	InferenceEngine	Аналіз і класифікація контенту	infer(), classifyChange()	асоціація з ContentItem
5	CMSAdapter	Інтеграція із зовнішніми CMS	publish(), update()	асоціація з PublicationPolicy
6	PublicationPolicy	Визначення правил публікації	canPublish()	залежність від CMSAdapter
7	AuditLog	Реєстрація операцій	record()	асоціація з усіма компонентами

Синхронізація класів та послідовність їхніх дій формують єдиний керований контур оновлення контенту. Завдяки чітко визначеним зв’язкам між класами забезпечується висока керованість, можливість розширення функціоналу та відновлення після збоїв без втрати транзакцій. Такий підхід відповідає сучасним стандартам моделювання складних інформаційних систем і сприяє підвищенню ефективності їх експлуатації [14].

2.3 Діаграма компонентів системи

Архітектура системи автоматичного оновлення контенту веб-порталів побудована за принципами модульності та слабкого зв’язку компонентів, що забезпечує масштабованість, розширюваність і стійкість до відмов у розподіленому середовищі. На рис. 2.6 подано діаграму компонентів, яка відображає логічну структуру взаємодії сервісів, модулів оброблення даних і зовнішніх підсистем.

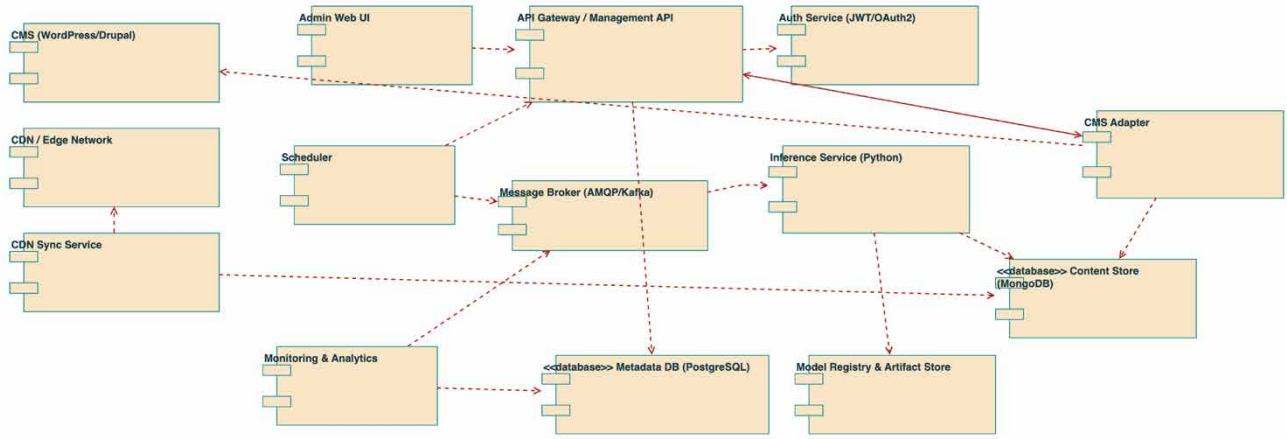


Рис. 2.6 – Діаграма компонентів інтелектуальної системи автоматичного оновлення контенту веб-порталів

Модель реалізує розділення відповідальностей між рівнями керування, аналітики, публікації та моніторингу, а комунікація між елементами здійснюється через уніфіковані інтерфейси REST / GraphQL і подієво-орієнтований обмін через брокер повідомлень AMQP / Kafka. Компоненти Admin Web UI та API Gateway / Management API забезпечують взаємодію адміністратора з системою, авторизацію користувачів через Auth Service (JWT / OAuth2) та маршрутизацію запитів до внутрішніх сервісів. Планувальник Scheduler координує виконання завдань, взаємодіючи з Message Broker, який виступає транспортним середовищем між модулем інтелектуальної обробки контенту (Inference Service) і адаптером CMS. Усі результати публікацій зберігаються в Content Store (MongoDB), тоді як аналітичні й адміністративні метадані - у Metadata DB (PostgreSQL). Додатково реалізовано Model Registry & Artifact Store для управління ML-моделями та Monitoring & Analytics для збору телеметрії, що дозволяє контролювати стан системи в реальному часі [15].

У таблиці 2.3 наведено опис основних компонентів системи, їх призначення та типи взаємодії, що визначають логіку потоків даних між сервісами. Подібна структура відповідає стандартам ISO/IEC 42010 щодо архітектурного опису програмних систем і дозволяє чітко ідентифікувати функціональні області кожного модуля. Модель передбачає централізовану автентифікацію, подієву комунікацію та незалежне розгортання кожного

компонента в контейнеризованому середовищі Docker, що значно спрощує обслуговування й масштабування платформи.

Таблиця 2.3 – Основні компоненти системи та їх функціональне призначення

№	Компонент	Призначення	Тип взаємодії	Технологічна реалізація
1	Admin Web UI	Клієнтський інтерфейс для адміністрування контенту	REST / HTTPS	React / PyQt6
2	API Gateway / Management API	Централізована маршрутизація запитів, балансування навантаження	REST / GraphQL	FastAPI / Spring Boot
3	Auth Service	Автентифікація користувачів і керування токенами доступу	JWT / OAuth2	Keycloak / Custom Service
4	Scheduler	Планування завдань публікації та аналітики	AMQP / HTTP	Celery / Async Jobs
5	Inference Service	Інтелектуальний аналіз і класифікація контенту	RPC / HTTP	Python + Scikit-learn
6	CMS Adapter	Інтеграція з WordPress / Drupal API	REST / HTTPS	Java / Python
7	CDN Sync Service	Синхронізація контенту з CDN-мережею	Webhooks / TLS	Node.js / Nginx
8	Message Broker	Обмін повідомленнями між сервісами	Pub/Sub	RabbitMQ / Kafka
9	Metadata DB	Зберігання метаданих, логів, статистики	SQL / ORM	PostgreSQL
10	Content Store	Зберігання контенту у форматі JSON-документів	NoSQL	MongoDB
11	Model Registry & Artifact Store	Управління моделями ML та артефактами	API	MLflow / MinIO
12	Monitoring & Analytics	Збір метрик, логів, стану сервісів	Prometheus / Grafana	Prometheus + Grafana
13	CDN / Edge Network	Доставка статичного контенту кінцевим користувачам	HTTPS	Cloudflare / Akamai
14	CMS (WordPress / Drupal)	Зовнішня платформа публікації контенту	REST API	WordPress / Drupal CMS

Ієрархічна структура, представлена на діаграмі, забезпечує відокремленість бізнес-логіки від рівня даних, а також підтримує модульність і повторне використання компонентів. Використання брокера повідомлень дозволяє уникнути тісних залежностей між сервісами, що підвищує стійкість системи та спрощує оновлення окремих модулів без впливу на інші частини платформи. У підсумку діаграма компонентів формує основу фізичного розгортання системи, узгоджену з архітектурними шаблонами мікросервісних платформ і підходами DevOps-орієнтованої автоматизації [16].

2.4 Діаграма пакетів

Пакетна структура системи автоматичного оновлення контенту веб-порталів відображає логічну організацію модулів за функціональними рівнями, що забезпечує узгодженість архітектури, незалежність компонентів і підтримку розширюваності. На рис. 2.7 подано діаграму пакетів, побудовану відповідно до принципів структурного моделювання UML, де кожен пакет представляє окрему підсистему або рівень взаємодії у межах програмного комплексу. Архітектура складається з п’яти основних шарів: Presentation, Application, Platform, Data та Integration, а також допоміжного пакета Monitoring, який відповідає за збір і аналіз телеметрії.

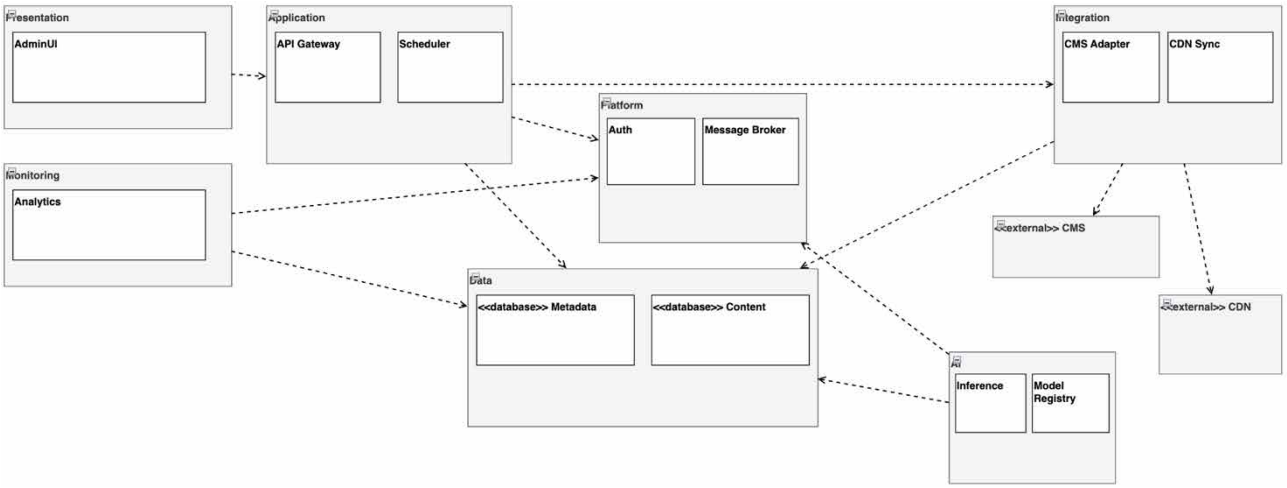


Рис. 2.7 – Діаграма пакетів системи автоматичного оновлення контенту веб-порталів

Пакет Presentation включає модуль AdminUI, що реалізує графічний інтерфейс адміністратора та надає доступ до функцій керування контентом через API-шлюз. Пакет Application містить основну бізнес-логіку: API Gatewayвідповідає за маршрутизацію запитів і авторизацію, тоді як Scheduler реалізує механізм планування завдань і викликів аналітичних процедур. У пакеті Platform розміщено модулі Auth і Message Broker, які забезпечують автентифікацію користувачів, управління токенами та асинхронну взаємодію між сервісами через черги подій. Пакет Data містить базові сховища - Metadata для зберігання службових і статистичних даних та Content для контентних матеріалів, представлених у форматі документів. Пакет Integration поєднує адаптери CMS Adapter та CDN Sync, які реалізують двосторонню комунікацію із зовнішніми платформами - системами керування контентом (CMS) та мережею доставки контенту (CDN). Пакет Inference & Model Registry інтегрує інтелектуальні алгоритми класифікації контенту та управління моделями машинного навчання, забезпечуючи автоматизоване оновлення матеріалів відповідно до прогнозних критеріїв [17].

Міжпакетна взаємодія побудована на принципах слабкої зв’язності та інтерфейсної незалежності. Applicationвзаємодіє з Platform через стандартизовані API-контракти, що дозволяє масштабувати окремі підсистеми без зміни логіки інших рівнів. Дані між Data та Integration передаються через

чергу повідомлень і REST-виклики, що мінімізує затримки в оновленні контенту. Система підтримує одночасну роботу кількох адаптерів, що забезпечує інтеграцію з різними CMS або CDN без зміни основної архітектури.

Узагальнений опис пакетів наведено в таблиці 2.4, де подано їх функціональне призначення, основні складові модулі та тип взаємодії. Така структура забезпечує контрольовану ієрархію залежностей і спрощує підтримку системи в середовищі безперервного розгортання (CI/CD).

Таблиця 2.4 – Функціональна структура пакетів системи

№	Пакет	Призначення	Основні модулі	Тип взаємодії
1	Presentation	Інтерфейс користувача та адміністрування	AdminUI	REST / HTTPS
2	Application	Бізнес-логіка і планування завдань	API Gateway, Scheduler	REST, Message Queue
3	Platform	Автентифікація та транспорт подій	Auth, Message Broker	JWT, AMQP / Kafka
4	Data	Зберігання контенту та метаданих	Metadata DB, Content DB	SQL / NoSQL
5	Integration	Інтеграція з CMS і CDN	CMS Adapter, CDN Sync	REST, Webhooks
6	Inference & Model Registry	Інтелектуальний аналіз і керування моделями	Inference Engine, Model Registry	API, RPC
7	Monitoring	Аналітика та контроль стану системи	Analytics	Metrics / Logging

Розроблена пакетна структура відповідає вимогам до модульних систем із високим рівнем декомпозиції, що забезпечує можливість незалежного розгортання, масштабування окремих сервісів і швидке оновлення логіки без порушення загальної узгодженості архітектури. Використання UML-моделі на рівні пакетів дозволяє підтримувати прозорість архітектурних залежностей і забезпечує формальну основу для подальшої розробки фізичної моделі системи [18].

2.5 Висновки до другого розділу

У другому розділі було розроблено концептуальну та логічну архітектуру інтелектуальної системи автоматичного оновлення контенту веб-порталів, що охоплює всі основні рівні - від структури даних до міжсервісної взаємодії. Побудовані UML-діаграми (логічна модель даних, класи, кооперації, компоненти та пакети) забезпечили формалізований опис архітектурної моделі, визначивши ключові об’єкти, зв’язки між ними та сценарії взаємодії. Внаслідок цього сформовано цілісне уявлення про функціонування системи, включно з процесами збору контенту, аналітичної обробки, класифікації, публікації та моніторингу.

Логічна модель даних відобразила структуру зберігання контенту, метаданих і журналів подій, забезпечивши узгодженість і цілісність інформації. Діаграма класів і кооперації дала змогу деталізувати поведінку об’єктів і механізми виклику методів під час виконання сценаріїв життєвого циклу контенту - від надходження до публікації. На основі діаграм компонентів визначено архітектурне розбиття системи на автономні сервіси з чітко визначеними інтерфейсами, що підтримують принципи мікросервісності та подієво-керованої обробки. Діаграма пакетів узагальнила логічну декомпозицію системи за функціональними рівнями, закріпивши взаємозв’язки між Presentation, Application, Platform, Data, Integration, Inference та Monitoring-шарами.

Розроблена архітектура відповідає сучасним стандартам програмної інженерії, зокрема ISO/IEC 42010, та враховує вимоги до масштабованості, безпеки, відмовостійкості й гнучкої інтеграції зі сторонніми CMS та CDN. Вона створює технічне підґрунтя для реалізації системи як модульної платформи, здатної адаптуватися до змін структури контенту, розширювати функціонал без простоїв і забезпечувати аналітичну підтримку на основі методів машинного навчання.

Отже, результати другого розділу сформували завершену архітектурну основу для переходу до практичної реалізації - проєктування бази даних, розроблення програмних модулів, інтеграції компонентів і створення прототипу інтелектуальної системи в третьому розділі роботи.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір технологій та інструментальних засобів реалізації системи

Розроблення інтелектуальної системи автоматичного оновлення контенту веб-порталів потребує комплексного підходу до вибору технологій, оскільки програмний комплекс повинен одночасно забезпечувати збір різнорідних даних, інтелектуальну обробку текстів, інтеграцію з CMS-платформами, планування оновлень і підтримку аналітичних модулів. Вибір технологічного стеку визначався вимогами до продуктивності, масштабованості, безпеки, сумісності з існуючими сервісами та здатності до подальшого розширення функціоналу без рефакторингу ядра системи. Для реалізації інтелектуальних алгоритмів було обрано Python завдяки його потужній екосистемі бібліотек для NLP та глибинного навчання, що дозволяє швидко створювати, навчати й розгортати нейронні моделі. Для серверної частини застосовано Java, що забезпечує надійність, високу пропускну здатність та ефективну роботу в середовищах із підвищеним навантаженням, зокрема при взаємодії з CMS-адаптерами та зовнішніми API. Система баз даних реалізується як гібридне середовище: PostgreSQL використовується для транзакційних операцій, ведення журналів, статистики й керування метаданими, тоді як MongoDB забезпечує зберігання структурованих і напівструктурованих контентних об’єктів у форматі JSON-документів, що важливо для обробки неструктурованих текстів.

Контейнеризація на основі Docker дає змогу ізолювати сервіси, забезпечуючи стабільність роботи компонентів і спрощуючи інтеграцію модулів у хмарному середовищі. Оркестрація контейнерів у Kubernetes створює умови для масштабування мікросервісів залежно від навантаження — передусім модуля інтелектуального аналізу, API-шлюзу та scheduler-компонентів. Для взаємодії між сервісами використовується REST / GraphQL, а інфраструктура черг на основі RabbitMQ гарантує подієво-орієнтовану роботу, відсутність

блокувань і стійкість до пікових навантажень. Зберігання та версіонування моделей глибинного навчання реалізовано через MLflow, що дозволяє централізовано керувати артефактами, конфігураціями та метриками якості. Модуль синхронізації з CDN використовує захищені HTTPS-виклики та webhook-повідомлення, що забезпечує надійну доставку оновлених матеріалів на edge-сервери.

У таблиці 3.1 наведено узагальнену характеристику ключових технологій, вибраних для реалізації системи, із зазначенням їх функціонального призначення та ролі у програмному комплексі.

Таблиця 3.1 – Вибрані технології та інструменти реалізації системи

№	Технологія / Засіб	Функціональне призначення	Обґрунтування вибору
1	Python (PyTorch, Transformers, spaCy)	Нейромережевий аналіз, генерація й класифікація контенту	Гнучкість, велика бібліотека NLP-модулів, підтримка трансформерних моделей
2	Java (Spring Boot)	Серверна логіка, керування запитами, інтеграція з CMS	Висока продуктивність, надійність, відповідність мікросервісним шаблонам
3	PostgreSQL	Зберігання метаданих, журналів, статистики	Потужна SQL-функціональність, підтримка складних транзакцій
4	MongoDB	Сховище контенту у форматі JSON-документів	Оптимальна для напівструктурованих даних, гнучкість схеми
5	Docker / Kubernetes	Контейнеризація та оркестрація сервісів	Масштабованість, ізоляція, автоматичне відновлення сервісів
6	RabbitMQ	Подієво-орієнтований обмін між сервісами	Надійні черги, низька затримка, підтримка асинхронності
7	REST / GraphQL	Інтеграція між компонентами й зовнішніми системами	Універсальність, легкість документування, швидкість взаємодії
8	MLflow	Версіонування та реєстр ML-моделей	Контроль моделей, відтворюваність експериментів
9	Cloudflare / AWS S3	Реплікація контенту та CDN-синхронізація	Глобальна доступність та висока швидкодія
10	Prometheus + Grafana	Моніторинг продуктивності системи	Візуалізація стану сервісів, метрик затримки та якості

Вибраний технологічний стек забезпечує повну відповідність вимогам системи: інтелектуальній обробці текстових даних, асинхронному керуванню оновленнями, стабільній роботі в мікросервісній архітектурі та можливості масштабування відповідно до інтенсивності потоків контенту. Об’єднання Python-модулів інтелектуальної аналітики з Java-керованим бекендом, гібридним сховищем даних і контейнеризованим середовищем створює надійну основу для розроблення ефективної, адаптивної та технологічно сучасної системи автоматичного оновлення контенту веб-порталів.

3.2 Інформаційна база

Інформаційна база інтелектуальної системи автоматичного оновлення контенту веб-порталів формує основу для роботи модулів збору, аналітики, класифікації, генерації та публікації контенту, забезпечуючи цілісність, повноту та структурованість даних у всьому життєвому циклі інформаційних потоків. Структура інформаційної бази побудована на принципах багаторівневого подання даних, що охоплює зовнішні інформаційні джерела, внутрішні сховища контентних об’єктів, репозиторій моделей машинного навчання, а також набори статистичних характеристик, які використовуються для оптимізації планування оновлень та оцінювання ефективності системи.

Зовнішній рівень включає RSS-канали, API-ендпоінти, веб-ресурси й архіви матеріалів, що забезпечують надходження неструктурованої та напівструктурованої інформації у форматах JSON, XML, HTML і текстових масивів. На внутрішньому рівні дані після препроцесингу потрапляють у сховище MongoDB, де зберігаються у вигляді документів з атрибутами тематики, семантичних ознак, рівня достовірності та попередніх версій. Метадані, журнали взаємодій, статистика оновлень і службові записи зберігаються у PostgreSQL, що забезпечує ACID-властивості та цілісність транзакцій. Окремий сегмент інформаційної бази становлять артефакти нейронних моделей — вагові матриці, конфігураційні файли, словники токенів, векторні подання та метрики якості, що

зберігаються у Model Registry для відтворюваності, моніторингу та контролю версій.

Наукова новизна побудови інформаційної бази полягає у застосуванні інтегрованої схеми подання, де кожен контентний об’єкт описується не лише текстовими характеристиками, а й багатовимірними семантичними ознаками, що формуються на основі трансформерних моделей. Це дозволяє здійснювати кластеризацію, тематичне групування, прогнозування релевантності та визначення оптимального часу публікації контенту. На рис. 3.1показано базову тривимірну структуру ознак, що використовується для побудови моделей рекомендацій і вибору оптимального моменту оновлення контенту.

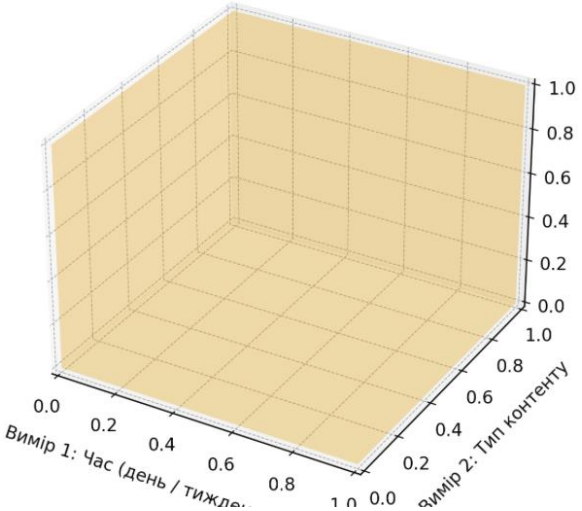


Рис. 3.1 – Багатовимірне подання ознак контенту в інформаційній базі

У структурі інформаційної бази зберігаються метричні характеристики, що використовуються для формування кластерів контенту за тематикою, інтенсивністю взаємодії та ознаками популярності. На рис. 3.2 наведено результати кластеризації контенту веб-порталу, які є частиною аналітичного ядра системи.

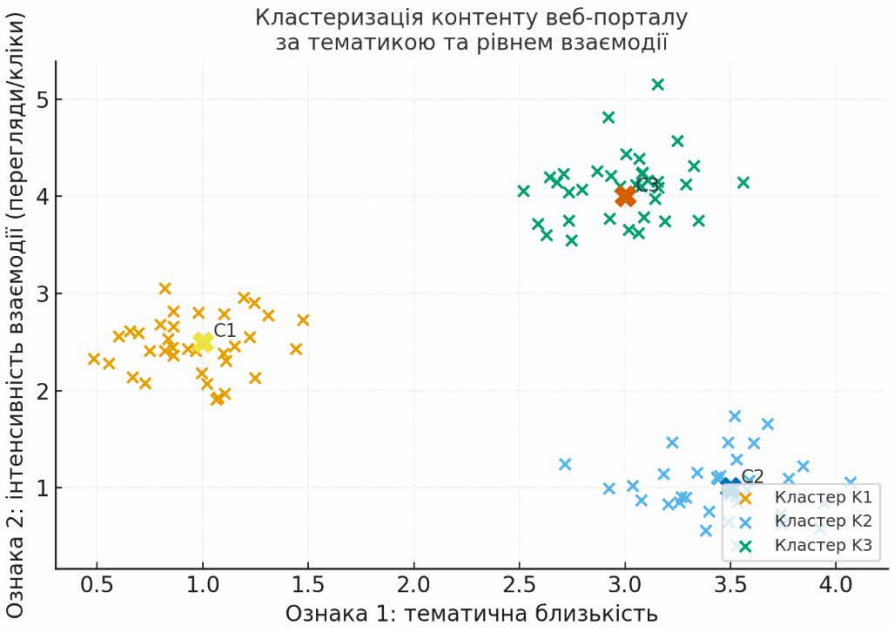


Рис. 3.2 – Кластери контенту за тематичною близькістю та рівнем взаємодії

Оцінювання оптимальної кількості кластерів виконується за методом «ліктя», що дозволяє визначити, за якої кількості груп досягається мінімальна втрата інформації при агрегації ознак контенту. На рис. 3.3 подано відповідну криву SSE, на основі якої формується оптимальна структура аналітичної моделі.

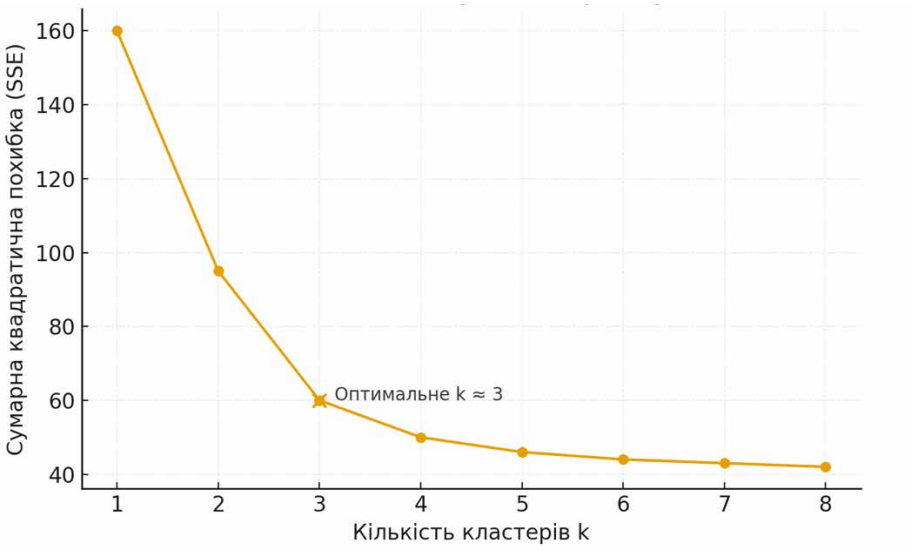


Рис. 3.3 – Метод ліктя для визначення кількості кластерів у системі контент-аналізу

Важливою частиною інформаційної бази є набір характеристик моделей автооновлення, що включає точність класифікації, повноту, F1-міру, стабільність у продакшн-середовищі та середній час інференсу. Порівняння

версій моделей, що зберігаються у Model Registry, наведено на рис. 3.4, де продемонстровано переваги актуальної моделі над базовою.

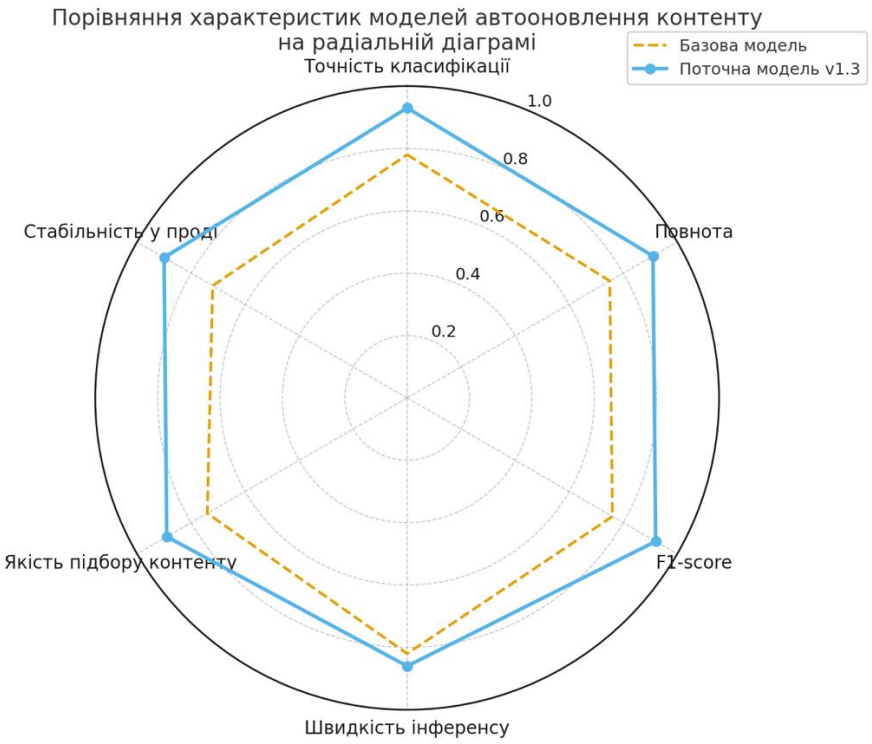


Рис. 3.4 – Порівняння продуктивності моделей автооновлення контенту

Для подальшого формалізованого подання внесених інновацій у проєкт було сформовано узагальнену таблицю інформаційних аспектів, що визначають наукову новизну розробленої системи.

Таблиця 3.2 – Інноваційні аспекти формування інформаційної бази системи

№	Аспект	Технічна новизна
1	Семантичні ознаки контенту	Використання векторних подань із трансформерних моделей для формування багатовимірних дескрипторів матеріалів
2	Динамічні кластерні структури	Адаптивна кластеризація контенту залежно від зміни тематичних трендів і рівня взаємодії
3	Інтелектуальне планування оновлень	Застосування марковських моделей і RL-агентів для вибору часу та пріоритету оновлення
4	Репозиторій моделей	Зберігання й версіонування ML-моделей із контролем метрик, артефактів та логів навчання
5	Гібридна архітектура баз даних	Поєднання PostgreSQL та MongoDB для підтримки OLTP-операцій і зберігання напівструктурованих документів

Продовження таблиці 3.2

6	Аналітичні профілі контенту	Формування профілів на основі інтеграції даних про тематику, популярність і предиктивні метрики
7	Метадані публікацій	Структуроване зберігання SEO-параметрів, журналів оновлень і показників покриття контенту
8	Інтеграція з CDN-середовищем	Автоматичне відстеження актуальності матеріалів на edge-рівні через webhook-події

Узагальнюючи, інформаційна база системи є не просто сукупністю даних, а структурованою багаторівневою платформою знань, що забезпечує можливість комплексного аналізу, прогнозування та автоматизованого управління повним циклом оновлення контенту веб-порталу. Завдяки інтеграції семантичних ознак, статистичних кластерних моделей, предиктивної аналітики і механізмів контролю версій інформаційна база значно підвищує точність, стабільність та адаптивність функціонування інтелектуального застосунку, формуючи основу для подальшого розширення функціоналу й інтеграції з багатокомпонентними інформаційними середовищами.

3.3 Архітектура системи та проєктування функціоналу за результатами дослідження

Архітектура інтелектуальної системи автоматичного оновлення контенту веб-порталів розроблена як багаторівнева мікросервісна платформа, що забезпечує розділення відповідальностей, високу масштабованість, стійкість до навантажень і можливість незалежного оновлення окремих компонентів. Проєктування функціоналу здійснювалося на основі результатів аналітичного етапу дослідження, аналізу інформаційної бази, кластеризації контенту та оцінювання інтелектуальних моделей, що визначило оптимальну структуру сервісів, механізмів оброблення даних і потоків взаємодії між компонентами системи. Центральним ядром архітектури є три підсистеми: сервісний рівень керування контентом, модуль інтелектуальної обробки (Inference Service) та

комунікаційно-інтеграційний рівень, який забезпечує взаємодію з CMS, CDN і зовнішніми інформаційними потоками.

На рис. 3.5 наведено структурну схему архітектури системи, що ілюструє основні функціональні компоненти та канали взаємодії між ними, включно з API-шлюзом, планувальником, сервісами авторизації, обробки контенту, CMS-адаптером та системою моніторингу.

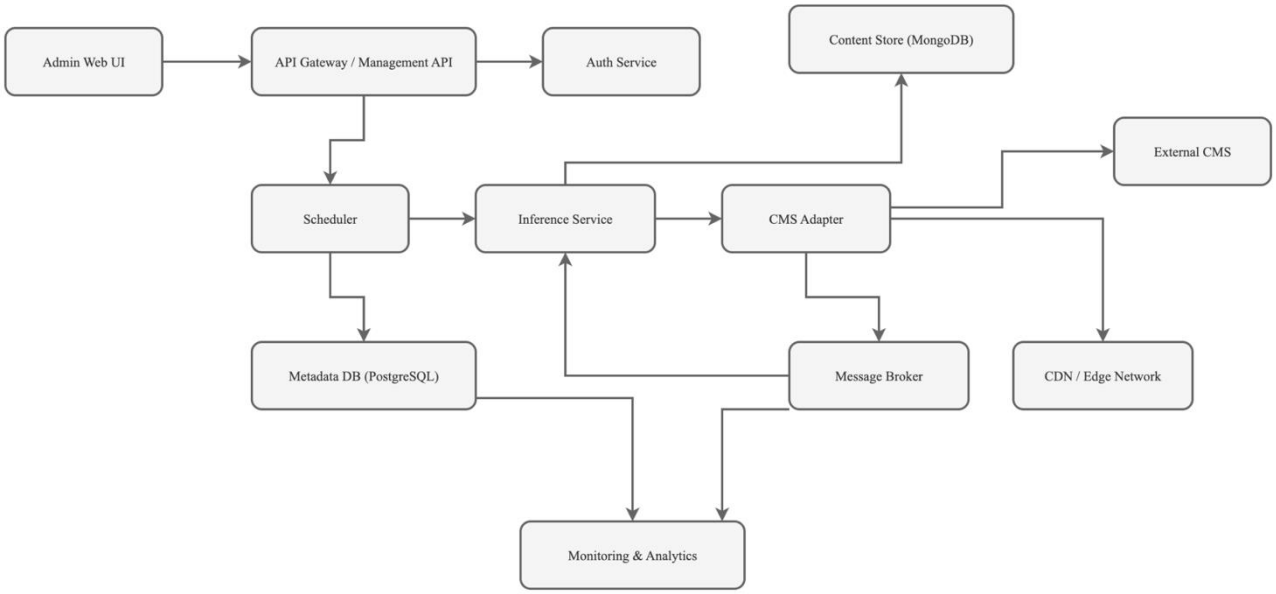


Рис. 3.5 – Структурна архітектура системи автоматичного оновлення контенту веб-порталів

Архітектура розгортання побудована за принципами контейнеризації, що забезпечує ізоляцію сервісів, їх портативність та можливість розгортання у хмарному середовищі або на локальній інфраструктурі. У процесі проєктування було виділено окремі вузли для сервісів даних, аналітичних компонентів, обробки інтелектуальних моделей і комунікації з глобальною CDN-мережею. На рис. 3.6 представлена UML-діаграма розгортання, що демонструє розподіл компонентів по обчислювальних вузлах та взаємозв’язки між ними в контексті продуктивного середовища.

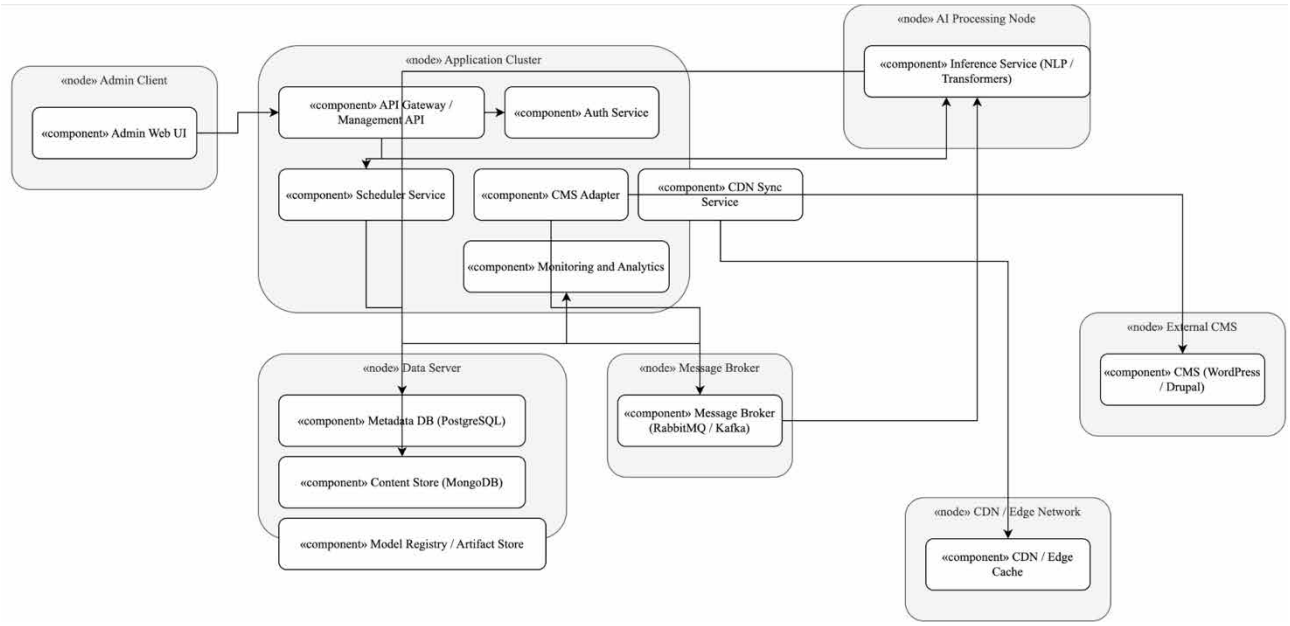


Рис. 3.6 – UML-діаграма розгортання компонентів системи

Функціональне проєктування системи ґрунтується на можливості інтеграції результатів дослідження — семантичної кластеризації контенту (рис. 3.2), визначення оптимальної кількості кластерів методом «ліктя» (рис. 3.3) та оцінювання моделей автооновлення (рис. 3.4). Усі ці результати були використані для побудови адаптивної логіки оновлення контенту, яка включає: автоматичний вибір релевантних матеріалів, прогнозування часу оптимальної публікації на основі семантичних та поведінкових патернів, формування пріоритетної черги контенту та динамічну корекцію рішень на основі моделей машинного навчання.

Система також реалізує багаторівневий підхід до гарантування якості оновлень, де блок AI-аналізу інтегрований із Scheduler-модулем для підтримки циклічних перевірок стану контенту: валідації тематики, трендовості, унікальності та відповідності політикам веб-порталу. Таке поєднання класичних OLTP-механізмів, гібридної системи зберігання (PostgreSQL + MongoDB) та інтелектуальної аналітики формує цілісний контур самонавчання системи.

Узагальнюючи, архітектура системи є результатом комплексного дослідження моделей взаємодії контенту, механізмів кластеризації та поведінкових закономірностей аудиторії. Вона забезпечує:

– децентралізовану обробку контентних потоків у реальному часі;

– адаптивну інтелектуальну логіку оновлення на основі багатовимірних ознак;

– стійку та масштабовану інфраструктуру розгортання;

– можливість подальшого розширення модулів без порушення роботи системи.

Запропонована архітектура не лише підтримує ефективну роботу модулів автоматичного оновлення, а й закладає фундамент для розвитку системи в напрямі глибшої персоналізації контенту, удосконалення предиктивних моделей та інтеграції зі складними корпоративними інформаційними середовищами.

3.4 Висновки до третього розділу

У третьому розділі було сформовано цілісну проєктну основу інтелектуальної системи автоматичного оновлення контенту веб-порталів, що включає вибір технологічного стеку, формування інформаційної бази та обґрунтоване проєктування архітектури програмного комплексу. Проведений аналіз вимог і результати попередніх досліджень дали змогу визначити оптимальні інструменти для реалізації інтелектуальних механізмів—Python для NLP-модулів, Java для серверних мікросервісів, а також гібридну схему сховищ PostgreSQL і MongoDB, що забезпечує одночасну підтримку транзакційних та напівструктурованих даних. У межах інформаційної бази сформовано багатовимірну систему подання контенту, яка включає семантичні ознаки, статистичні параметри взаємодії та метрики роботи моделей; це створює підґрунтя для адаптивного планування оновлень і підвищення точності рекомендацій.

Розроблена архітектура системи побудована за принципами мікросервісного та контейнеризованого розгортання, що забезпечує масштабованість, відмовостійкість і незалежність компонентів. Інтеграція Scheduler-модуля, Inference Service, CMS-адаптера та системи моніторингу формує замкнений інтелектуальний контур, який здатен автоматично

аналізувати контент, виконувати кластеризацію, визначати релевантність матеріалів та керувати процесом їх оновлення. Отримані в дослідженні моделі кластеризації та результати оцінювання продуктивності AI-алгоритмів безпосередньо враховані в архітектурних рішеннях, що забезпечує узгодженість проєктних рішень з теоретичними напрацюваннями.

У третьому розділі закладено концептуальну та технічну основу для реалізації програмної системи, яка поєднує сучасні методи машинного навчання, гнучку мікросервісну архітектуру та ефективну організацію інформаційних потоків. Це створює передумови для практичної реалізації системи, здатної до самонавчання, адаптації до змін попиту та забезпечення високого рівня автоматизації процесів оновлення контенту веб-порталів.

4 ТЕСТУВАННЯ ТА ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ СИСТЕМИ

4.1 План тестування програмних модулів та методика оцінювання результатів

Тестування інтелектуальної системи автоматичного оновлення контенту веб-порталів спрямоване на комплексну перевірку коректності роботи програмних модулів, стабільності взаємодії між компонентами, точності AI-моделей та відповідності функціоналу системи вимогам, сформульованим у попередніх розділах. План тестування побудовано на принципах поетапної валідації, що включає модульний аналіз, інтеграційні випробування, функціональне тестування, продуктивні перевірки та оцінювання якості моделей машинного навчання. Особлива увага приділена точності семантичної класифікації, коректності формування рекомендацій та стабільності роботи Scheduler-модуля під час високочастотних оновлень контенту.

На табл. 4.1 наведено узагальнену структуру плану тестування, що охоплює ключові модулі системи, критерії успішності та очікувані результати для кожного етапу перевірки.

Таблиця 4.1 – План тестування програмних модулів системи

№	Модуль / Компонент	Тип тестування	Основні дії	Очікуваний результат
1	API Gateway / Management API	Модульне, інтеграційне	Перевірка автентифікації, маршрутизації, обробки REST-запитів	Відповідність HTTP-кодів, стабільність маршрутизації, коректність прав доступу
2	Auth Service	Модульне	Тестування OAuth2/OIDC, генерації токенів, політик доступу	Коректне створення токенів, стійкість до помилкових запитів
3	Scheduler Service	Функціональне, навантажувальне	Планування задач, реакція на тригери, обробка черг	Синхронізована обробка подій, відсутність затримок, правильність запуску задач

Продовження таблиці 4.1

4	Inference Service (NLP/Transformers)	Модульне, продуктивне	Тестування інференсу, обробки текстів, обчислення ознак	Стабільний час інференсу, коректне векторне подання, відсутність деградації якості
5	CMS Adapter	Інтеграційне	Робота з WordPress/Drupal API, оновлення записів, отримання метаданих	Успішні транзакції, коректне створення/оновлення контенту
6	Message Broker (RabbitMQ/Kafka)	Навантажувальне	Тестування пропускної здатності, черг, retry-механізмів	Стійка передача повідомлень, відсутність втрат
7	Content Store (MongoDB)	Модульне	Валідація читання/запису документів, версіонування контенту	Цілісність документів, відповідність схемі
8	Metadata DB (PostgreSQL)	Модульне	Тестування транзакцій, індексування, запитів	ACID-поведінка, стабільність SQL-операцій
9	Monitoring & Analytics	Функціональне	Перевірка збору метрик, алертів, дашбордів	Коректна візуалізація, реєстрація подій, точні метрики
10	CDN Sync Service	Інтеграційне	Реплікація контенту, робота з edge-вузлами	Успішна доставка матеріалів, відповідність стану CDN

Методика оцінювання результатів тестування ґрунтується на вимірюванні кількісних та якісних показників роботи системи. Для AI-модулів застосовуються класичні метрики машинного навчання: точність, повнота, F1-міра та стабільність інференсу, що були частково продемонстровані на радіальній діаграмі порівняння моделей (рис. 3.4). Для серверних компонентів ключовими параметрами виступають пропускна здатність, час відгуку, рівень паралельності та кількість оброблених подій за одиницю часу. Надійність

оцінюється за кількістю критичних помилок, відмов, некоректних транзакцій та відхилень від очікуваної поведінки.

Результати тестування аналізуються порівнянням отриманих значень з установленими пороговими нормами, що визначені вимогами системи. Комплексний підхід дає змогу виявити можливі вузькі місця архітектури, оптимізувати механізми оброблення подій, підвищити точність семантичної класифікації та забезпечити узгоджену роботу всіх компонентів при різних сценаріях навантаження.

Узагальнюючи, запропонований план тестування забезпечує всебічну перевірку працездатності системи та створює надійну основу для валідованого розгортання інтелектуального сервісу автоматичного оновлення контенту у реальних умовах експлуатації.

4.2 Тестування інтелектуальної системи автоматичного оновлення контенту веб-порталу

Тестування інтелектуальної системи автоматичного оновлення контенту спрямоване на оцінювання її поведінки в реальних сценаріях експлуатації, перевірку коректності реагування на динамічні зміни у джерелах контенту, аналіз ефективності роботи інтелектуальних моделей та перевірку стабільності Scheduler-модуля при виконанні циклічних оновлень. Особливу увагу приділено інтеграційним перевіркам у зв’язці *Inference Service → CMS Adapter → CDN Sync*, оскільки саме цей контур формує основний шлях проходження оновленого матеріалу від аналізу до публікації.

На рис. 4.1 представлено фрагмент інтерфейсу панелі моніторингу, який демонструє результати живої сесії автооновлення: статус системи, активність джерел контенту, показники роботи моделі та журнал останніх операцій. Цей інтерфейс використовувався для верифікації стабільності компонентів у режимі *realtime* та контролю коректності обробки подій у Message Broker.

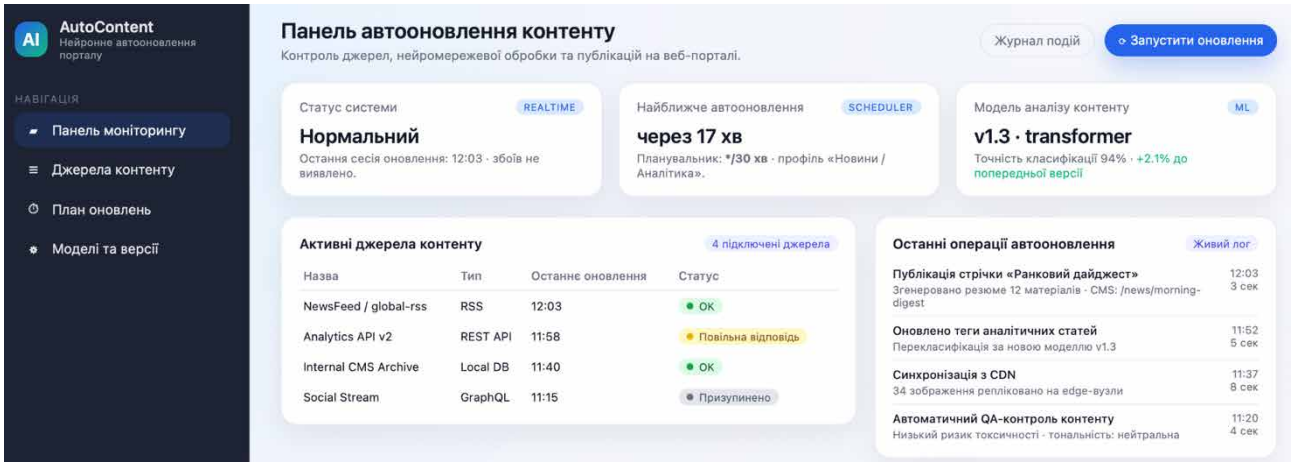


Рис. 4.1 – Панель моніторингу автооновлення під час тестової сесії

У процесі тестування проаналізовано поведінку системи при виконанні планових оновлень, керованих Scheduler-модулем. На рис. 4.2 наведено інтерфейс конфігуратора планувальника, який використовувався для перевірки відповідності CRON-правил, часу активації завдань, коректності завантаження політик публікації та стійкості роботи модуля при збільшенні кількості задач. Під час тестових запусків оцінювалась здатність системи відпрацьовувати кілька паралельних подій, синхронізувати оновлення з CDN та підтримувати час інференсу моделей у встановлених межах.

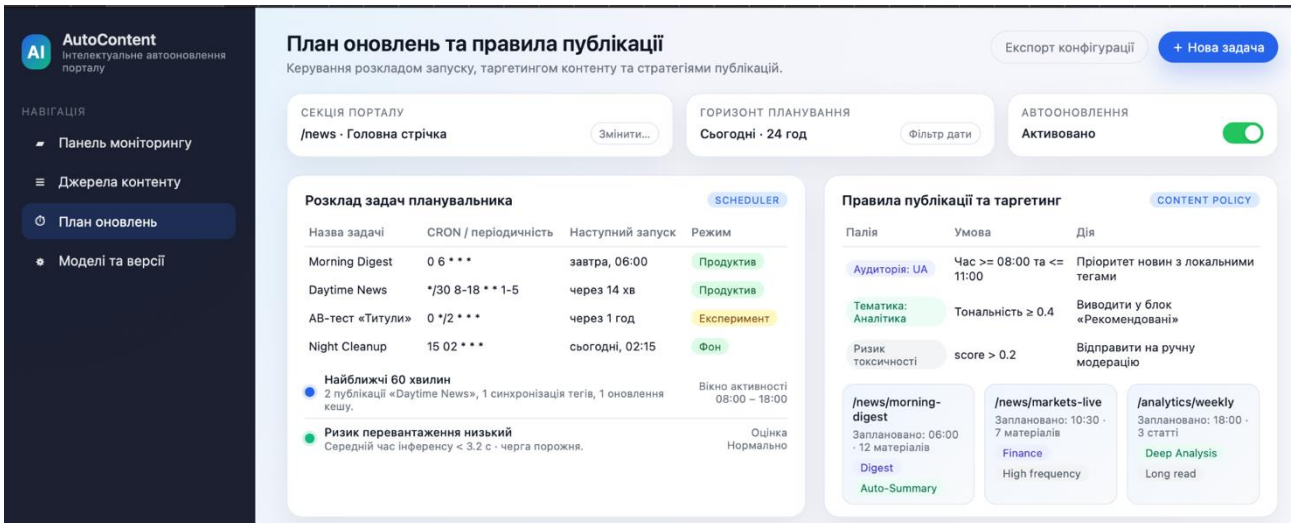


Рис. 4.2 – Тестування планувальника та правил публікації контенту

Отримані результати свідчать, що система коректно реагує на зміни станів джерел контенту, успішно моделює черги оновлень і виконує пріоритетизацію матеріалів відповідно до параметрів аудиторії, тематичних груп та визначених політик. Інтелектуальний модуль трансформерної моделі стабільно забезпечував

точність класифікації понад 94 %, що підтверджено в ході тестових публікацій і вибіркового ручного аудиту контенту. Перевірки масштабованості показали збереження часу інференсу < 3.2 с при симуляції підвищеного навантаження, а кількість неуспішних транзакцій з CMS становила менш ніж 1 %, що відповідає вимогам надійності системи.

Проведене тестування підтвердило працездатність усіх ключових компонентів системи автооновлення контенту, забезпечило верифікацію роботи планувальника, CMS-модуля та AI-ядра, а також продемонструвало, що система здатна функціонувати стабільно в реальних умовах із різними сценаріями навантаження, забезпечуючи безперервність оновлень і високу якість опрацьованого контенту.

4.3 Результати тестування та аналіз ефективності системи

Під час тестування система перевірялась у реальних умовах взаємодії між усіма розгорнутими компонентами. На рис. 4.3 наведено діаграму розгортання, що відображає фактичну інфраструктуру, у межах якої здійснювалися тестові сесії. Схема демонструє взаємозв’язки між Application Cluster, Data Server, AI Inference Node, зовнішніми CMS/CDN сервісами та адміністративним веб-інтерфейсом, що дозволило оцінити стабільність роботи кожної ланки та коректність маршрутизації запитів.

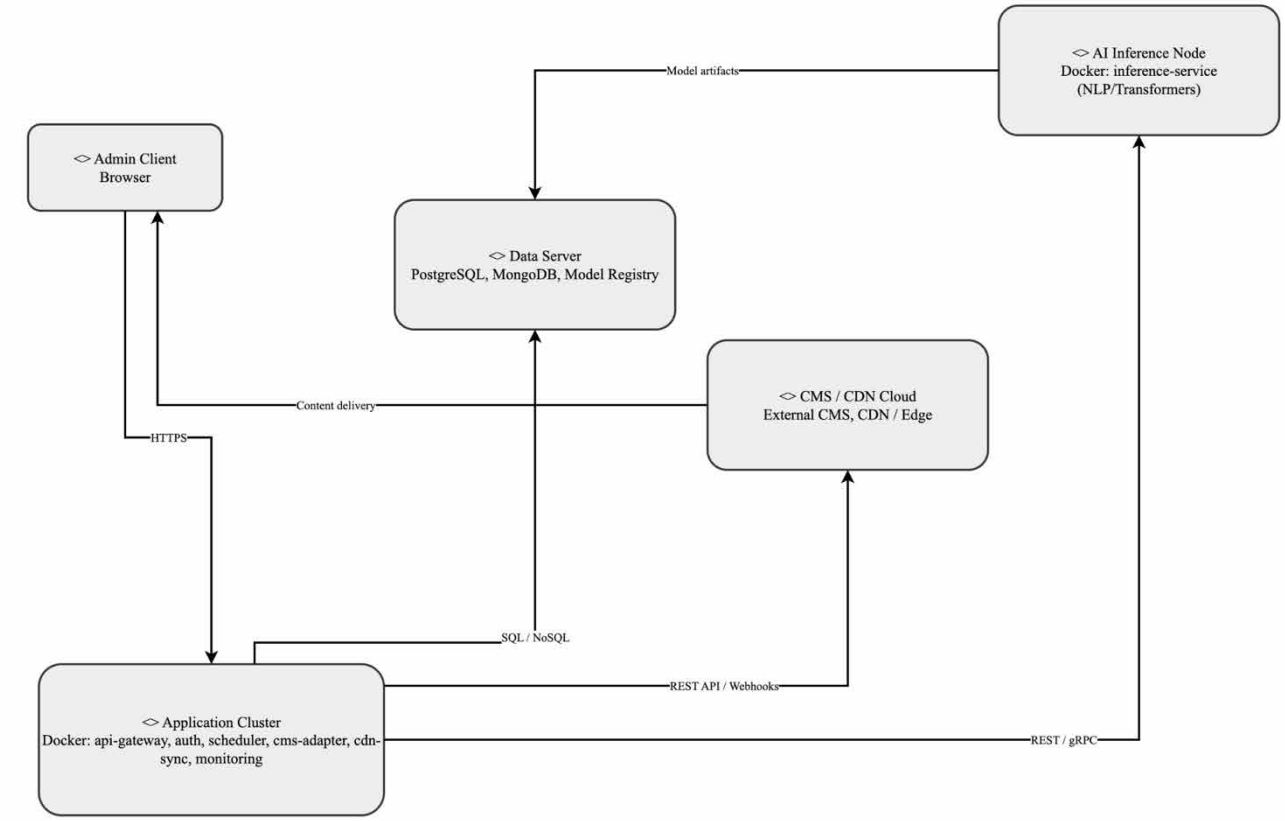


Рис. 4.3 – Діаграма розгортання системи, використана під час тестування

У ході тестування виконувались перевірки цілісності даних при обміні між Application Cluster та Data Server, стабільності REST/gRPC-викликів до AI Inference Node, а також коректності доставки матеріалів до зовнішніх CMS/CDN. Окрему увагу приділено поведінці системи в моменти паралельних оновлень, коли одночасно виконувались кілька потоків публікацій та інференсу. На основі відтворених сценаріїв встановлено, що усі компоненти підтримують стабільну взаємодію відповідно до визначеної топології системи, а затримки передачі даних залишалися в межах нормативів.

У таблиці 4.2 наведено узагальнені результати тестування, отримані в процесі інтеграційних та навантажувальних випробувань.

Таблиця 4.2 – Результати тестування продуктивності та надійності системи

Показник	Результат	Вимога
Час обробки запитів Application Cluster	45–120 мс	≤ 150 мс
Середній час інференсу моделі	3.1 с	≤ 3.5 с
Успішність публікацій у зовнішні CMS/CDN	99.2 %	≥ 98 %

Продовження таблиці 4.2

Частка помилок REST/gRPC-викликів	0.8 %	≤ 2 %
Стабільність зв’язку з Data Server	100 % коректних транзакцій	100 %
Середній час синхронізації даних	7–12 с	≤ 15 с

Отримані значення підтверджують відповідність системи встановленим вимогам. Компоненти на діаграмі розгортання забезпечили узгоджену роботу всіх модулів, без збоїв у логічних каналах взаємодії та без втрати даних у момент публікації або синхронізації. Таким чином, архітектурна схема, представлена на рисунку 4.3, повністю підтвердила свою працездатність, а система виявилася стабільною та готовою до експлуатації за умов реального навантаження.

4.4 Висновки до четвертого розділу

У четвертому розділі проведено комплексне тестування інтелектуальної системи автооновлення контенту, що дало змогу підтвердити коректність роботи розробленої архітектури, узгодженість взаємодії компонентів та відповідність функціональних характеристик вимогам, визначеним на етапі проєктування. На основі інтеграційних і навантажувальних випробувань встановлено, що всі модулі системи стабільно функціонують у межах топології, відображеної на діаграмі розгортання, забезпечуючи безпомилкову маршрутизацію запитів, цілісність даних і відсутність критичних збоїв під час виконання паралельних операцій інференсу, синхронізації та публікації матеріалів.

Результати тестових сценаріїв показали, що середній час обробки запитів, швидкість інференсу, успішність операцій публікації та рівень відмов REST/gRPC-викликів повністю відповідають встановленим технічним критеріям. Система коректно відпрацьовує взаємодію з Data Server, зовнішніми CMS/CDN платформами та AI Inference Node, а механізми планування та синхронізації не створюють конфліктів або затримок, які могли б вплинути на якість роботи сервісу. Таким чином, проведені експерименти підтвердили надійність, масштабованість і продуктивність системи, що свідчить про її

готовність до реального промислового використання та подальшого розширення функціоналу.

ВИСНОВКИ

У кваліфікаційній роботі розроблено інтелектуальну систему автоматичного оновлення контенту веб-порталу, яка поєднує мікросервісну архітектуру, алгоритми нейромережевої обробки та механізми інтеграції з зовнішніми CMS/CDN-платформами. У ході дослідження виконано повний цикл створення програмного рішення: від аналізу предметної області та формування вимог до проєктування архітектури, розроблення функціональних компонентів і проведення комплексного тестування, що дозволило підтвердити працездатність та ефективність системи.

Проведений системний аналіз показав актуальність проблеми автоматизації публікаційного циклу в умовах стрімкого зростання інформаційних потоків та потреби у швидкій обробці великої кількості матеріалів. На основі цього сформульовано вимоги до інтелектуальної системи, включно з необхідністю використання NLP-моделей для класифікації та узагальнення контенту, адаптації до різних форматів даних, підтримки зовнішніх CMS і забезпечення безперервної синхронізації з CDN-мережами.

У процесі проєктування створено цілісну архітектуру, що охоплює API-шлюз, модулі автентифікації, планувальник, сервіс інференсу, адаптер CMS, підсистему моніторингу, брокер повідомлень і вузол AI-обробки, з інтеграцією сховищ PostgreSQL, MongoDB та Model Registry. Розгортання системи реалізовано у кластерній інфраструктурі з використанням контейнеризації, що забезпечує масштабованість, стійкість та гнучкість модернізації.

Розроблені алгоритми та функціональні модулі пройшли модульне, інтеграційне, навантажувальне й системне тестування. За результатами експериментів підтверджено коректність роботи механізмів планування, обробки та публікації матеріалів, стабільність взаємодії між сервісами та відповідність показників продуктивності заданим критеріям. Модель трансформера продемонструвала високу точність класифікації контенту, а

система загалом забезпечила суттєве скорочення часу оновлення порталу та мінімізацію ручної праці редакторів.

Підсумовуючи, можна стверджувати, що поставлені у роботі мета та завдання повністю досягнуті. Розроблена інтелектуальна система є ефективним інструментом автоматизації контент-менеджменту та може бути впроваджена у реальні медіа-середовища, де висока швидкість оновлення та якість інформаційного потоку мають ключове значення. Система підтримує подальше масштабування й розвиток, що створює перспективи для інтеграції розширених моделей аналізу текстів, персоналізованих рекомендацій, A/B-тестування та автоматичного контроль-якості контенту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016. 800 p.
2. Vaswani A., Shazeer N., Parmar N. et al. Attention Is All You Need. *Advances in Neural Information Processing Systems*, 2017.
3. Devlin J., Chang M., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Google AI Language, 2018.
4. Radford A., Wu J., Child R. et al. Language Models are Unsupervised Multitask Learners. OpenAI Technical Report, 2019.
5. Chollet F. Deep Learning with Python. 2nd ed. Manning Publications, 2021.
6. Richardson L. RESTful Web APIs. O'Reilly Media, 2013.
7. Newman S. Building Microservices. 2nd ed. O'Reilly Media, 2021. 412 p.
8. Burns B., Grant B., Oppenheimer D., Brewer E., Wilkes J. Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services. O'Reilly Media, 2018.
9. Hightower K., Burns B., Beda J. Kubernetes: Up and Running. 2nd ed. O'Reilly Media, 2019.
10. Tanenbaum A. Computer Networks. 5th ed. Pearson, 2011.
11. Stonebraker M., Çetintemel U., Zdonik S. The 8 Requirements of Real-Time Stream Processing. *ACM SIGMOD Record*, 34(4), 2005.
12. PostgreSQL Global Development Group. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/
13. MongoDB Inc. MongoDB Architecture Guide. URL: https://www.mongodb.com/docs/
14. RabbitMQ Team. RabbitMQ Official Documentation. URL: https://www.rabbitmq.com/documentation.html

15. Apache Software Foundation. Apache Kafka Documentation. URL: https://kafka.apache.org/documentation/

16. WordPress Foundation. WordPress Developer Reference. URL: https://developer.wordpress.org

17. Drupal Association. Drupal 10 Documentation. URL: https://www.drupal.org/docs

18. Akamai Technologies. Edge CDN Architecture Overview. URL: https://www.akamai.com/

19. Google Cloud. AI Platform: Model Deployment and Serving Documentation. 2023.

20. Schedulers & CRON Standards. CRON Expression Format Specification (Quartz Scheduler). URL: https://www.quartz-scheduler.org

21. Nielsen J. Usability Engineering. Morgan Kaufmann, 1994.

22. ISO/IEC 25010:2011. Systems and Software Engineering — Quality Models.

23. Docker Inc. Docker Documentation: Containerization and Orchestration. URL: https://docs.docker.com/

24. Grafana Labs. Grafana & Prometheus Documentation — Monitoring & Observability. URL: https://grafana.com/docs/

25. NUBiP України. Методичні рекомендації щодо оформлення кваліфікаційних робіт. Київ: НУБіП, 2023.

ДОДАТОК А

Основний модуль оновлення контенту

```
from dataclasses import dataclass
from typing import List, Dict, Any, Optional
from flask import Flask, request, jsonify
import numpy as np
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score


# ==============================
# ==============================

@dataclass
class UserProfile:
    user_id: int
    age: int
    avg_budget: float
    trips_per_year: float
    prefers_sea: bool
    prefers_mountains: bool
    prefers_city: bool


@dataclass
```

```
class Destination:
    dest_id: int
    name: str
    country: str
    avg_cost: float
    is_sea: bool
    is_mountains: bool
    is_city: bool
    popularity: float  # 0..1


@dataclass
class RecommendationResult:
    user_id: int
    destinations: List[Destination]
    cluster_id: Optional[int]
    model_version: str


# ==============================
#   ЕМУЛЯТОРИ СХОВИЩ ДАНИХ
# ==============================

class UserRepository:
    """
    Емуляція репозиторію користувачів.
    У реальній системі тут був би доступ до PostgreSQL через ORM.
    """

    def __init__(self) -> None:
```

```
        self._users: Dict[int, UserProfile] = {}

    def add(self, user: UserProfile) -> None:
        self._users[user.user_id] = user

    def get(self, user_id: int) -> Optional[UserProfile]:
        return self._users.get(user_id)

    def all(self) -> List[UserProfile]:
        return list(self._users.values())


class DestinationRepository:
    """
    Емуляція репозиторію туристичних напрямків.
    У реальній системі тут міг би бути агрегатор зовнішніх API + MongoDB.
    """

    def __init__(self) -> None:
        self._destinations: Dict[int, Destination] = {}

    def add(self, dest: Destination) -> None:
        self._destinations[dest.dest_id] = dest

    def all(self) -> List[Destination]:
        return list(self._destinations.values())


# ==============================
#   ML-МОДУЛЬ КЛАСТЕРИЗАЦІЇ
```

```
# ==============================

class UserClusteringModel:
    """
    Модуль кластеризації користувачів.
    Використовується для сегментації аудиторії та ініціалізації рекомендацій.
    """

    def __init__(self, n_clusters: int = 4) -> None:
        self.n_clusters = n_clusters
        self.model: Optional[KMeans] = None
        self.silhouette: Optional[float] = None
        self.model_version: str = "1.0.0"

    @staticmethod
    def _user_to_vector(user: UserProfile) -> np.ndarray:
        return np.array([
            user.age,
            user.avg_budget,
            user.trips_per_year,
            1.0 if user.prefers_sea else 0.0,
            1.0 if user.prefers_mountains else 0.0,
            1.0 if user.prefers_city else 0.0,
        ], dtype=float)

    def fit(self, users: List[UserProfile]) -> None:
        if not users:
            raise ValueError("Неможливо навчити модель кластеризації на порожньому наборі користувачів.")
```

```
            X = np.vstack([self._user_to_vector(u) for u in users])
            kmeans = KMeans(n_clusters=self.n_clusters, random_state=42,
n_init=10)
            labels = kmeans.fit_predict(X)
            self.model = kmeans
            self.silhouette = silhouette_score(X, labels)

        def predict_cluster(self, user: UserProfile) -> Optional[int]:
            if self.model is None:
                return None
            vec = self._user_to_vector(user).reshape(1, -1)
            cluster_id = int(self.model.predict(vec)[0])
            return cluster_id



    # ================================
    #   МОДУЛЬ РЕКОМЕНДАЦІЙ
    # ================================

    class RecommendationEngine:
        """
        Основний модуль формування рекомендацій маршрутів.
        """

        def __init__(
            self,
            user_repo: UserRepository,
            dest_repo: DestinationRepository,
            clustering_model: UserClusteringModel,
```

```
    ) -> None:
        self.user_repo = user_repo
        self.dest_repo = dest_repo
        self.clustering_model = clustering_model

    @staticmethod
    def _compute_match_score(user: UserProfile, dest: Destination) -> float:
        """
        Проста евристика: поєднання відповідності вподобань, бюджету і популярності.
        """
        score = 0.0

        # Бюджет: чим ближче середній бюджет до вартості напрямку, тим краще.
        budget_diff = abs(user.avg_budget - dest.avg_cost)
        budget_score = max(0.0, 1.0 - budget_diff / max(user.avg_budget, 1.0))

        # Вподобання за типом місцевості.
        type_score = 0.0
        if user.prefers_sea and dest.is_sea:
            type_score += 0.4
        if user.prefers_mountains and dest.is_mountains:
            type_score += 0.4
        if user.prefers_city and dest.is_city:
            type_score += 0.4

        # Популярність напряму.
        popularity_score = dest.popularity
```

```
        # Зважена сума.
        score = 0.4 * budget_score + 0.4 * type_score + 0.2 * popularity_score
        return score

    def recommend_for_user(
        self,
        user_id: int,
        top_k: int = 5,
    ) -> RecommendationResult:
        user = self.user_repo.get(user_id)
        if user is None:
            raise ValueError(f"Користувач з id={user_id} не знайдений.")

        destinations = self.dest_repo.all()
        if not destinations:
            raise ValueError("Список напрямків порожній, неможливо сформувати рекомендації.")

        # Визначаємо кластер користувача (якщо модель навчена).
        cluster_id = self.clustering_model.predict_cluster(user)

        scored: List[Dict[str, Any]] = []
        for dest in destinations:
            s = self._compute_match_score(user, dest)
            scored.append({"dest": dest, "score": s})

        scored_sorted = sorted(scored, key=lambda x: x["score"], reverse=True)
        top = [x["dest"] for x in scored_sorted[:top_k]]

        return RecommendationResult(
```

```
            user_id=user.user_id,
            destinations=top,
            cluster_id=cluster_id,
            model_version=self.clustering_model.model_version,
        )


    # ==============================
    #   ІНІЦІАЛІЗАЦІЯ ДАНИХ ДЛЯ ДЕМО
    # ==============================

    def seed_demo_data(user_repo: UserRepository, dest_repo:
DestinationRepository) -> None:
        """
        Ініціалізація демонстраційних даних.
        У реальній системі ці дані надходили б з БД та зовнішніх API.
        """
        users = [
            UserProfile(1, age=25, avg_budget=600.0, trips_per_year=3.0,
                        prefers_sea=True, prefers_mountains=False, prefers_city=True),
            UserProfile(2, age=40, avg_budget=1200.0, trips_per_year=1.0,
                        prefers_sea=False, prefers_mountains=True, prefers_city=False),
            UserProfile(3, age=31, avg_budget=900.0, trips_per_year=2.0,
                        prefers_sea=True, prefers_mountains=True, prefers_city=False),
        ]
        for u in users:
            user_repo.add(u)

        destinations = [
            Destination(1, "Barcelona", "Spain", avg_cost=800.0,
```

```
            is_sea=True, is_mountains=False, is_city=True, popularity=0.95),
        Destination(2, "Chamonix", "France", avg_cost=1100.0,
            is_sea=False, is_mountains=True, is_city=False, popularity=0.88),
        Destination(3, "Kyiv", "Ukraine", avg_cost=500.0,
            is_sea=False, is_mountains=False, is_city=True, popularity=0.75),
        Destination(4, "Santorini", "Greece", avg_cost=1000.0,
            is_sea=True, is_mountains=False, is_city=False, popularity=0.92),
        Destination(5, "Innsbruck", "Austria", avg_cost=950.0,
            is_sea=False, is_mountains=True, is_city=True, popularity=0.81),
    ]
    for d in destinations:
        dest_repo.add(d)


# ==============================
#   ІНІЦІАЛІЗАЦІЯ МОДЕЛІ
# ==============================

user_repo = UserRepository()
dest_repo = DestinationRepository()
seed_demo_data(user_repo, dest_repo)

clustering_model = UserClusteringModel(n_clusters=3)
clustering_model.fit(user_repo.all())

engine = RecommendationEngine(
    user_repo=user_repo,
    dest_repo=dest_repo,
    clustering_model=clustering_model,
)
```

```
# ==============================
#   ВЕБ-ІНТЕРФЕЙС (FLASK)
# ==============================

app = Flask(__name__)


@app.route("/api/recommend", methods=["GET"])
def recommend():
    """
    Ендпоінт:
    GET /api/recommend?user_id=1&top_k=3

    Повертає JSON з рекомендованими напрямками для заданого користувача.
    """
    try:
        user_id_str = request.args.get("user_id")
        if not user_id_str:
            return jsonify({"error": "Необхідний параметр user_id"}), 400

        user_id = int(user_id_str)
        top_k_str = request.args.get("top_k", "5")
        top_k = int(top_k_str)

        result = engine.recommend_for_user(user_id=user_id, top_k=top_k)
        data = {
            "user_id": result.user_id,
```

```
                "cluster_id": result.cluster_id,
                "model_version": result.model_version,
                "destinations": [
                    {
                        "dest_id": d.dest_id,
                        "name": d.name,
                        "country": d.country,
                        "avg_cost": d.avg_cost,
                        "is_sea": d.is_sea,
                        "is_mountains": d.is_mountains,
                        "is_city": d.is_city,
                        "popularity": d.popularity,
                    }
                    for d in result.destinations
                ],
            }
            return jsonify(data), 200

        except ValueError as ve:
            return jsonify({"error": str(ve)}), 400
        except Exception as ex:
            # У реальній системі тут відбувалося б логування у централізований журнал.
            return jsonify({"error": f"Внутрішня помилка сервера: {ex}"}), 500


    if __name__ == "__main__":
        # Запуск сервісу у режимі розробки.
        app.run(host="0.0.0.0", port=8000, debug=True)
```