МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА АГРЕГАТОРА НОВИН ІЗ ВИКОРИСТАННЯМ SYMFONY ТА DOCTRINE»**

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

Керівник професор кафедри програмної інженерії, доцент, к.ф.-м.н. Кудін О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

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

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

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

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Книшу Сергію Вікторовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка агрегатора новин із використанням Symfony та Doctrine

керівник роботи Кудін Олексій Володимирович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

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

2. Строк подання студентом роботи 03.12.2025 р.

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз теоретичних та архітектурних підходів.
2. Інструменти та технології розробки та тестування.
3. Реалізація модульного агрегатора новин.
4. Аналіз результатів та перспективи.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 07.03.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	28.03.2025	
2.	Збір вихідних даних.	14.04.2025	
3.	Обробка методичних та теоретичних	12.05.2025	
	джерел.		
4.	Розробка першого та другого розділу.	11.07.2025	
5.	Розробка третього розділу.	07.11.2025	
6.	Розробка четвертого розділу.	24.11.2025	
7.	Оформлення та нормоконтроль	01.12.2025	
	кваліфікаційної роботи магістра.		
8.	Захист кваліфікаційної роботи.	17.12.2025	

Студент _____ С.В. Книш
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.В. Кудін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка агрегатора новин із використанням Symfony та Doctrine»: 98 с., 23 рис., 2 табл., 18 джерел.

АГРЕГАТОР НОВИН, ВЕБСКРЕЙПІНГ, ГЕКСАГОНАЛЬНА АРХІТЕКТУРА, МОДУЛЬНИЙ МОНОЛІТ, ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ШТУЧНИЙ ІНТЕЛЕКТ, DOCTRINE, REST API, SYMFONY.

Об’єкт дослідження – процес розробки модульної інформаційної системи для агрегування, обробки та надання новинного контенту.

Мета роботи – спроєктувати та реалізувати модульний агрегатор новин із підтримкою вебскрейпінгу, AI-обробки контенту та уніфікованого REST API, придатний до масштабування й інтеграції з клієнтськими застосунками.

Методи дослідження – аналіз наукових джерел та існуючих програмних рішень, моделювання архітектури, проєктування бази даних, об’єктно-орієнтоване й доменно-орієнтоване програмування, експериментальна перевірка, модульне, інтеграційне й поведінкове тестування, статичний аналіз коду.

У роботі розроблено модульний агрегатор новин, що автоматично збирає контент з RSS- і HTML-джерел, нормалізує його та зберігає у базі даних PostgreSQL. Архітектуру побудовано як модульний моноліт із виділенням модулів Harvester, REST API, вебінтерфейсів та AI-модуля на основі принципів DDD, гексагональної архітектури та CQRS. Інтегровано AI-функції перекладу, резюмування й авто-тегування новин, реалізовано прототипи клієнтів на Next.js і Flutter, використано Docker та автоматизоване тестування, що підтвердило працездатність і розширюваність запропонованого рішення.

SUMMARY

Master's qualifying paper "Development of a News Aggregator Using Symfony and Doctrine": 98 pages, 23 figures, 2 tables, 18 references.

AI, DOCTRINE, HEXAGONAL ARCHITECTURE, MODULAR MONOLITH, NEWS AGGREGATOR, REST API, SOFTWARE TESTING, SYMFONY, WEB SCRAPING.

The object of the study is the process of developing a modular information system for aggregation, processing and delivery of news content.

The aim of the study is to design and implement a modular news aggregator with web scraping, AI-based content enrichment and a unified REST API that can be scaled and integrated with client applications.

The methods of research are analysis of scientific sources and existing software solutions, software architecture modelling, database design, object-oriented and domain-driven programming, experimental evaluation, unit, integration and behaviour-driven testing, and static code analysis.

The paper presents a modular news aggregator that automatically collects content from RSS and HTML sources, normalises it and stores it in a PostgreSQL database. The architecture is implemented as a modular monolith with separate Harvester, REST API, web interface and AI modules based on the principles of DDD, hexagonal architecture and CQRS. AI functions for translation, summarisation and auto-tagging of news are integrated, prototypes of client applications in Next.js and Flutter are implemented, and Docker-based containerisation and automated testing are used, which confirms the viability and extensibility of the proposed solution

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 9
1 Аналіз теоретичних та архітектурних підходів 11
1.1 Вебскрейпінг у сучасних інформаційних системах 11
1.2 Основні методи та алгоритми вебскрейпінгу 13
1.2.1 Виклики та обмеження у вебскрейпінгу 15
1.3 Модульна архітектура програмних систем 16
1.3.1 Переваги модульного підходу 17
1.3.2 Архітектурні стилі: DDD, шарова та гексагональна архітектури 19
1.3.3 Патерн CQRS у модульних системах 21
1.4 Огляд фреймворку Symfony..... 23
1.4.1 Переваги symfony для веброзробки 24
1.5 REST API: архітектурні принципи і призначення..... 25
1.5.1 Основні концепції REST 25
1.5.2 Безпека REST API: JWT та API-ключі..... 26
1.6 Аналіз існуючих новинних агрегаторів..... 27
1.7 Юридичні та етичні аспекти вебскрейпінгу..... 28
1.7.1 Законодавство про захист даних 28
1.7.2 Етичні норми у зборі інформації..... 28
1.8 Висновки з розділу 1..... 29
2 Інструменти та технології розробки та тестування 31
2.1 Інструменти вебскрейпінгу..... 31
2.1.1 PHP-бібліотеки для скрапінгу (guzzle, domcrawler) 32
2.1.2 Python-бібліотеки (beautifulsoup, scrapy)..... 33

2.2 Технології розробки REST API 34
2.2.1 Власна розробка REST API 35
2.2.2 Документація API зі Swagger (OpenApi) 37
2.3 Формати обміну даними 38
2.4 Допоміжні інструменти розробника 39
2.4.1 Git та інтегровані середовища розробки (IDE) 40
2.4.2 Postman та тестування API 41
2.5 Засоби забезпечення якості коду 42
2.5.1 Статичний аналіз коду та архітектури 42
2.5.2 Phpunit та Behat як засоби тестування 44
2.6 Висновки з розділу 2 46
3 Реалізація модульного агрегатора новин 47
3.1 Архітектура застосунку та взаємодія модулів 47
3.1.1 Основні модулі 48
3.1.2 Взаємодія між модулями 50
3.2 Реалізація модуля вебскрейпінгу 52
3.2.1 Алгоритми парсингу новинних ресурсів та нормалізація даних 53
3.2.2 Запис даних у базу та потік AI-обробки новини 55
3.3 Розробка REST API 59
3.3.1 Структура api та маршрути 59
3.3.2 Методи та ресурси 61
3.3.3 Безпека та аутентифікація 63
3.3.4 Вебінтерфейси користувачів, адміністратора та майстер Додавання джерел з підтримкою AI 65
3.4 Організація бази даних 69
3.4.1 База даних *news-harvester*: модель новин та ai-метадані 70
3.4.2 База даних *news-rest*: користувачі, API-ключі та журнали запитів 74
3.5 Використання Docker для локального середовища розробки 76

3.6 Клієнтські застосунки – споживачі REST API 79
3.6.1 Вебклієнт на Next.js 79
3.6.2 Мобільний додаток на Flutter 82
3.7 Тестування функціональності та продуктивності 85
3.7.1 Автоматизоване тестування (Phpunit) 85
3.7.2 Поведінкове тестування (Behat) 86
3.8 Висновки з розділу 3 88
4 Аналіз результатів та перспективи 90
4.1 Оцінка ефективності системи 90
4.2 Аналіз масштабованості та продуктивності 91
4.3 Обмеження та труднощі дослідження 92
4.4 Перспективи розвитку та впровадження 93
4.5 Висновки з розділу 4 94
Висновки 95
Перелік посилань 97

ВСТУП

Стрімке зростання кількості вебресурсів, онлайн-медіа та соціальних мереж призводить до інформаційного перевантаження: новини розпорошені між багатьма сайтами, подаються в різних форматах і не мають єдиного пункту доступу. Це ускладнює централізований збір, аналіз і повторне використання контенту, тому актуальним є створення програмних засобів, здатних автоматично отримувати новини з різних джерел, нормалізувати їх та надавати через зручний інтерфейс.

Кваліфікаційна робота присвячена розробці модульного агрегатора новин на базі фреймворку Symfony [16] з використанням Doctrine ORM і реляційної СУБД PostgreSQL [15]. Система реалізується як модульний моноліт і містить модулі збору новин (скрапінгу), REST API й адміністративного інтерфейсу з вбудованими засобами AI-обробки (переклад, стислий виклад, формування тегів).

Об’єкт дослідження – програмні системи для автоматизованого збору, обробки та надання новинних даних з вебджерел.

Предмет дослідження – методи вебскрейпінгу, архітектурні підходи до побудови модульних вебзастосунків і способи організації REST API для доступу до агрегованих новин.

Мета магістерської роботи – розробити модульний агрегатор новин на базі Symfony, який забезпечує збір новин із RSS- та HTML-джерел, їх нормалізацію та збереження в реляційній базі даних, надання REST API для доступу до контенту та використання засобів штучного інтелекту для його збагачення.

Для досягнення мети необхідно розв’язати такі завдання:

– проаналізувати підходи до вебскрейпінгу, модульної архітектури та REST API у сучасних вебзастосунках;
– дослідити інструменти вебскрейпінгу в екосистемах PHP і Python та обґрунтувати вибір стеку технологій;

- обґрунтувати застосування доменно-орієнтованого дизайну, гексагональної архітектури та патерну CQRS у модульному моноліті;
- спроєктувати й реалізувати модуль збору новинних ресурсів та REST API для роботи з новинами й джерелами;
- спроєктувати структуру бази даних, налаштувати збереження даних у Doctrine ORM та виконати базове тестування функціональності системи.

Теоретичне значення роботи полягає в узагальненні підходів до вебскрейпінгу та модульної архітектури вебзастосунків, а також у систематизації способів інтеграції доменно-орієнтованого дизайну, гексагональної архітектури, патерну CQRS і засобів штучного інтелекту в новинні агрегатори.

Практичне значення роботи полягає у створенні прототипу агрегатора новин, який може бути основою для реальної новинної платформи або компонентом інформаційних систем, де потрібні автоматичний збір, обробка та збагачення текстового контенту.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків та списку використаних джерел. Перший розділ містить теоретичні відомості про вебскрейпінг та архітектуру модульних систем. У другому розділі розглянуто вибір інструментів і технологій. Третій розділ присвячено архітектурі та реалізації програмного забезпечення. У четвертому розділі наведено результати експериментальних досліджень та окреслено напрями подальшого розвитку системи.

1 АНАЛІЗ ТЕОРЕТИЧНИХ ТА АРХІТЕКТУРНИХ ПІДХОДІВ

1.1 Вебскрейпінг у сучасних інформаційних системах

Вебскрейпінг (web scraping) – це автоматизований процес отримання структурованих даних з вебсторінок [18]. На відміну від традиційного перегляду вебсайтів користувачем, вебскрейпінг виконується програмно, із застосуванням спеціалізованих бібліотек та інструментів, що дозволяють надсилати HTTP-запити, аналізувати HTML-, XML- або JSON-документи та витягувати з них потрібні фрагменти.

У загальному випадку процес вебскрейпінгу включає такі етапи:

– надсилання HTTP-запиту до цільового ресурсу;
– отримання HTML-, XML- або JSON-відповіді;
– виділення релевантних ділянок документа (наприклад, списку новинних блоків);
– парсинг окремих полів (заголовок, опис, посилання, дата публікації, зображення тощо);
– нормалізація даних і перетворення їх до внутрішнього формату системи;
– збереження результатів у сховищі даних, як правило, в реляційній базі даних (у даній роботі – PostgreSQL [15]) для подальшої обробки та аналітики.

У розробленому програмному комплексі вебскрейпінг використовується як засіб автоматичного наповнення бази новин з офіційних новинних ресурсів, що публікують контент через RSS або HTML-сторінки. Це дозволяє централізовано збирати дані з декількох джерел, уніфікувати їх структуру та готувати до подальшої обробки засобами штучного інтелекту.

Вебскрейпінг відіграє важливу роль у багатьох сучасних інформаційних системах, зокрема:

– новинні агрегатори – збирають матеріали з різних ЗМІ для формування єдиної стрічки новин;
– системи моніторингу цін – відстежують зміну вартості товарів в інтернет-магазинах;
– аналітичні платформи – накопичують дані для подальшої бізнес-аналітики, машинного навчання та прогнозування;
– інструменти конкурентної розвідки – фіксують оновлення продуктів, новини компаній та активність конкурентів;
– системи моніторингу соціальних мереж – аналізують публікації, коментарі та реакції користувачів.

У контексті цієї кваліфікаційної роботи вебскрейпінг є основним механізмом отримання новинних даних з різних джерел, їх нормалізації та подальшого збереження в реляційній базі даних PostgreSQL. Надалі ці дані обробляються AI-модулем (переклад, резюмування, автоматичне тегування) та надаються клієнтським застосункам через REST API.

На рисунку 1.1 наведено узагальнену схему процесу вебскрейпінгу та подальшої обробки новинних даних у розробленій системі.

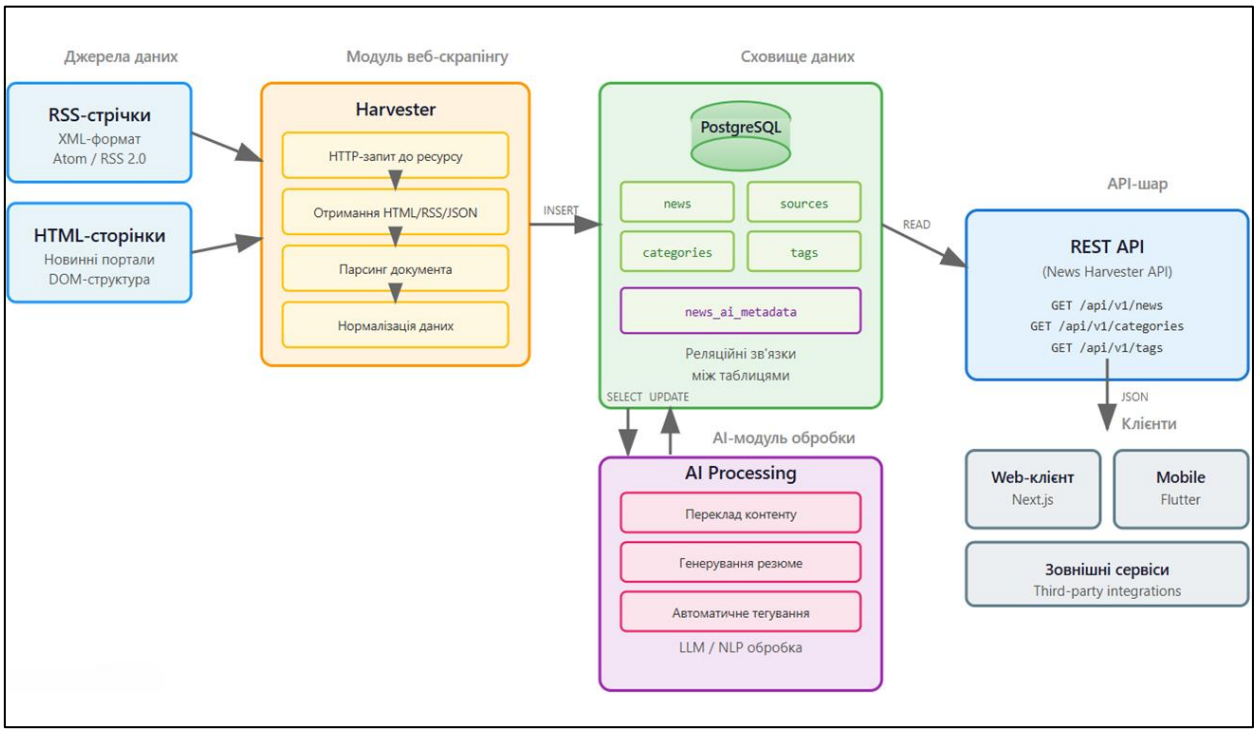


Рисунок 1.1 – Загальна схема процесу вебскрейпінгу в розробленій системі

Таким чином вебскрейпінг виступає початковою ланкою цілісного конвеєра обробки інформації, який включає збирання, зберігання, інтелектуальну обробку та публікацію новинного контенту.

1.2 Основні методи та алгоритми вебскрейпінгу

Розглянемо найбільш поширені методи вебскрейпінгу.

Парсинг статичних HTML-сторінок. У цьому випадку вебсторінка генерується на сервері й надсилається клієнту як готовий HTML-документ. Для отримання даних достатньо завантажити HTML-код, виділити потрібні фрагменти та розпарсити їх. Цей підхід є відносно простим у реалізації, оскільки структура сторінки зазвичай є стабільною, а зміни у верстці можна відстежувати й поступово адаптовувати парсер. Для парсингу використовуються DOM-парсери, механізми пошуку елементів за CSS-селекторами або XPath, а також процедури очищення розмітки.

Робота з RSS/Atom-стрічками. Багато новинних сайтів надають RSS- або Atom-стрічки, які містять структуровані дані про публікації (заголовок, посилання, короткий опис, дату, іноді зображення). Такий формат значно спрощує парсинг, оскільки має чітку XML-структуру й менш залежний від змін верстки сайту. Використання RSS дозволяє швидко отримувати основні метадані новин, а за потреби – додатково завантажувати повні тексти матеріалів зі сторінок за посиланнями.

Скрапінг динамічних сторінок. Частина сучасних сайтів активно застосовує JavaScript для підвантаження контенту (наприклад, через AJAX-запити або використання SPA-фреймворків). У таких випадках простий HTTP-запит до HTML-сторінки може не містити потрібних даних. Для вирішення цієї проблеми застосовують або емуляцію браузера (наприклад, за допомогою headless-браузерів), або аналіз мережевої активності сторінки з подальшим зверненням безпосередньо до внутрішніх API. Цей підхід складніший, але дозволяє отримувати дані з динамічно згенерованого контенту.

Комбіновані підходи. На практиці системи збору даних часто поєднують кілька методів: RSS використовується для виявлення нових публікацій, а HTML-парсинг – для витягування повного тексту, зображень і додаткових метаданих. Такий підхід дозволяє зменшити навантаження на зовнішні ресурси й водночас забезпечити більш повний та структурований набір даних для подальшого збереження в реляційній базі даних (зокрема PostgreSQL) та обробки засобами штучного інтелекту.

Парсинг HTML зазвичай базується на використанні:

– DOM-парсерів, які перетворюють HTML у дерево вузлів;
– CSS-селекторів або XPath для пошуку потрібних елементів;
– механізмів очищення розмітки (видалення скриптів, стилів, технічних блоків).

Узагальнену класифікацію основних методів вебскрейпінгу наведено на рисунку 1.2.

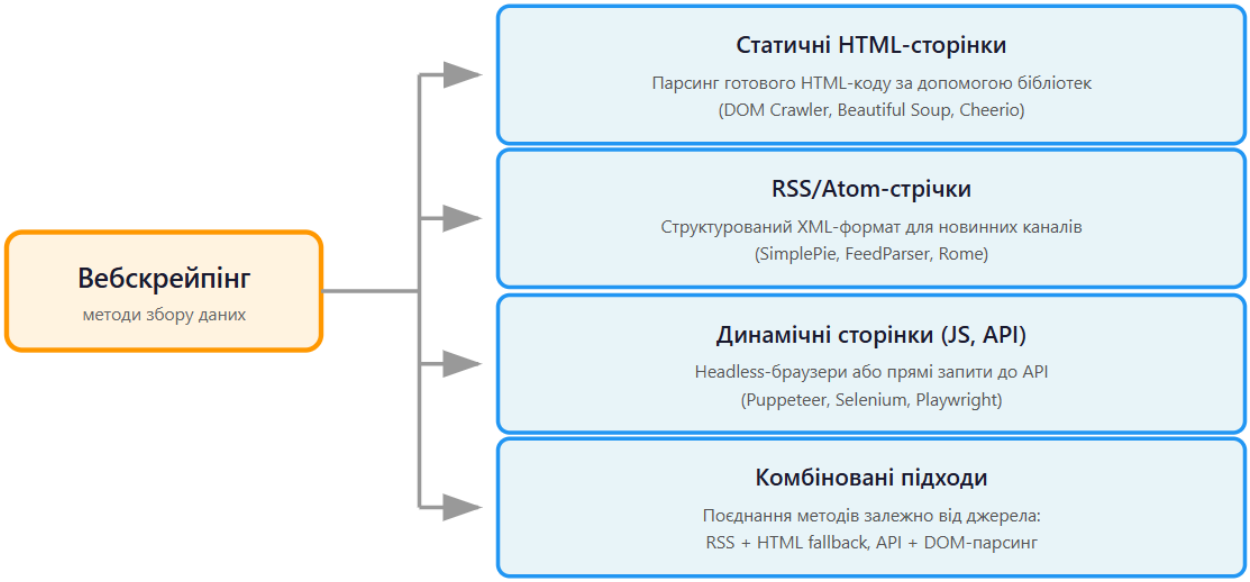


Рисунок 1.2 – Класифікація основних методів вебскрейпінгу

У PHP для цього широко застосовуються бібліотеки на кшталт Symfony DomCrawler [16], які дозволяють будувати запити до DOM-дерева, поєднані з HTTP-клієнтами (наприклад, Guzzle [9] або Symfony HttpClient [16]) для отримання вмісту сторінок. У результаті вихідні неструктуровані HTML-

документи перетворюються на структуровані об’єкти, придатні для подальшого збереження у базі даних.

JSON у контексті вебскрейпінгу використовується у двох основних сценаріях:

– багато сайтів надають публічні або внутрішні API, що повертають дані безпосередньо у форматі JSON;
– окремі сторінки містять вбудовані JSON-блоки (наприклад, у тегах *<script type="application/ld+json">*), які описують структуровані дані про публікації.

Отримані JSON-дані можуть безпосередньо перетворюватися у внутрішні DTO- або доменні моделі й зберігатися в реляційній базі даних без проміжного HTML-парсингу. У межах цієї роботи основний акцент зроблено на комбінованому використанні RSS та HTML-парсингу, що забезпечує заповнення структури бази даних PostgreSQL повними текстами новин, метаданими та полями, необхідними для подальшої AI-обробки й надання доступу до контенту через REST API.

1.2.1 Виклики та обмеження у вебскрейпінгу

Реалізація вебскрейпінгу пов’язана з низкою викликів і обмежень, які необхідно враховувати під час розробки програмних систем.

Зміна структури сайту. Будь-які зміни у верстці або структурі HTML-документів можуть призвести до некоректної роботи парсерів. Це вимагає регулярного моніторингу джерел, оновлення селекторів і підтримки окремих парсер-класів для різних сайтів, бажано з виділенням спільної базової логіки.

Обмеження за частотою запитів. Багато сайтів обмежують кількість запитів від одного клієнта (rate limiting) або захищаються від автоматизованого доступу. Тому важливо враховувати затримки між запитами, кешування результатів, раціональне планування розкладу запуску скрапінгу та дотримання правил файлу **robots.txt**.

Юридичні та етичні ризики. Некоректний або агресивний скрапінг може суперечити умовам використання ресурсу, авторським правам або законодавству про захист даних. Необхідно враховувати публічний статус джерел, ліцензійні обмеження та дотримуватися етичних принципів використання зібраної інформації.

Продуктивність і масштабованість. Одночасний збір великого обсягу даних із багатьох джерел створює значне навантаження як на зовнішні ресурси, так і на внутрішню інфраструктуру системи – мережу, черги завдань, базу даних. Для мінімізації затримок та забезпечення стійкої роботи застосовують черги повідомлень, асинхронні HTTP-запити, оптимізацію структури сховища даних (наприклад, у PostgreSQL – індекси, продуману схему таблиць) та горизонтальне масштабування окремих компонентів.

У даній роботі основну увагу приділено розробці надійного процесу скрапінгу для помірної кількості якісно відібраних джерел, із фокусом на стабільність парсерів, коректне поводження з зовнішніми ресурсами та підготовку даних до подальшої обробки засобами штучного інтелекту й публікації через REST API.

1.3 Модульна архітектура програмних систем

Модульна архітектура програмних систем передбачає розподіл застосунку на відносно незалежні частини (модулі), кожна з яких відповідає за окрему підобласть функціональності і взаємодіє з іншими через чітко визначені інтерфейси. Такий підхід дозволяє зменшити зв’язність компонентів, спростити супровід, розширення та тестування системи.

Модуль зазвичай об’єднує в собі повний набір елементів, необхідних для реалізації певних бізнес-можливостей: доменні моделі, сервіси, інтерфейси доступу до даних, адаптери зовнішніх інтеграцій. На відміну від простого розбиття коду за технічними ознаками (наприклад, «контролери», «сервіси»,

«репозиторії»), модульний підхід фокусується на предметній області та бізнес-функціях.

У контексті агрегаторів новин модульний підхід дозволяє виділити окремі компоненти для збору новин (web-scraping/harvester), обробки та збагачення даних засобами штучного інтелекту, надання REST API для клієнтських застосунків, а також керування користувачами й правами доступу. При цьому всі модулі можуть спільно використовувати єдину реляційну базу даних (наприклад, PostgreSQL [15]), але взаємодія з нею відбувається через чітко визначені інтерфейси та репозиторії.

У подальших підрозділах розглянуто ключові переваги модульної архітектури, а також основні архітектурні стилі (Domain-Driven Design, шарову та гексагональну архітектури) і патерн CQRS, які застосовуються при проєктуванні модульних систем. У другому розділі ці підходи будуть конкретизовані на прикладі архітектури агрегатора новин та структури його бази даних.

1.3.1 Переваги модульного підходу

Модульна архітектура має низку істотних переваг порівняно з монолітними системами без внутрішнього поділу на модулі.

По-перше, зменшується зв’язність компонентів. Кожен модуль має чітко визначений контракт взаємодії з іншими модулями (інтерфейси, порти, DTO), що спрощує внесення змін у внутрішню реалізацію без впливу на решту системи. Це особливо важливо для довготривалих проєктів, які активно розвиваються.

По-друге, спрощується масштабування. Окремі модулі, що піддаються найбільшому навантаженню (наприклад, модуль скрапінгу новин або публічний REST API), можуть бути масштабовані незалежно від інших компонентів. Навіть у межах одного виконуваного файла (модульного

моноліту) такий підхід полегшує подальший перехід до мікросервісної архітектури.

По-третє, підвищується тестованість системи. Кожен модуль можна тестувати в ізоляції, використовуючи підміни (mocks, stubs) замість реальних реалізацій інших модулів. Це спрощує модульне й інтеграційне тестування та дозволяє швидше виявляти помилки.

По-четверте, модульний підхід сприяє кращому розподілу відповідальності в команді. Різні групи розробників можуть працювати над окремими модулями (наприклад, над модулем вебскрейпінгу, AI-модулем, модулем REST API), узгоджуючи між собою лише контракти взаємодії, але не втручаючись у внутрішню реалізацію один одного.

На рисунку 1.3 наведено приклад загальної структури модульної архітектури для агрегатора новин.

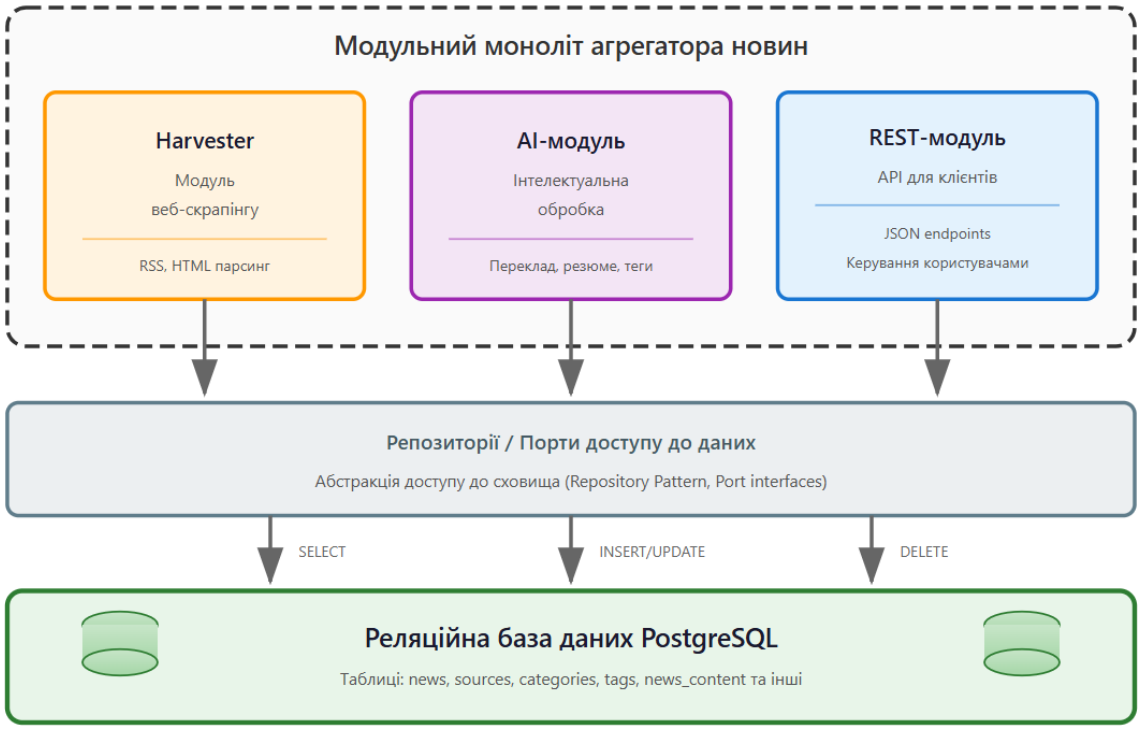


Рисунок 1.3 – Загальна схема модульної архітектури агрегатора новин

Для систем, що зберігають значні обсяги даних у реляційних базах (зокрема в PostgreSQL), модульна архітектура дозволяє чітко прив’язати певні групи таблиць та моделей до окремих модулів. Це полегшує еволюцію схеми

бази даних, розробку міграцій і розподіл прав доступу між компонентами застосунку.

1.3.2 Архітектурні стилі: DDD, шарова та гексагональна архітектури

Доменно-орієнтований дизайн (Domain-Driven Design, DDD) є підходом до розробки програмного забезпечення [1], [17], який фокусується на глибокому розумінні предметної області та спільній роботі розробників із доменними експертами. Основні ідеї DDD включають:

- виділення чітких *bounded context* – меж контекстів [1], у межах яких терміни та моделі мають однозначне значення;
- моделювання доменних сутностей, об’єктів-значень (Value Objects), агрегатів і доменних сервісів;
- концентрацію бізнес-логіки в доменному шарі;
- побудову *ubiquitous language* [1] – спільної мови, якою користуються як розробники, так і експерти предметної області.

Шарова архітектура є одним із базових стилів [4], що широко застосовується у корпоративних системах. Класичний варіант передбачає поділ кодової бази на такі шари:

- презентаційний (UI, REST-контролери, CLI-команди);
- прикладний (application layer, use cases, координація бізнес-операцій);
- доменний (сутності, доменні сервіси, інтерфейси репозиторіїв);
- інфраструктурний (реалізації репозиторіїв, доступ до бази даних, інтеграція з зовнішніми сервісами).

Основною перевагою шарової архітектури є логічне відокремлення відповідальностей між шарами та чіткий напрямок залежностей: верхні шари залежать від нижчих, але не навпаки.

Гексагональна архітектура (Hexagonal Architecture, Ports and Adapters) робить акцент на ізоляції бізнес-логіки від зовнішніх залежностей [4]. У центрі знаходиться доменне ядро – код, який реалізує предметну логіку та не залежить від фреймворків, бібліотек чи конкретної СУБД. Взаємодія із зовнішнім світом організовується через порти – абстрактні інтерфейси, які визначає домен.

На рисунку 1.4 зображено вид спрощеної гексагональної архітектури

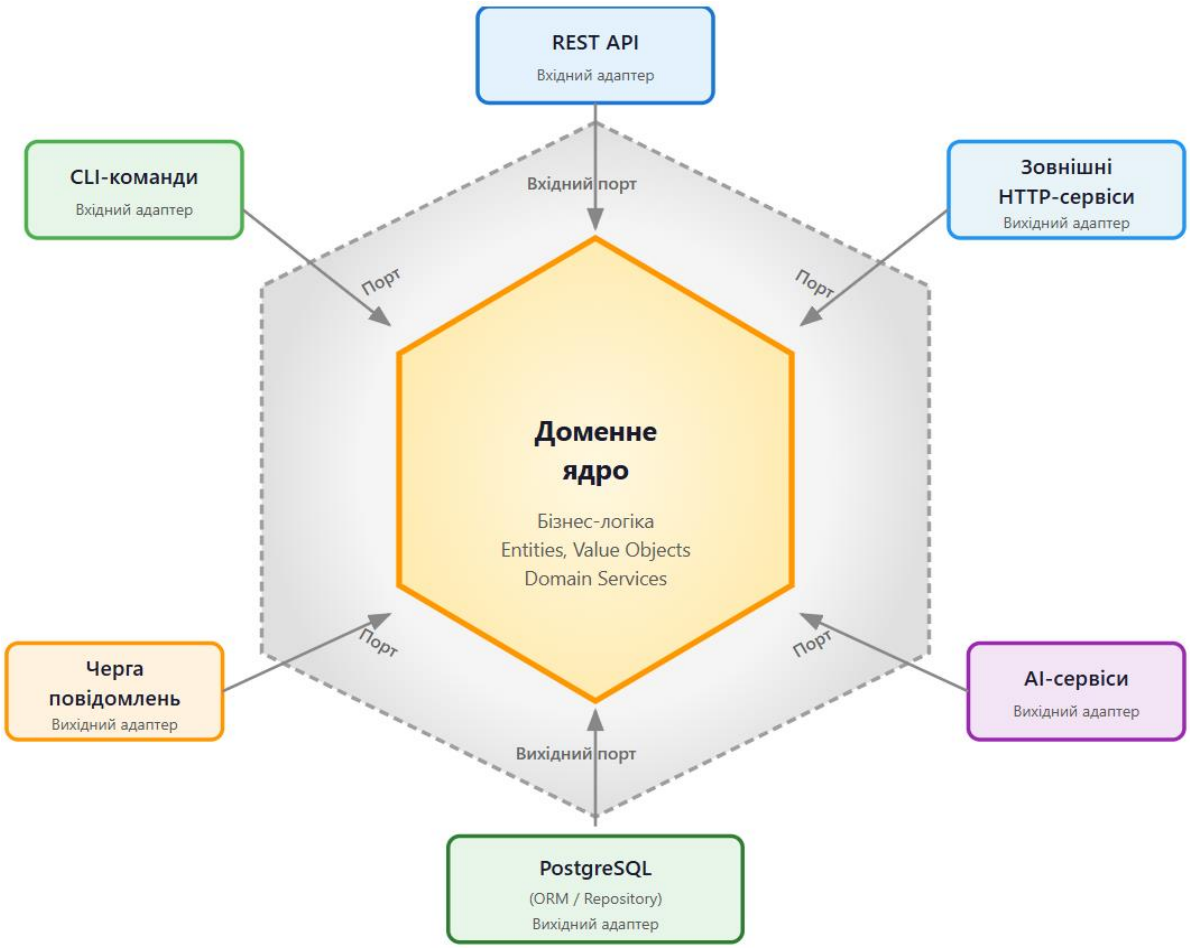


Рисунок 1.4 – Спрощене зображення гексагональної архітектури

Порти розрізняють на:

– вхідні порти (input/driving ports) – інтерфейси, через які зовнішні компоненти (UI, REST API, CLI) ініціюють дії в системі;
– вихідні порти (output/driven ports) – інтерфейси, які домен використовує для доступу до зовнішніх ресурсів (база даних, черги повідомлень, зовнішні HTTP-сервіси тощо).

Адаптери є конкретними реалізаціями цих портів. Наприклад, адаптер бази даних реалізує вихідний порт за допомогою ORM (у даній роботі – Doctrine, що працює з PostgreSQL), а REST-контролери реалізують вхідні порти, приймаючи запити клієнтів і викликаючи відповідні методи прикладного шару.

Ключова перевага гексагональної архітектури полягає в тому, що доменне ядро не залежить від технічних деталей реалізації. Це спрощує заміну інфраструктурних компонентів, тестування бізнес-логіки (через використання макетів портів) і поступову еволюцію системи. У поєднанні з модульним підходом DDD та гексагональна архітектура дозволяють будувати модульні моноліти, які в майбутньому можуть бути розбиті на окремі мікросервіси без радикальної переробки всієї системи.

1.3.3 Патерн CQRS у модульних системах

Патерн **CQRS** (Command Query Responsibility Segregation) є архітектурним підходом [1], [17], який передбачає розділення операцій читання та запису даних на різні моделі та, за можливості, різні компоненти. Він розвиває ідею принципу CQS (Command Query Separation), згідно з яким метод або змінює стан системи, або повертає дані, але не робить обидві дії одночасно.

У межах CQRS:

– **команди** (commands) відповідають за зміну стану системи (створення, оновлення, видалення об’єктів) і зазвичай не повертають результатів, окрім підтвердження успішного виконання;
– **запити** (queries) призначені для читання даних і не змінюють стан системи.

Традиційний CRUD-підхід часто використовує єдину модель доменних об’єктів як для запису, так і для читання, що може ускладнювати оптимізацію

запитів для різних сценаріїв. CQRS пропонує розділити моделі: одна – для команд (зосереджена на цілісності даних і бізнес-правилах), інша – для запитів (оптимізована для швидкого доступу й зручного формату даних для клієнтів).

У складніших варіантах CQRS може поєднуватися з використанням різних сховищ даних для читання й запису, а також із патерном Event Sourcing. Проте навіть у простій формі (розділення команд і запитів у межах одного застосунку) патерн полегшує підтримку коду й дозволяє явно розділити відповідальність.

На рисунку 1.5 подано узагальнену схему патерна CQRS, яка ілюструє окремі контури обробки команд (COMMAND) і запитів (QUERY), а також синхронізацію між базою запису (PostgreSQL) та оптимізованою базою читання (Redis / Elastic).

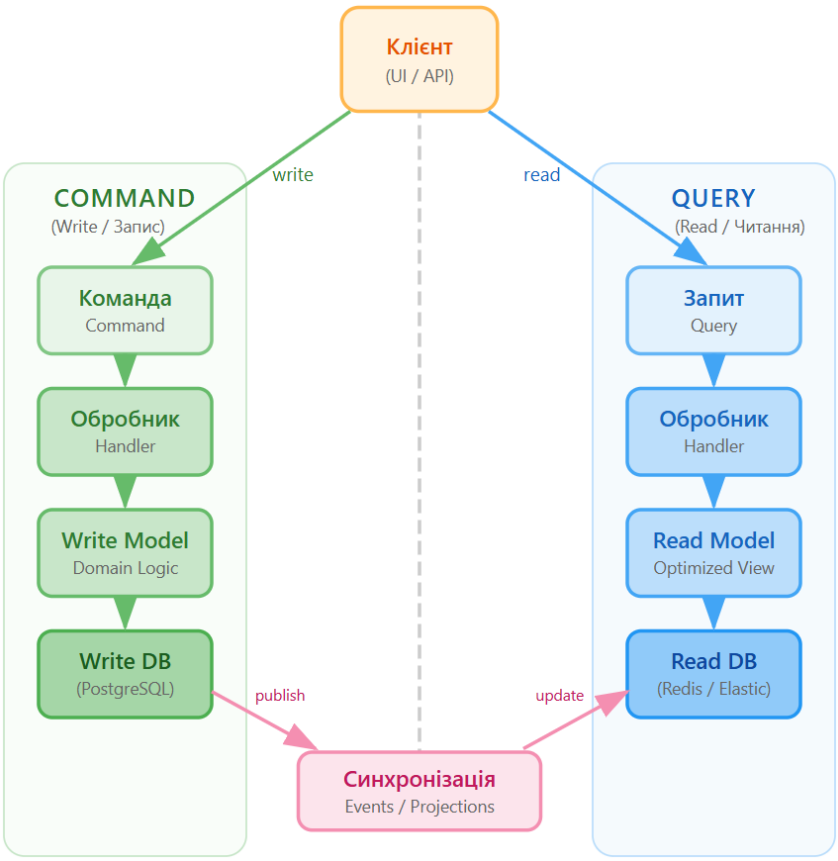


Рисунок 1.5 – Схематичне зображення CQRS патерну

У подальших розділах CQRS розглядається як один із патернів, що використовується під час проєктування логіки команд і запитів в агрегаторі

новин, зокрема при організації шарів застосунку та структури доступу до даних у PostgreSQL.

Для модульних систем, які працюють з великим обсягом даних у реляційних базах, застосування CQRS дає змогу:

– спроєктувати командну частину навколо доменної моделі та транзакційної цілісності (наприклад, робота з таблицями PostgreSQL через ORM);
– побудувати окремий шар запитів, який використовує спеціально підготовлені SELECT-запити, проекції та представлення даних для REST API без зайвої бізнес-логіки.

1.4 Огляд фреймворку Symfony

Symfony [16] – це сучасний PHP-фреймворк, орієнтований на розробку масштабованих вебзастосунків і сервісів. Фреймворк надає розвинену колекцію компонентів, які можуть використовуватися як у складі повного фреймворку, так і окремо в інших проєктах.

У межах розробки новинного агрегатора застосовуються такі ключові компоненти Symfony:

– HttpKernel та Routing – для обробки HTTP-запитів і маршрутизації;
– DependencyInjection – для керування залежностями й конфігурацією сервісів;
– Doctrine ORM та Doctrine DBAL – для взаємодії з реляційною базою даних PostgreSQL;
– Console – для реалізації CLI-команд, зокрема для імпорту новин та запуску допоміжних задач;
– Security – для організації механізмів автентифікації й авторизації в адміністративній частині;
– Validator, Serializer – для валідації вхідних даних та серіалізації DTO в JSON-подання;

– засоби інтеграції з OpenAPI [13] (наприклад, NelmioApiDocBundle [16]) – для автоматичної генерації документації до REST API.

Гнучка система конфігурації сервісів, розподіл коду за просторами імен та наявність розвиненої екосистеми пакетів роблять Symfony придатним для реалізації модульної архітектури та підходів на основі DDD і гексагональної архітектури.

1.4.1 Переваги Symfony для веброзробки

Вибір Symfony як основного фреймворку для побудови агрегатора новин зумовлений низкою його переваг.

По-перше, фреймворк характеризується стабільністю і зрілістю. Symfony широко використовується в промислових проєктах, має довгострокову підтримку версій і розвинену документацію, що важливо для проєктів із тривалим життєвим циклом.

По-друге, Symfony пропонує гнучку архітектуру і не нав’язує жорсткої структури застосунку. Це дозволяє реалізувати модульний моноліт, застосувати підходи Domain-Driven Design, гексагональну архітектуру та патерн CQRS, зберігаючи при цьому сумісність зі стандартними практиками PHP-розробки.

По-третє, значною перевагою є багата екосистема. Наявність численних бандлів і готових рішень (для реалізації безпеки, кешування, документування API, інтеграції з чергами повідомлень тощо) скорочує витрати часу на розробку допоміжної інфраструктури й дозволяє зосередитися на бізнес-логіці.

По-четверте, Symfony надає зручні інструменти для розробника: командний інтерфейс, профайлер, інтеграцію з сучасними середовищами розробки та засобами контейнеризації (Docker), підтримку актуальних стандартів PHP та PSR-рекомендацій.

У поєднанні з можливістю побудови модульної архітектури ці властивості роблять Symfony природним вибором для реалізації ядра агрегатора новин із використанням реляційної бази даних PostgreSQL та інтегрованого AI-функціоналу.

1.5 REST API: архітектурні принципи і призначення

1.5.1 Основні концепції REST

REST (Representational State Transfer) – це архітектурний стиль побудови розподілених систем, у якому взаємодія між клієнтом і сервером здійснюється через стандартизовані HTTP-запити до ресурсів. До основних принципів REST належать:

- клієнт-серверна архітектура, що розділяє відповідальність за інтерфейс користувача та зберігання даних;
- відсутність стану на стороні сервера в межах одного запиту (statelessness);
- використання єдиного інтерфейсу взаємодії (URI для ресурсів, стандартні методи HTTP – GET, POST, PUT, DELETE тощо);
- підтримка кешування відповідей для підвищення продуктивності;
- шаруватість (layered system), що допускає наявність проміжних проксі, кешів та шлюзів.

Ресурсами в REST-застосунках зазвичай виступають сутності предметної області (наприклад, новина, джерело, категорія, тег). Їхній стан передається у вигляді представлень (JSON, XML тощо), а маніпуляції зі станом виконуються через HTTP-методи відповідно до семантики CRUD-операцій.

У розроблюваному агрегаторі новин REST API використовується як універсальний спосіб доступу до новинних даних. Єдиний програмний інтерфейс забезпечує роботу різних клієнтських застосунків (вебінтерфейс,

мобільний клієнт, сторонні інтеграції) з однією й тією ж серверною логікою та базою даних PostgreSQL, що спрощує підтримку й розширення системи.

1.5.2 Безпека REST API: JWT та API-ключі

Питання безпеки є ключовими для будь-якого REST API, оскільки доступ до ресурсів має бути обмежений відповідно до прав користувачів і сценаріїв використання. На практиці широко застосовуються два поширені підходи: автентифікація на основі токенів (зокрема JSON Web Token, JWT) та використання API-ключів.

API-ключі зазвичай застосовуються для анонімного або умовно анонімного доступу до публічних ресурсів. Клієнту видається унікальний ключ, який передається в заголовку Authorization (наприклад, як Bearer-токен) або в параметрах запиту. Такий механізм дозволяє:

– ідентифікувати джерело запитів;
– застосовувати обмеження швидкості (rate limiting) для окремих ключів;
– відстежувати використання API та при необхідності блокувати ключі, що порушують правила.

JWT (JSON Web Token) здебільшого використовується для автентифікації користувачів у сценаріях, де потрібен персоналізований доступ до ресурсів (наприклад, збережені налаштування, «обране», коментарі). Токен містить закодовані відомості про користувача та його ролі, підписується на стороні сервера й передається клієнтом у вигляді заголовка Authorization під час кожного запиту.

У контексті агрегатора новин доцільно поєднувати обидва підходи: публічний доступ до новин може здійснюватися через API-ключі з обмеженням швидкості та логуванням запитів, тоді як для повноцінної взаємодії автентифікованих користувачів (наприклад, у мобільному клієнті) застосовується окремий користувацький API з використанням JWT. Детальні

механізми реалізації розглядаються в подальших розділах, присвячених архітектурі та реалізації системи.

1.6 Аналіз існуючих новинних агрегаторів

На ринку представлена значна кількість новинних агрегаторів, серед яких:

– міжнародні платформи (Google News, Apple News тощо);
– локальні інформаційні портали;
– тематичні агрегатори (наприклад, у сфері IT, фінансів, спорту).

Більшість великих агрегаторів є закритими системами. Для них характерні такі особливості:

– відсутність відкритого публічного API або суттєві обмеження на його використання;
– складність отримання доступу до структурованих даних у зручному для обробки форматі;
– обмежені можливості інтеграції власних сервісів обробки даних, зокрема AI-сервісів для перекладу, аналізу чи автоматичного тегування.

Під час аналізу доступних API новин було виявлено, що частина з них є надто спрощеними й не підтримує складні сценарії фільтрації, сортування чи агрегації. Інша частина надається на комерційних умовах із жорсткими лімітами за кількістю запитів і обсягом даних, що ускладнює побудову гнучких дослідницьких або навчальних рішень.

У зв’язку з цим актуальною є побудова власного новинного агрегатора як гнучкої платформи з модульною архітектурою, можливістю підключення нових джерел, повноцінним REST API для власних і сторонніх клієнтів, а також інтегрованим AI-функціоналом на рівні доменної логіки. Такий підхід дозволяє контролювати структуру бази даних, алгоритми обробки й правила доступу до інформації.

1.7 Юридичні та етичні аспекти вебскрейпінгу

1.7.1 Законодавство про захист даних

Вебскрейпінг повинен здійснюватися з урахуванням чинного законодавства про авторські права [2], захист персональних даних та електронні комунікації. До відповідних нормативних актів належать, зокрема, Загальний регламент із захисту даних (GDPR) в ЄС [8], Закон України «Про захист персональних даних» [3] та інші галузеві нормативні документи.

Під час розробки систем, що виконують скрапінг, доцільно дотримуватися таких принципів:

- використовувати як джерела офіційні новинні сайти, які публікують матеріали для широкої аудиторії;
- не збирати й не зберігати персональні дані користувачів без необхідності та належної правової підстави;
- зберігати та відображати посилання на оригінальні джерела новин;
- передбачати можливість оперативного вимкнення скрапінгу для окремого ресурсу у разі відповідної вимоги з боку його власника або зміни умов використання.

Дотримання цих принципів зменшує ризики порушення законодавства та сприяє побудові відповідальних інформаційних сервісів.

1.7.2 Етичні норми у зборі інформації

Окрім юридичних вимог, важливе значення мають етичні аспекти вебскрейпінгу. До основних етичних принципів можна віднести:

- недопущення надмірного навантаження на зовнішні ресурси шляхом обмеження частоти запитів та використання кешування;
- повагу до вказівок, що містяться у файлі robots.txt та умовах використання вебсайту;

– чітке зазначення джерела новини в інтерфейсах клієнтських застосунків;
– утримання від змін змісту новин, які можуть призвести до спотворення їхнього сенсу або контексту.

У межах розроблюваного агрегатора новин застосовуються розумні інтервали між запитами, налагоджується логування помилок і аномальних ситуацій під час імпорту, зберігаються оригінальні заголовки, короткі анотації та URL-адреси публікацій. Це дозволяє поєднати технічну ефективність системи з дотриманням етичних стандартів використання відкритих вебджерел.

1.8 Висновки з розділу 1

У першому розділі розглянуто теоретичні засади вебскрейпінгу та його роль у сучасних інформаційних системах, проаналізовано основні методи й алгоритми отримання даних з вебресурсів, а також виклики та обмеження, пов’язані з їх практичним застосуванням. Окрему увагу приділено модульній архітектурі програмних систем, підходу Domain-Driven Design, шаровій та гексагональній архітектурам, а також патерну CQRS як основі для побудови модульного моноліту.

Наведено огляд фреймворку Symfony та обґрунтовано доцільність його використання для розробки агрегатора новин із реляційною базою даних PostgreSQL та інтегрованим AI-функціоналом. Розглянуто основні концепції REST та підходи до забезпечення безпеки REST API на основі JWT і API-ключів.

Проведено аналіз існуючих новинних агрегаторів, виявлено обмеження надання структурованих даних і відкритих API, що підтверджує актуальність розробки власного рішення з модульною архітектурою та гнучкими можливостями інтеграції. Окреслено юридичні та етичні аспекти

вебскрейпінгу, дотримання яких є необхідною умовою побудови відповідальних інформаційних сервісів.

Сформульовані у розділі теоретичні положення та узагальнення слугують основою для подальшого проєктування архітектури агрегатора новин, визначення структури його бази даних і реалізації програмного забезпечення, що розглядається в наступних розділах.

2 ІНСТРУМЕНТИ ТА ТЕХНОЛОГІЇ РОЗРОБКИ ТА ТЕСТУВАННЯ

2.1 Інструменти вебскрейпінгу

У цьому підрозділі розглянуто інструментальні засоби вебскрейпінгу, що використовуються під час розроблення програмного комплексу агрегатора новин. Основною задачею цих засобів є отримання HTML- та RSS-документів із зовнішніх новинних ресурсів, їх синтаксичний аналіз, виділення релевантних фрагментів (заголовків, текстів публікацій, метаданих) та передавання нормалізованих даних до внутрішніх моделей системи.

Інструменти вебскрейпінгу умовно поділяються на засоби екосистеми PHP, які безпосередньо застосовуються в реалізації модулів системи, та поширені бібліотеки мови Python, що розглядаються з метою порівняння підходів і потенційного використання в подальших версіях програмного комплексу.

Структуру основних етапів процесу вебскрейпінгу наведено на рисунку 2.1.

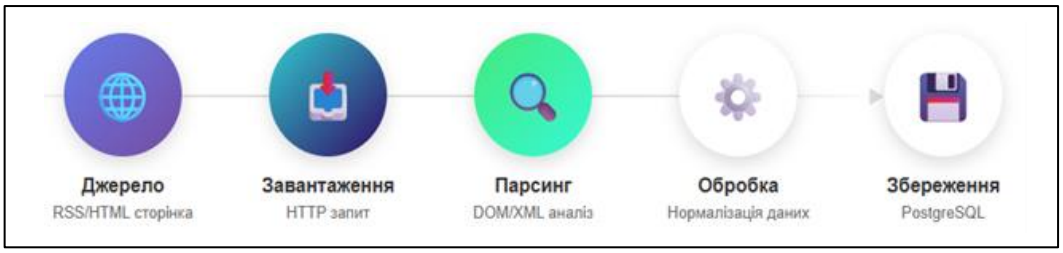


Рисунок 2.1 – Основні етапи процесу вебскрейпінгу новинних ресурсів

Під час роботи модуля послідовно виконуються такі кроки: «Джерело» (отримання RSS/HTML-сторінок), «Завантаження» (формування та виконання HTTP-запитів), «Парсинг» (DOM/XML-аналіз отриманого документа), «Обробка» (нормалізація та підготовка даних) та «Збереження» (запис структурованих даних у сховище, зокрема у базу даних PostgreSQL).

2.1.1 PHP-бібліотеки для скрапінгу (Guzzle, DomCrawler)

Для реалізації модуля вебскрейпінгу використовується стек інструментів на основі мови PHP та фреймворку Symfony. Такий підхід забезпечує інтеграцію процесу збору даних із загальною модульною архітектурою застосунку та дає змогу дотримуватися принципів доменно-орієнтованого проєктування та гексагональної архітектури.

До основних компонентів PHP-стека належать:

– бібліотеки Symfony HttpClient [16] / Guzzle [9] для виконання HTTP-запитів до новинних ресурсів та отримання HTML- і RSS-документів, які надають засоби конфігурування заголовків запитів, обробки кодів відповіді, а також підтримують синхронний та асинхронний режими роботи;
– компонент Symfony DomCrawler [16] для аналізу структури HTML-сторінок та навігації DOM-деревом, що дозволяє послідовно витягувати заголовки, анотації, посилання, дати публікації та інші атрибути новинних матеріалів;
– допоміжний компонент CssSelector, що забезпечує можливість використання CSS-селекторів під час пошуку елементів DOM, що робить правила парсингу більш компактними та зрозумілими.

Застосування зазначених бібліотек дає змогу:

– організувати ефективне виконання HTTP-запитів, у тому числі з використанням асинхронних викликів для зменшення часу імпорту даних;
– гнучко адаптувати парсери під різні структури вебсторінок завдяки використанню окремих профілів для кожного джерела;
– відокремити інфраструктурні аспекти доступу до зовнішніх ресурсів від доменної логіки, що спрощує супровід і подальший розвиток системи.

Отримані за допомогою PHP-бібліотек дані перетворюються на

внутрішні транспортні об’єкти та доменні моделі, які надалі використовуються іншими модулями програмного комплексу.

2.1.2 Python-бібліотеки (BeautifulSoup, Scrapy)

Поряд із PHP-стеком доцільно розглянути типові інструменти вебскрейпінгу, що застосовуються в екосистемі Python. Хоча ці бібліотеки безпосередньо не використовуються в реалізації досліджуваного програмного комплексу, їх аналіз дає змогу порівняти підходи до організації процесу скрапінгу та окреслити можливі напрямки розширення системи.

До найпоширеніших Python-інструментів вебскрейпінгу належать:

– бібліотека **BeautifulSoup** для парсингу HTML- та XML-документів із простим і виразним API, яка забезпечує пошук елементів за тегами, атрибутами та CSS-селекторами, що робить її зручною для побудови невеликих скриптів і прототипів;
– фреймворк **Scrapy** для розроблення повноцінних вебкраулерів, орієнтований на масовий збір даних, що підтримує організацію «спайдерів», повторне використання компонентів, обробку помилок, кешування та збереження результатів у різні сховища.

Python-підхід є особливо корисним у сценаріях, де необхідний масовий скрапінг великої кількості сторінок або інтеграція з інструментами аналізу даних та машинного навчання.

У контексті даної роботи розгляд бібліотек BeautifulSoup та Scrapy виконує порівняльну функцію та може бути використаний як основа для потенційного виділення окремих мікросервісів вебскрейпінгу в майбутніх версіях системи.

Узагальнену характеристику основних інструментів вебскрейпінгу, розглянутих у цьому підрозділі, наведено в таблиці 2.1.

Таблиця 2.1 – Характеристика інструментів вебскрейпінгу

№	Екосистема / технологія	Інструмент	Основне призначення	Статус використання в проєкті
1	PHP / Symfony	Symfony HttpClient / Guzzle	Виконання HTTP-запитів до новинних ресурсів, отримання HTML- та RSS-документів	Використовується для реалізації модуля вебскрейпінгу
2	PHP / Symfony	DomCrawler	Аналіз структури HTML-сторінок, навігація DOM-деревом, витягнення потрібних елементів	Використовується для парсингу контенту новинних сторінок
3	PHP / Symfony	CssSelector	Пошук елементів DOM за CSS-селекторами, опис правил парсингу у компактній формі	Використовується як допоміжний інструмент під час парсингу
4	Python	BeautifulSoup	Парсинг HTML- та XML-документів, пошук елементів за тегами, атрибутами та селекторами	Не використовується у поточній версії, розглядається порівняльно
5	Python	Scrapy	Розроблення повноцінних вебкраулерів для масового збору даних	Не використовується у поточній версії; перспективний варіант для окремого мікросервісу

2.2 Технології розробки REST API

Теоретичні засади REST-підходу та його основні принципи розглянуто в розділі 1. У цьому підрозділі увага зосереджується на конкретному стеку технологій, які використано для реалізації REST API розроблюваного агрегатора новин, та на організації взаємодії між серверною частиною системи і клієнтськими застосунками.

Основу серверної частини становить PHP-фреймворк Symfony, який забезпечує маршрутизацію, обробку HTTP-запитів, серіалізацію відповідей та

інтеграцію з інструментами документування API. Формальний опис інтерфейсів реалізовано за допомогою специфікації OpenAPI (Swagger) [13], що спрощує інтеграцію з різними типами клієнтів (вебінтерфейс, мобільні додатки тощо).

Таким чином, у контексті REST API дана робота зосереджується на одному основному серверному застосунку (агрегаторі новин), який надає публічний інтерфейс для читання новинного контенту та пов’язаних даних. Приклади використання цього інтерфейсу клієнтськими застосунками розглядаються далі в розділі, присвяченому реалізації.

2.2.1 Власна розробка REST API

REST API агрегатора новин реалізовано як набір модулів серверної частини, що дотримуються єдиних підходів до проєктування інтерфейсів і відображають основні сутності предметної області.

Ключові рішення можна узагальнити так:

– ресурсно-орієнтований підхід, при якому кінцеві точки формулюються в термінах ресурсів («новини», «джерела», «категорії», «теги» тощо), що робить структуру API узгодженою з доменною моделлю та зрозумілою для розробників клієнтів;

– використання DTO, що передбачає застосування об’єктів передавання даних (DTO) для обміну даними між REST-шаром і внутрішніми сервісами, дозволяючи відокремити внутрішні моделі зберігання від зовнішнього формату представлення та підтримувати стабільні контракти;

– підхід *Ports and Adapters,* за якого взаємодія контролерів із бізнес-логікою здійснюється через порт-інтерфейси (контролери звертаються до абстракцій на кшталт NewsQueryServiceInterface чи NewsCommandServiceInterface), що полегшує модульність, тестування та подальший розвиток системи;

– підтримка пагінації та фільтрації, зокрема для роботи з великими обсягами новинного контенту реалізовано механізми пагінації, фільтрації та сортування за параметрами запиту;
– уніфікований формат відповідей, що мають єдину структуру, де чітко розділено корисні дані, службову інформацію (наприклад, метадані пагінації) та відомості про помилки, що спрощує обробку результатів на стороні клієнтів.

У межах самого агрегатора новин REST API слугує «фасадом» між зовнішнім світом і внутрішніми модулями, які відповідають за скрапінг, обробку та зберігання новин. Клієнтські застосунки (вебінтерфейс, мобільні додатки) бачать систему саме через цей інтерфейс.

Як приклад інтеграції можна розглядати мобільний застосунок, розроблений на Flutter. Він споживає REST API агрегатора для отримання новинного контенту, а додатково може взаємодіяти з окремим бекенд-сервісом на базі Laravel, який:

– обслуговує власних користувачів мобільного додатка;
– реалізує специфічні для клієнта сценарії (наприклад, його внутрішню систему ролей, push-сповіщення тощо);
– за потреби комбінує дані, отримані від агрегатора, з іншою інформацією.

Важливо підкреслити, що Laravel-сервіс у цьому випадку не входить до складу розроблюваного в роботі програмного комплексу. Він розглядається як самостійна стороння система, що використовує REST API агрегатора новин для власних цілей, демонструючи можливості інтеграції зовнішніх клієнтів.

Загальну картину такої взаємодії доцільно подати на схемі, де буде показано:

– серверну частину агрегатора новин із REST API;
– клієнтські застосунки (вебінтерфейс, мобільний додаток);
– за потреби – окремий клієнтський бекенд (наприклад, Laravel-сервіс) як зовнішню систему.

2.2.2 Документація API зі Swagger (OpenAPI)

Для забезпечення зручної інтеграції з REST API та зменшення витрат на ознайомлення розробників із системою використовується специфікація OpenAPI (Swagger) [13].

У проєкті інтегровано пакет NelmioApiDocBundle, що дозволяє описувати інтерфейси безпосередньо в коді та автоматично генерувати документацію.

Згенерована документація надає інтерактивний вебінтерфейс, через який розробники можуть переглядати доступні ендпоінти, їхні параметри та формати відповідей, а також виконувати тестові запити без використання сторонніх інструментів.

Основні підходи до документування такі:

– опис ресурсів, параметрів запитів і форматів відповідей виконується за допомогою анотацій у контролерах та DTO;
– на основі анотацій генерується OpenAPI-специфікація у форматі JSON, яка може використовуватися для аналізу, валідації та, за потреби, автоматичної генерації клієнтських бібліотек;
– для розробників надається інтерактивний інтерфейс Swagger UI, у якому можна переглядати структуру API, приклади запитів і відповідей та тестувати виклики.

Підтримання актуальності документації розглядається як складова процесу розробки: зміни в REST API супроводжуються оновленням анотацій, завдяки чому опис інтерфейсів залишається узгодженим із реалізацією. Це спрощує розробку і супровід клієнтських застосунків, які використовують агрегатор новин як джерело даних.

Такий підхід суттєво прискорює процес інтеграції клієнтських застосунків та забезпечує актуальність документації, оскільки вона формується безпосередньо на основі програмного коду.

2.3 Формати обміну даними

Під час проєктування агрегатора новин важливим завданням є вибір форматів обміну даними між серверною частиною, клієнтськими застосунками та зовнішніми системами. Усі зовнішні інтеграції, зокрема робота REST API та взаємодія з веб та мобільними клієнтами, у поточній реалізації ґрунтуються на використанні формату JSON як основного.

Такий вибір зумовлений низкою причин. По-перше, JSON має широке поширення й добре підтримується сучасними мовами програмування та фреймворками. По-друге, структура JSON є достатньо простою для читання як людиною, так і машиною, що полегшує налагодження та тестування. По-третє, JSON природно поєднується з REST-стилем побудови вебсервісів, де обмін даними зазвичай відбувається у вигляді серіалізованих об’єктів.

У відповідях REST API JSON-структури базуються на об’єктах передавання даних (DTO), які описують новини, джерела, категорії, теги та метадані пагінації. Детальніше структура цих відповідей розглядається далі в розділі, присвяченому реалізації програмного забезпечення, тому в цьому підрозділі зосереджено увагу саме на форматах обміну.

Поряд із JSON у системі використовується XML, насамперед у вигляді RSS-каналів під час імпорту новин з зовнішніх ресурсів. У цьому разі XML виступає вхідним форматом, який перетворюється на внутрішні моделі системи. Крім того, XML розглядається як потенційний формат експорту для сторонніх систем, що очікують саме XML-подання даних або використовують його у вже наявній інфраструктурі. Внутрішня архітектура агрегатора при цьому не орієнтується на XML як основний формат зберігання чи взаємодії між модулями.

Для майбутніх етапів розвитку системи передбачається можливість використання додаткових форматів обміну даними:

– CSV – для імпорту та експорту табличних даних, наприклад масового завантаження або вивантаження новинних записів, списків джерел чи категорій;

– NDJSON (newline-delimited JSON) – для потокової передачі великих обсягів новин, коли кожний запис передається окремим рядком, що спрощує обробку стримінгових даних;
– Protobuf / Avro – у разі переходу до високонавантажених мікросервісних інтеграцій.

У поточній реалізації система обмежується використанням JSON як основного формату для клієнтських застосунків та XML як вхідного формату під час роботи з RSS-каналами. Такий підхід є достатнім для типових сценаріїв веб та мобільних клієнтів і водночас залишає можливість подальшого розширення набору форматів за потреби.

Узагальнену характеристику форматів обміну даними, розглянутих у цьому підрозділі, наведено в таблиці 2.2.

Таблиця 2.2 – Формати обміну даними в агрегаторі новин

Формат	**Основне призначення**	**Статус у системі**
JSON	Обмін даними між REST API та клієнтськими застосунками	Основний формат
XML (RSS)	Отримання новин із зовнішніх джерел, можливий експорт	Використовується як вхідний / опційний
CSV	Масовий імпорт/експорт табличних даних	Перспективний формат
NDJSON	Потокова передача великих обсягів новин	Перспективний формат
Protobuf / Avro	Ефективний обмін у високонавантажених мікросервісних інтеграціях	Перспективний формат

2.4 Допоміжні інструменти розробника

Окрім основних технологій, безпосередньо пов’язаних із вебскрейпінгом та реалізацією REST API, у процесі розробки агрегатора новин використовувався набір допоміжних інструментів. Вони забезпечують керування вихідним кодом, організацію командної роботи, зручне середовище розробки та підтримку процесу тестування.

До ключових допоміжних засобів належать:

– система керування версіями Git;
– інтегровані середовища розробки (IDE) з підтримкою PHP та Docker;
– інструмент для тестування HTTP-запитів **Postman**, який доповнює вбудовані засоби перевірки REST API.

Таке поєднання інструментів дозволяє підтримувати структуровану історію змін, автоматизувати частину перевірок якості коду та спростити налагодження взаємодії клієнтів із REST API.

2.4.1 Git та інтегровані середовища розробки (IDE)

Для керування історією вихідного коду та організації роботи над проєктом використовується система контролю версій Git. Як платформу для хостингу репозиторіїв обрано GitLab, що поєднує функції віддаленого сховища коду з вбудованими інструментами для організації робочого процесу. Репозиторій містить як код основного Symfony-проєкту агрегатора новин, так і код клієнтських застосунків – мобільного та вебклієнта, – які слугують демонстраційними рішеннями для взаємодії з REST API системи. GitLab надає можливість налаштування процесів безперервної інтеграції та доставки (CI/CD) через файл .gitlab-ci.yml.

Застосування Git у поєднанні з GitLab забезпечує:

– збереження відтворюваної історії змін та можливість повернення до попередніх версій;
– контроль якості коду за рахунок поступового введення перевірок на рівні комітів і гілок;
– зручну організацію роботи над кількома компонентами системи (бекенд, клієнтські застосунки);
– автоматизувати запуск модульних тестів, статичного аналізу та інших перевірок якості коду при кожному коміті або злитті гілок.

У процесі розробки використовувались кілька інтегрованих середовищ розробки, кожне з яких було підібрано з огляду на тип проєкту:

– середовище **PhpStorm**, що використовувалося для роботи з основним бекендом на Symfony [16] (агрегатор новин) та окремим бекенд-сервісом на Laravel [10], який слугує серверною частиною для мобільного клієнта, оскільки воно забезпечує розвинену підтримку PHP, фреймворків Symfony і Laravel, інтеграцію з Git, засоби навігації по коду та запуску тестів;
– редактор **Visual Studio Code**, який застосовувався для розробки мобільного застосунку на Flutter [7] з використанням відповідних розширень для мови Dart і фреймворку Flutter, а також для другого клієнта – вебсайту на базі Next.js [11], який реалізує інтерфейс для читання новин, адже набір плагінів для JavaScript/TypeScript і React/Next.js у VS Code спрощує роботу з фронтенд-частиною системи.

Такий розподіл інструментів дозволяє використовувати сильні сторони кожного IDE залежно від технологічного стека: PhpStorm – для глибокої роботи з PHP-бекендом, VS Code – для кросплатформеної розробки клієнтських застосунків на Flutter і Next.js. Детальний опис клієнтських застосунків (мобільного та вебклієнта) подається далі в роботі, у розділі, присвяченому реалізації інтерфейсів користувача.

2.4.2 Postman та тестування API

Для ручного та напівавтоматизованого тестування REST API агрегатора новин застосовується інструмент Postman. Він використовується як доповнення до автоматично згенерованої документації Swagger/OpenAPI та дозволяє детально перевіряти поведінку API в типових і нестандартних сценаріях.

У межах проєкту в Postman сформовано окремі колекції запитів до основних кінцевих точок REST API. Колекції містять приклади:

– запитів на отримання списку новин з параметрами пагінації та фільтрації;
– запитів до ресурсів джерел, категорій та тегів;
– запитів, що потребують автентифікації або використання токенів доступу.

Збережені приклади відповідей дозволяють швидко порівнювати фактичну поведінку API з очікуваною, а також полегшують комунікацію між розробниками серверної частини та клієнтських застосунків. Postman зручний і для перевірки складніших сценаріїв, коли потрібно послідовно виконати кілька запитів із передаванням проміжних даних між ними.

У поєднанні зі Swagger UI Postman відіграє роль практичного інструмента для налагодження й верифікації REST API під час розробки та тестування.

2.5 Засоби забезпечення якості коду

2.5.1 Статичний аналіз коду та архітектури

Для підтримання належної якості коду агрегатора новин у проєкті застосовується набір інструментів статичного аналізу. Їх використання дає змогу виявляти помилки та небажані залежності на ранніх етапах розробки, ще до виконання коду, а також підтримувати узгоджений стиль програмування в межах усього репозиторію.

У контексті даної роботи статичний аналіз охоплює декілька аспектів.

По-перше, контроль стилю та базових помилок у коді здійснюється за допомогою інструментів **PHP_CodeSniffer** та **PHPMD (PHP Mess Detector)**. PHP_CodeSniffer перевіряє відповідність коду вибраному набору правил оформлення (code style), тоді як PHPMD виявляє потенційно проблемні

конструкції, надлишкову складність методів, «мертвий» код та інші ознаки погіршення якості.

По-друге, аналіз типів і пошук прихованих помилок виконуються за допомогою **PHPStan**. Завдяки обов'язковій типізації в сигнатурах методів і коментарях PHPStan дозволяє виявляти несумісності типів, звернення до неіснуючих властивостей або методів, а також інші помилки, які не завжди проявляються під час виконання модульних тестів.

По-третє, для підтримання актуальності та однорідності коду використовується інструмент **Rector**, орієнтований на автоматизований рефакторинг. Він застосовується для поступового переведення коду на більш сучасні версії PHP, спрощення конструкцій, уніфікації підходів до обробки винятків, роботи з колекціями тощо. Це особливо актуально для модульної структури проєкту, де переважно використовується DDD- та Ports and Adapters-підхід.

По-четверте, для контролю архітектурних залежностей застосовується інструмент **Deptrac**. Він дозволяє формально задати правила взаємодії між шарами та модулями (наприклад, щоб доменний шар не залежав від інфраструктури, а REST-контролери не зверталися безпосередньо до репозиторіїв). Порушення цих правил фіксуються під час аналізу, що допомагає зберігати заявлену архітектуру (DDD, гексагональна архітектура, CQRS) не лише на рівні діаграм, а й у реальному коді.

Усі перелічені інструменти інтегровано в процес **GitLab CI/CD**. Репозиторій основного Symfony-проєкту розміщено на платформі GitLab, що дало змогу:

- організувати централізований запуск PHP_CodeSniffer, PHPMD, PHPStan, Rector та Deptrac разом із PHPUnit-тестами при кожному пуші до основних гілок;
- запобігати потраплянню у головну гілку змін, які порушують стилістичні вимоги, архітектурні обмеження або призводять до зниження якості коду;

– зберігати результати перевірок у вигляді звітів, доступних безпосередньо з інтерфейсу GitLab.

Такий підхід дозволяє розглядати якість коду не як разову дію на етапі завершення проєкту, а як безперервний процес, інтегрований у щоденний цикл розробки. Детальні приклади використання статичних аналізаторів і сценаріїв CI/CD наводяться далі в розділі, присвяченому реалізації та тестуванню програмного забезпечення.

2.5.2 PHPUnit та Behat як засоби тестування

Автоматизоване тестування є важливою складовою розробки програмного забезпечення, оскільки дає змогу своєчасно виявляти регресії та перевіряти коректність роботи ключових компонентів системи. У межах розробки агрегатора новин як основний інструмент модульного й інтеграційного тестування PHP-коду використано фреймворк PHPUnit [14], а як перспективний засіб поведінкового тестування розглянуто Behat [5].

У проєкті PHPUnit застосовується для перевірки:

– доменної логіки та сервісів, у тому числі CQRS-хендлерів;

– компонентів, пов’язаних із парсингом новинних джерел;

– базової поведінки REST-контролерів і форматів відповідей API.

Тести виконуються як локально під час розробки, так і в рамках процесу безперервної інтеграції: при кожному оновленні основної гілки репозиторію запускається набір PHPUnit-тестів із формуванням звітів покриття. Це дозволяє контролювати, які частини системи перевірено найкраще, та своєчасно виявляти ділянки, що потребують додаткового тестового покриття.

Фреймворк Behat розглядається як інструмент поведінкового тестування (BDD), орієнтований на опис сценаріїв роботи системи мовою, наближеною до природної. Для агрегатора новин такі сценарії можуть відображати типові варіанти використання, зокрема:

– отримання списку новин з пагінацією;

– фільтрацію новин за джерелом чи категорією;
– запуск імпорту новин з подальшою перевіркою результату.

Інтеграція Behat із Symfony забезпечується через розширення, які надають доступ до сервіс-контейнера, що дозволяє тестувати як HTTP-взаємодію через API, так і внутрішню логіку застосунку. Це робить Behat особливо корисним для наскрізного тестування, де перевіряється коректність роботи системи від запиту користувача до формування відповіді.

У поточній версії програмного комплексу основний акцент зроблено на модульних та інтеграційних тестах на базі PHPUnit. Behat на цьому етапі розглядається як перспективний засіб для формалізації та автоматизації найбільш важливих бізнес-сценаріїв; базову інфраструктуру для BDD-тестів планується розширювати в подальшому разом із розвитком системи.

Детальні приклади побудови тестів і результати їх виконання наводяться далі у розділі, присвяченому реалізації та тестуванню програмного забезпечення.

Узагальнену структуру рівнів автоматизованого тестування, що застосовуються під час розробки агрегатора новин (модульні та інтеграційні тести на базі PHPUnit, а також перспективні поведінкові тести Behat), наведено на рисунку 2.2.



Рисунок 2.2 – Узагальнена схема рівнів автоматизованого тестування

2.6 Висновки з розділу 2

У цьому розділі узагальнено інструменти та технології, що використовуються під час розробки модульного агрегатора новин. Описано засоби вебскрейпінгу (з акцентом на PHP-стек Guzzle та Symfony DomCrawler), підхід до побудови REST API на базі Symfony та OpenAPI/Swagger, а також формати обміну даними, серед яких основним є JSON, а RSS/XML застосовується для інтеграції з новинними джерелами.

Крім того, розглянуто допоміжні засоби розробки – Git із розміщенням репозиторію на GitLab, середовища PhpStorm та Visual Studio Code, інструмент Postman – та комплекс засобів забезпечення якості коду (PHP_CodeSniffer, PHPMD, PHPStan, Rector, Deptrac у поєднанні з PHPUnit і, за потреби, Behat), інтегрованих у процес CI/CD. Сукупність цих рішень формує цілісний технологічний стек, що створює основу для подальшого проєктування архітектури системи та структури бази даних, які розглядаються у наступному розділі.

3 РЕАЛІЗАЦІЯ МОДУЛЬНОГО АГРЕГАТОРА НОВИН

3.1 Архітектура застосунку та взаємодія модулів

Архітектура розробленого агрегатора новин побудована як модульний моноліт на базі фреймворку Symfony. Такий підхід поєднує єдиний виконуваний застосунок із чітким поділом на предметно орієнтовані модулі, що взаємодіють між собою через порт-інтерфейси. У межах одного проєкту реалізовано окремі підсистеми вебскрейпінгу, інтеграції із засобами штучного інтелекту, публічного REST API та адміністративного вебінтерфейсу. Застосовано принципи предметно-орієнтоване проєктування, гексагональної архітектури та патерну CQRS, розглянуті у попередніх розділах роботи.

Рисунок 3.1 ілюструє загальну структуру системи та взаємодію агрегатора новин із клієнтськими застосунками через REST API.

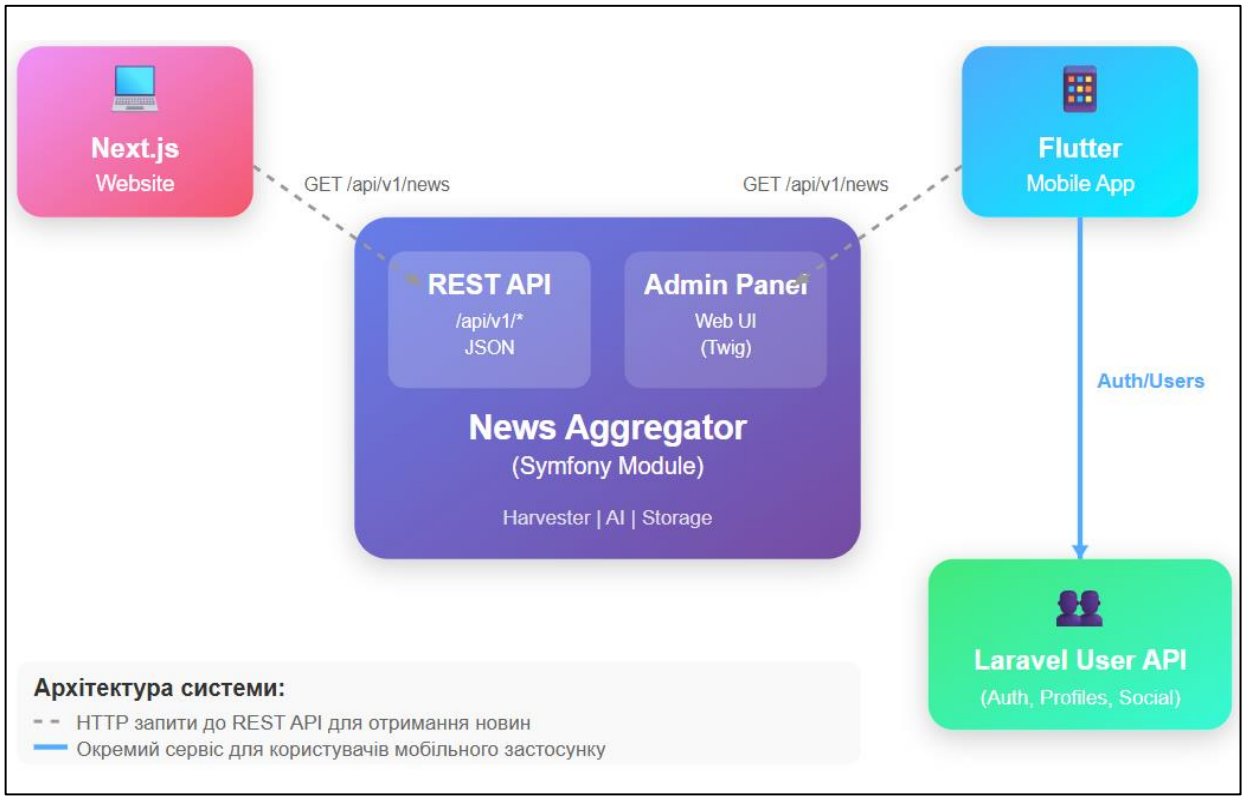


Рисунок 3.1 – Загальна архітектура агрегатора новин та клієнтських застосунків

3.1.1 Основні модулі

Ядро системи реалізовано як модульний моноліт Symfony, у межах якого виділено кілька ключових модулів.

Harvester – модуль збору новин з RSS- та HTML-джерел. Він відповідає за планування й запуск процесів скрапінгу, надсилання HTTP-запитів (через Guzzle або Symfony HttpClient), парсинг HTML- та XML-документів за допомогою DomCrawler та CssSelector, нормалізацію даних у доменні моделі й збереження їх у базі даних PostgreSQL [15] через Doctrine ORM. Запуск Harvester здійснюється за допомогою консольних команд, які прив'язуються до cron-завдань. Інтервали запуску є «плаваючими», а рядок User-Agent ротується, що зменшує ризик блокування з боку зовнішніх ресурсів.

AI-модуль – підсистема інтеграції із сервісами штучного інтелекту. Її завданням є виконання операцій перекладу новин, формування короткого резюме (summarization), автоматичного тегування та допоміжної обробки тексту. На рівні домену використовується узагальнена модель «AI-модель» із зазначенням призначення (translation, summarization, auto-tagging, а також експериментальна мета – генерація шаблонів парсерів), конфігураційних параметрів та стану активності. На інфраструктурному рівні реалізовані адаптери до конкретних провайдерів. Наразі використовується один провайдер – сервіс OpenAI [12], однак завдяки виділенню інтерфейсів провайдерів і моделей, у майбутньому можна додати інші сервіси без модифікації бізнес-логіки.

REST-модуль – модуль, що надає публічний REST API й вебінтерфейси керування. Для зовнішніх клієнтів він експонує HTTP-ендпоінти /api/v1/* для доступу до списку новин, джерел, категорій, тегів та результатів AI-обробки, повертаючи відповіді у форматі JSON. Документація до API формується у форматі OpenAPI/Swagger [13] (NelmioApiDocBundle [16]), що спрощує інтеграцію з боку сторонніх розробників. Окрім власне API, REST-модуль містить **публічну привітальну сторінку** (Web UI на базі Twig-шаблонів), на якій розміщено загальну інформацію про сервіс і посилання на документацію

Swagger. Через цю сторінку користувач може зареєструватися, увійти в систему та, перебуваючи в автентифікованому стані, згенерувати для себе токен доступу (API-ключ) до REST API. У тому ж вебінтерфейсі реалізовано **адміністративну панель** (Admin Panel на рисунку 3.1), яка використовується для керування користувачами, їхніми API-токенами, журналами використання API, AI-моделями та джерелами новин.

Shared-модуль – набір спільних компонентів, що використовуються кількома підсистемами. До нього належать DTO, допоміжні сервіси, інтерфейси репозиторіїв, базові класи винятків і утилітарні функції. Завдяки цьому загальна логіка й типові структури даних не дублюються між модулями Harvester, AI та REST.

З метою розділення відповідальності та спрощення масштабування система використовує **дві окремі бази даних PostgreSQL**. База даних *news-harvester* зберігає новинні сутності, пов’язані з ними категорії, теги та артефакти, пов’язані з AI-обробкою (моделі, журнали використання). Друга база даних *news-rest* належить модулю REST API й містить інформацію про користувачів вебінтерфейсу, видані ключі доступу (API keys) та журнали звернень до API. Такий поділ дозволяє ізолювати доменні дані агрегатора від облікових записів і технічних логів зовнішніх клієнтів.

Окремо від Symfony-застосунку існують зовнішні клієнтські системи, які використовують REST API агрегатора.

news-user-api (Laravel [10]) – окремий бекенд для мобільного застосунку, що відповідає за управління користувачами, коментарями, лайками та списками «обраного». Цей сервіс має власну базу даних і власні бізнес-правила. Агрегатор новин для нього виступає лише джерелом даних: новини отримуються через REST API з використанням виданого API-ключа, а інформація про лайки та коментарі зберігається виключно у Laravel-бекенді й не впливає на основну схему даних агрегатора.

news-web-client (Next.js [11]) – вебсайт, який реалізує роль «ридера» новин. Він працює як типова клієнтська SPA/SSR-застосунок: отримує новини через REST API, дозволяє фільтрувати їх за джерелами й категоріями,

застосовувати переклад і переглядати короткі резюме. Для доступу до API використовується окремий API-ключ, згенерований у REST-модулі.

news-mobile-app (Flutter [7]) – мобільний застосунок для Android, який також споживає REST API агрегатора. Авторизація користувачів, збереження їхніх профілів, коментарів і лайків реалізовано у власному бекенді на Laravel (news-user-api). Таким чином, мобільний клієнт поєднує дані з двох джерел: новинний контент – з агрегатора, а соціальну активність – із окремої системи управління користувачами.

3.1.2 Взаємодія між модулями

Взаємодія між внутрішніми модулями агрегатора (Harvester, AI, REST) організована відповідно до принципів гексагональної архітектури. Кожен модуль експонує порт-інтерфейси, які реалізуються у відповідному доменному або інфраструктурному шарі, тоді як Symfony виступає засобом компонування цих реалізацій через контейнер залежностей.

На стороні REST-модуля HTTP-запити від клієнтів обробляються контролерами Symfony, які не звертаються безпосередньо до бази даних або інфраструктурних сервісів. Натомість вони працюють із прикладними сервісами, визначеними через інтерфейси на кшталт NewsQueryServiceInterface, NewsCommandServiceInterface, AIModelQueryServiceInterface тощо. Такі сервіси реалізують вхідні порти (input ports) системи й координують виконання команд і запитів у межах патерну CQRS: від простих операцій читання списку новин до виклику AI-операцій для перекладу та резюмування.

У модулі Harvester відповідні інтерфейси реалізуються сервісами, що інкапсулюють логіку імпорту новин. Вони делегують виклики CQRS-хендлерам та репозиторіям, які працюють з доменними моделями та таблицями PostgreSQL. Запуск Harvester відбувається через CLI-команди Symfony, прив’язані до cron-завдань. Для зменшення навантаження на

зовнішні ресурси й уникнення підозрілих шаблонів трафіку застосовується плаваючий графік запуску імпорту та ротація User-Agent під час HTTP-запитів. Отримані й нормалізовані новини зберігаються у спільній базі даних, після чого можуть бути додатково оброблені AI-модулем.

AI-модуль реалізує окрему групу вихідних портів (output ports), які описують операції перекладу, тегування, резюмування та допоміжні сценарії, пов’язані з аналізом структури новинних документів для генерації шаблонів парсерів. На рівні домену визначено сутність «AI-модель», яка зберігає параметри конкретної моделі (призначення, обмеження довжини відповіді, параметри генерації тощо) та її зв’язок із постачальником. На рівні інфраструктури реалізовано адаптер для провайдера OpenAI, що бере на себе перетворення внутрішніх запитів на виклики зовнішнього API та обробку отриманих відповідей. Завдяки абстракції провайдерів і моделей у майбутньому можна додати нові AI-сервіси, не змінюючи контракти модулів Harvester та REST.

Адміністративний вебінтерфейс REST-модуля також використовує ці порти. Через нього адміністратор може:

– переглядати список користувачів, виданих API-ключів і журнал їх використання;
– конфігурувати доступні AI-моделі (активувати, деактивувати, змінювати параметри й призначення – переклад, резюмування, авто-тегування, генерація парсерів);
– працювати з джерелами новин (джерела, їхні категорії, технічні налаштування).

Для додавання нового джерела використовується покроковий «майстер», який допомагає послідовно ввести базові дані про ресурс, налаштувати категорії та підготувати початковий шаблон парсер-класу. На завершальному етапі інтегровано експериментальну AI-підтримку: система аналізує зразок зовнішнього документа й пропонує заготовку парсера, яку розробник може доопрацювати вручну. Навіть якщо автоматичний аналіз не дає ідеального результату, у базі даних фіксуються всі необхідні описові дані

про джерело, а створений шаблон класу слугує відправною точкою для подальшої ручної адаптації.

Зовнішні клієнти – вебсайт на Next.js та мобільний застосунок на Flutter – сприймають агрегатор як «чорну скриньку» із чітко визначеним REST API. Кожен клієнт використовує власний API-ключ, отриманий через публічний вебінтерфейс REST-модуля. Завдяки цьому зміни у внутрішній взаємодії між Harvester, AI та REST (наприклад, виділення окремого сервісу для скрапінгу чи підключення нового AI-провайдера) не потребують змін у клієнтських застосунках, доки зберігається стабільність контрактів REST API.

У підсумку взаємодія модулів побудована таким чином, щоб мінімізувати зв’язність між внутрішньою реалізацією й зовнішніми інтерфейсами. Порти та адаптери забезпечують ізоляцію доменної логіки, а модульний моноліт на Symfony створює основу для поступового переходу до мікросервісної архітектури без радикальної зміни загальної структури системи.

3.2 Реалізація модуля вебскрейпінгу

Модуль вебскрейпінгу (Harvester) відповідає за автоматизований збір новин із зовнішніх HTML- та RSS-джерел, їх перетворення у внутрішні доменні моделі та збереження в реляційній базі даних. У межах модульного моноліту він реалізований як окремий модуль, що взаємодіє з рештою системи через чітко визначені порти застосунку та доменний шар.

Реалізація Harvester спирається на архітектурні підходи DDD, гексагональну архітектуру та патерн CQRS. Основна бізнес-логіка імпорту новин зосереджена у доменному та прикладному шарах, тоді як робота з HTTP-ресурсами, парсинг HTML/RSS та доступ до бази даних винесені в інфраструктурні адаптери. Запуск процесу збору даних відбувається через CLI-команди Symfony, які можуть ініціюватися вручну або плануватися за допомогою cron-завдань.

Модуль підтримує два основні типи джерел: RSS-стрічки та HTML-сторінки. Для RSS-джерел застосовується стандартний XML-парсинг із витягуванням метаданих публікацій, тоді як HTML-джерела потребують індивідуальних парсерів, налаштованих під структуру кожного конкретного сайту. Конфігурація джерел зберігається в базі даних у вигляді сутностей Source та SourceLink, що дозволяє додавати нові джерела без модифікації програмного коду.

Для зменшення навантаження на зовнішні ресурси модуль використовує плаваючий розклад запуску та випадкові затримки між HTTP-запитами. Крім того, при кожному зверненні до джерела випадковим чином обирається User-Agent, що знижує ризик блокування з боку цільових сайтів. Асинхронне завантаження контенту реалізовано з використанням HTTP-клієнта Symfony, який підтримує паралельну обробку кількох запитів із контролем таймаутів та автоматичним повторенням у разі тимчасових збоїв.

3.2.1 Алгоритми парсингу новинних ресурсів та нормалізація даних

Основний робочий процес імпорту новин реалізовано як CLI-команди Symfony, які викликають відповідні CQRS-команди. Для кожного джерела передбачено окрему команду, наприклад:

– bin/console app:harvester:import:news korrespondent;

– bin/console app:harvester:import:news unian.

Далі розглядається послідовність кроків яку ініціює виконання такої команди.

Формування команди ImportNewsCommand. Презентаційний шар (CLI-команда) створює об’єкт ImportNewsCommand, у якому зазначаються ідентифікатор джерела, режим імпорту та додаткові параметри (обмеження за датою, максимальною кількістю новин тощо).

Обробка команди ImportNewsHandler. Прикладний шар отримує команду та передає її до обробника ImportNewsHandler, який координує

подальші дії:

- завантажує з репозиторію список об’єктів SourceLink, що описують RSS-стрічки або HTML-сторінки конкретного джерела;
- звертається до сервісу, що реалізує інтерфейс NewsFetcherInterface, для асинхронного завантаження вмісту за всіма посиланнями з урахуванням випадкових затримок і обраного User-Agent;
- передає отримані «сирі» дані (HTML, XML, JSON) до фабрики парсерів NewsParserFactory.

Парсинг і побудова доменних моделей. Фабрика NewsParserFactory обирає конкретний парсер залежно від джерела (наприклад, KorrespondentNewsParser, UnianNewsParser), які наслідують базовий клас BaseNewsParser. Кожен парсер:

- аналізує структуру документа за допомогою DomCrawler та CSS-селекторів;
- витягує основні поля новини (заголовок, короткий опис, повний текст, посилання, дату публікації, зображення, категорію);
- формує доменну сутність News та пов’язані з нею об’єкти (джерело, категорії, теги).

Нормалізація та валідація даних. Щоб забезпечити єдину структуру новин незалежно від джерела, у модулі застосовується низка перетворень:

- усі дати публікації перетворюються до типу DateTimeImmutable з урахуванням часових поясів;
- текстові поля очищуються від службової HTML-розмітки, скриптів і непотрібних елементів верстки;
- мова новини явно фіксується у відповідному полі (наприклад, language), що спрощує подальшу AI-обробку;
- за потреби URL-адреси зображень та зовнішніх посилань нормалізуються до абсолютної форми.

Дедуплікація та збереження в базі даних. Сформовані доменні сутності News передаються до репозиторію, який:

– обчислює для кожної новини унікальний хеш (наприклад, SHA-256 від основного контенту) або використовує зовнішній contentId із джерела;
– перевіряє наявність записів із тим самим хешем/ідентифікатором і відкидає дублікати;
– зберігає нові записи пакетно (batch-вставки), що зменшує кількість транзакцій і підвищує продуктивність роботи з PostgreSQL.

Підготовка даних до подальшої обробки. Після успішного збереження новин у базі даних система формує внутрішні події або позначки стану, які використовуються AI-модулем для запуску операцій перекладу, резюмування та автоматичного тегування.

Таким чином, процес вебскрейпінгу завершується формуванням узгоджених доменних моделей, готових до подальшого збагачення та публікації через REST API. Узагальнену послідовність описаних етапів – від запуску CLI-команди до запису нормалізованих даних у базу та передачі їх до AI-модуля – представлено на рисунку 3.2 (стор. 56).

На кожному етапі процесу передбачено механізми обробки помилок: у разі недоступності джерела, некоректної структури HTML-документа або перевищення таймауту система фіксує відповідну інформацію в журналі подій та продовжує обробку інших посилань. Це забезпечує стійкість процесу імпорту навіть за умов нестабільної роботи зовнішніх ресурсів.

3.2.2 Запис даних у базу та потік AI-обробки новини

Для збереження результатів вебскрейпінгу використовується реляційна база даних PostgreSQL. Модуль Harvester працює з окремою базою *news-harvester*, у якій зосереджені всі доменні сутності, пов’язані з новинним контентом та його AI-обробкою. Такий підхід спрощує ізоляцію даних від облікових записів користувачів та технічних логів REST-модуля, що зберігаються в іншій базі даних.

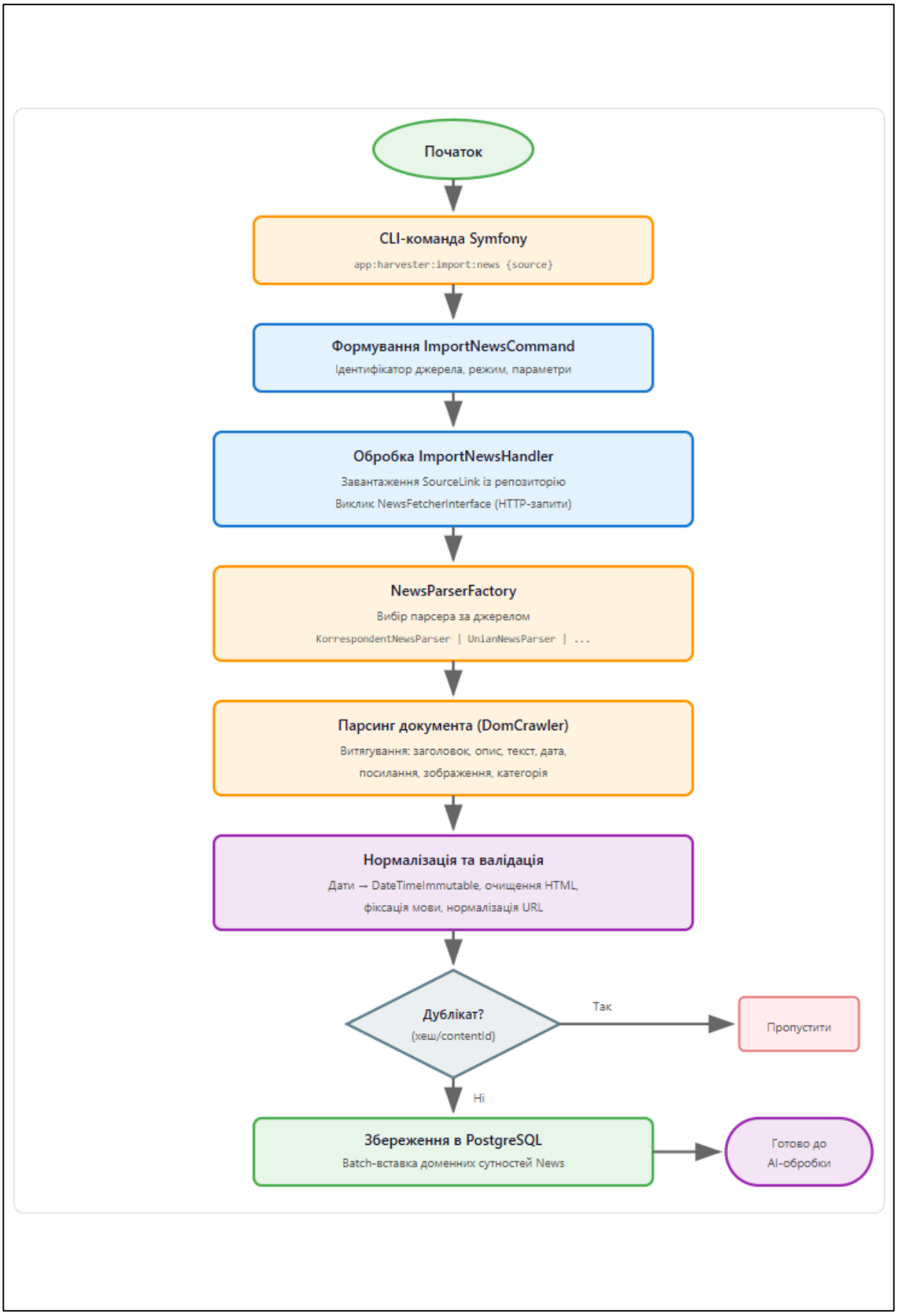


Рисунок 3.2 – Узагальнена блок-схема процесу імпорту новин у модулі вебскрейпінгу

Під час проєктування схеми БД для модуля Harvester було застосовано такі принципи:

– ULID як первинні ключі;
– нормалізована схема;
– дедуплікація новин.

ULID як первинні ключі. Для більшості таблиць використовуються ULID-ідентифікатори. Вони забезпечують унікальність записів, зручне сортування за часом створення та добре підходять для потенційно розподілених сценаріїв, коли записи можуть створюватися на різних вузлах системи.

Нормалізована схема. Окремі аспекти предметної області винесено в окремі таблиці: новини (news), джерела (source), категорії (category), зв’язки між джерелами й категоріями, теги та зв’язки «новина – тег». Це дозволяє уникнути дублювання даних, спростити модифікацію окремих довідників і підвищити узгодженість інформації в системі.

Дедуплікація новин. Перед вставкою нових записів репозиторій отримує список уже існуючих ідентифікаторів вмісту (contentId) або обчислює хеш (наприклад, SHA-256) від основного тексту новини. Якщо для вибраного джерела знайдено збіг, новина вважається дублем і не вставляється повторно. Це запобігає накопиченню однакових матеріалів у разі повторної публікації або тимчасових збоїв під час імпорту.

Узагальнену логічну структуру бази даних news-harvester буде докладніше розглянуто в окремому підрозділі розділу 3, де наведено ER-діаграму основних сутностей і зв’язків між ними.

Окрему увагу в схемі бази даних приділено зберіганню результатів AI-обробки новин. Для цього, крім базової таблиці *news*, використовуються додаткові сутності, які дозволяють:

– зберігати переклади новин різними мовами (зокрема заголовок, короткий опис і повний текст);
– фіксувати згенеровані теги з ознакою того, чи створені вони автоматично, чи додані вручну;

– журналювати використання AI-сервісу (яка модель застосовувалася, для якої новини, скільки токенів витрачено, який статус операції).

Такий підхід дає змогу:

– чітко відокремити «оригінальний» контент, отриманий під час вебскрейпінгу, від похідного контенту, згенерованого засобами штучного інтелекту;
– у будь-який момент змінити активну AI-модель або перегенерувати переклад, резюме чи теги без втрати історії попередніх результатів;
– формувати статистичні звіти щодо навантаження на AI-сервіс, кількості запитів, витрат токенів та ефективності різних конфігурацій моделей.

AI-функціонал інтегровано в життєвий цикл новини після етапу імпорту та збереження в базі даних. Типовий сценарій виглядає так:

– новина імпортується з джерела й потрапляє до таблиці news у вигляді оригінального контенту з мовою джерела;
– адміністратор у вебінтерфейсі відкриває сторінку новини та бачить, які переклади, теги та резюме вже існують, а які ще не згенеровано;
– за потреби він запускає операцію перекладу, тегування чи резюмування – для окремої новини або пакетно для вибраної добірки матеріалів;
– REST-модуль формує відповідні команди застосунку, які передаються до AI-модуля; останній звертається до підключеного AI-провайдера (наразі OpenAI), отримує результат і зберігає його в пов’язаних таблицях бази даних разом із записом у журналі використання моделі;
– оновлені дані (переклад, резюме, теги) стають доступними як через адміністративний інтерфейс, так і через публічний REST API, де вони відображаються як додаткові поля у відповідях на запити до новин.

Таким чином, модуль вебскрейпінгу не лише записує нормалізовані дані до реляційної бази, а й інтегрований із AI-модулем у єдиний потік обробки:

від первинного імпорту до збагаченого контенту, готового для подальшого використання клієнтськими застосунками.

3.3 Розробка REST API

Модуль REST API забезпечує зовнішній доступ до даних агрегатора новин та відіграє роль «фасаду» системи для веб і мобільних клієнтів. Він реалізований як окремий модуль у складі модульного моноліту Symfony та дотримується принципів гексагональної архітектури: HTTP-контролери виступають адаптерами презентаційного шару, тоді як доменна логіка доступу до новин, джерел, категорій, тегів та результатів AI-обробки інкапсульована у прикладних сервісах і порт-інтерфейсах.

Окрім безпосередньо REST API, модуль містить вебінтерфейс, який виконує дві функції.

По-перше, це публічна привітальна сторінка з описом сервісу та посиланням на інтерактивну документацію Swagger.

По-друге, це простий кабінет користувача, де можна зареєструватися, увійти в систему, згенерувати особистий API-ключ та отримати доступ до адміністративних можливостей (адмін-панель).

Таким чином, REST-модуль поєднує API-рівень та зручний вебінтерфейс керування доступом і AI-конфігураціями.

3.3.1 Структура API та маршрути

Публічний REST API організовано за принципами ресурсно-орієнтованого підходу з явним версіонуванням маршрутів. Усі зовнішні ендпоінти згруповано під префіксом /api/v1, що дозволяє в майбутньому розвивати інтерфейс без порушення сумісності з наявними клієнтами.

На рівні маршрутизації виділено кілька логічних груп:

– група новин (/api/v1/news…) – отримання списку новин, детальної інформації, пошуку та вибірки останніх публікацій;
– довідникові ресурси (/api/v1/sources, /api/v1/categories, /api/v1/tags) – дані, які використовуються для фільтрації та навігації в клієнтських застосунках;
– допоміжні операції з AI (наприклад, маршрути на кшталт /api/v1/news/{id}/translate) – ініціювання перекладу та іншої AI-обробки для конкретної новини.

Технічно маршрути описуються засобами Symfony (атрибути або конфігурація маршрутизатора) та прив’язуються до контролерів REST-модуля.

Кожен контролер відповідає лише за один тип задач: запити читання (query-контролери) або команди змін стану (command-контролери). Це відповідає підходу CQRS і спрощує розділення читання та запису в коді.

Контролери не містять бізнес-логіки й працюють виключно як адаптери HTTP-рівня. Вони викликають порти прикладного шару, наприклад *NewsQueryServiceInterface* для операцій читання та відповідні інтерфейси командних сервісів для операцій змін стану. Реалізації цих інтерфейсів інкапсулюють доступ до доменних сервісів і сховища даних, дотримуючись принципів гексагональної архітектури.

Для всіх груп маршрутів автоматично генерується специфікація **OpenAPI**. На її основі формується інтерактивна документація **Swagger UI**, доступна за окремою URL-адресою та з головної сторінки вебінтерфейсу. Приклад згенерованої інтерактивної документації, наведено на рисунку 3.3.

Розробник може обрати потрібний ендпоінт, переглянути структуру запиту й відповіді та виконати тестовий виклик безпосередньо з браузера, використовуючи свій API-ключ.

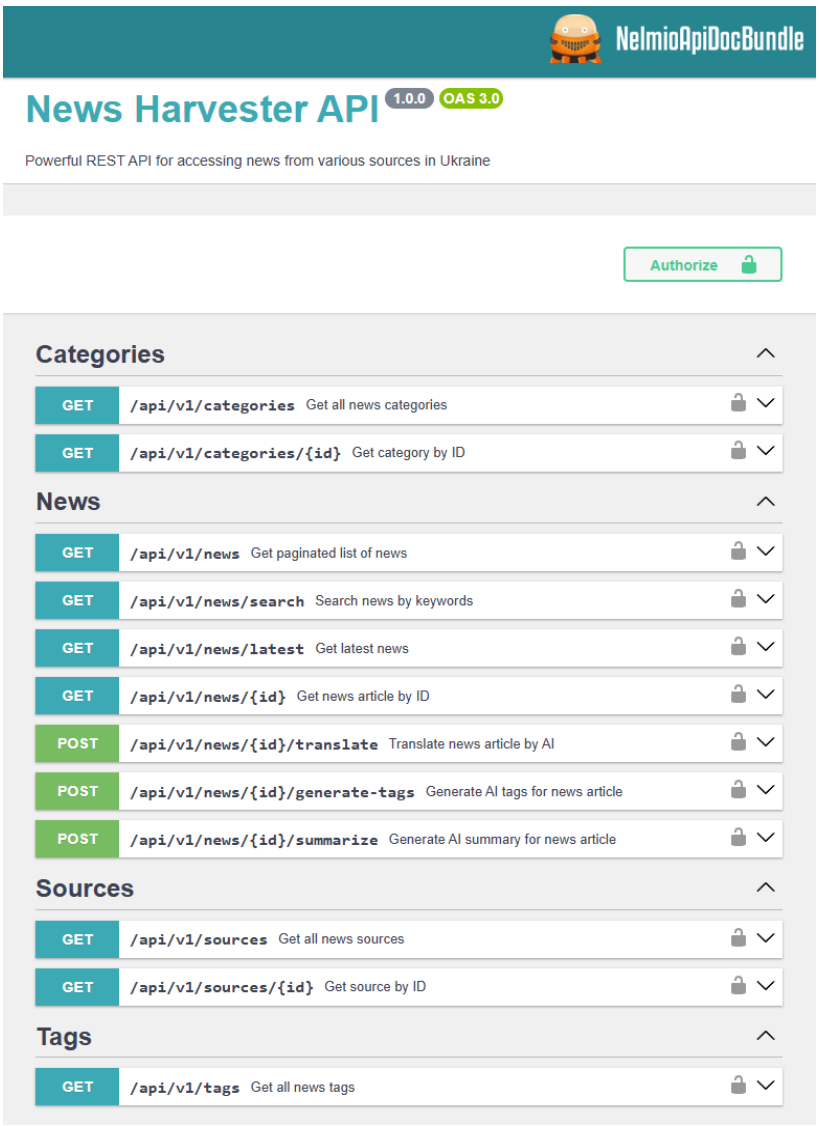


Рисунок 3.3 – Інтерактивна документація REST API в Swagger UI

3.3.2 Методи та ресурси

Ресурси REST API побудовано навколо основних сутностей предметної області: новини, джерела, категорії, теги, а також пов’язаних із ними результатів AI-обробки. Для кожного ресурсу реалізовано типові сценарії читання, які є передбачуваними для клієнтських застосунків.

Для ресурсу «новини» базовим є виклик на кшталт `GET /api/v1/news`. Він повертає колекцію новин, до якої можуть застосовуватися параметри пагінації та фільтрації (номер сторінки, розмір сторінки, джерело, категорія, мова тощо). Відповідь містить:

– масив елементів у полі *data*, де кожен елемент є DTO з полями, необхідними для побудови списку новин;
– об’єкт *meta* з інформацією про поточну сторінку, загальну кількість записів і допоміжними атрибутами.

Детальний перегляд новини реалізовано окремим запитом GET /api/v1/news/{id}, який повертає один DTO з повним набором полів: заголовок, текст, метадані, посилання на джерело, інформацію про переклади, теги та резюме (якщо вони вже згенеровані AI-модулем). Такий поділ дозволяє оптимізувати структуру відповідей для списків і детальних переглядів.

Довідникові ресурси – джерела, категорії, теги – надаються у вигляді колекцій через запити GET /api/v1/sources, GET /api/v1/categories, GET /api/v1/tags. Їхня структура спрощена: кожен елемент містить стабільний ідентифікатор, людинозрозумілу назву та, за потреби, додаткові атрибути (наприклад, «slug» для формування URL або групування). Ці ресурси зручно використовувати у клієнтських застосунках для формування фільтрів, меню та вибірок.

Окремий клас методів пов’язаний із AI-обробкою новин. На рівні REST API такі методи поводяться як команди, що змінюють стан системи: вони приймають параметри операції й повертають підтвердження її виконання або результат. Показовим прикладом є запит:

POST /api/v1/news/{id}/translate – ініціює переклад новини на вказану мову, використовуючи одну з налаштованих AI-моделей. Контролер приймає ідентифікатор новини та параметри перекладу, передає команду в прикладний сервіс через порт і повертає DTO з даними про створений переклад.

Аналогічним чином організовано операції резюмування та авто-тегування, хоча на поточному етапі вони доступні насамперед через адміністративний вебінтерфейс. Це дозволяє обмежити витрати на AI-сервіси та зберегти гнучкість у налаштуванні політик їх використання.

Усі відповіді REST API побудовано за єдиним шаблоном:

– поле *meta* – супровідна інформація (пагінація, службові параметри, додаткові ознаки відповіді);

– поле *data* – основний вміст (один DTO або масив DTO);
– поле *error* (та, за потреби, message) – структурований опис помилки, який використовується у випадках неуспішного виконання запиту.

Єдиний формат полегшує обробку результатів на стороні клієнтів та спрощує підключення нових ресурсів, оскільки клієнтські застосунки можуть повторно використовувати наявну інфраструктуру для роботи з відповідями API.

3.3.3 Безпека та аутентифікація

Доступ до новинного REST API захищено за допомогою схеми API-ключів. Кожному зареєстрованому користувачу може бути видано один або кілька ключів, які зберігаються в окремій базі даних *news-rest* у таблиці *api_keys*, пов'язаній з таблицею *users*. Отримання й базове керування ключами (створення, відкликання) здійснюється через вебінтерфейс REST-модуля; детальний опис відповідних екранів наведено у підрозділі 3.3.4.

Під час звернення до захищених ресурсів клієнт передає ключ у HTTP-заголовку запиту за схемою – *Authorization: Bearer {API_KEY}*

На стороні сервера проміжний шар безпеки послідовно виконує такі перевірки:

– коректність формату заголовка та наявність значення ключа;
– існування ключа в таблиці *api_keys* та зв'язок із активним користувачем;
– статус ключа (активний, відкликаний, заблокований);
– за потреби – додаткові обмеження (строк дії, прив'язка до конкретного клієнтського застосунку тощо).

Лише у випадку успішного проходження всіх перевірок запит передається до контролера REST API. У разі порушення будь-якої умови сервер повертає структуровану помилку автентифікації або авторизації з описом причини відмови.

Для захисту від зловживань реалізовано кілька механізмів:

– обмеження швидкості запитів (rate limiting) для окремого API-ключа або IP-адреси, що зменшує ризик brute-force-атак і надмірного навантаження на систему;
– журналювання звернень до REST API в таблиці *api_request_logs* бази даних *news-rest* (для кожного запиту фіксуються час звернення, endpoint, HTTP-метод, ідентифікатор ключа, IP-адреса клієнта та код відповіді сервера – ці дані використовуються для аудиту, аналізу навантаження й розслідування інцидентів безпеки).

У вебінтерфейсі передбачено розмежування прав доступу на основі ролей. Мінімально система розрізняє:

– звичайного користувача, який керує лише власними API-ключами та може переглядати базову інформацію;
– адміністратора, який має доступ до адміністративних розділів: може переглядати користувачів, відкликати ключі, переглядати журнали запитів і змінювати налаштування, пов’язані з безпекою (наприклад, блокувати окремі облікові записи або ключі, що порушують політику використання).

У контексті мобільного сценарію (клієнт на Flutter із власним бекендом на Laravel) REST API агрегатора виступає окремим, самодостатнім джерелом контенту.

Аутентифікація кінцевих користувачів, зберігання лайків і коментарів реалізуються у сторонній системі **news-user-api** і не змішуються зі схемою безпеки агрегатора. Це дозволяє чітко розділити зони відповідальності й спростити супровід обох підсистем.

У підсумку модуль REST API поєднує механізм централізованої аутентифікації за API-ключами, контроль доступу на основі ролей і системне журналювання, що забезпечує керований та передбачуваний рівень безпеки без ускладнення зовнішнього контракту для клієнтських застосунків.

3.3.4 Вебінтерфейси користувачів, адміністратора та майстер додавання джерел з підтримкою AI

Окрім публічного REST API, модуль містить вебінтерфейс, який забезпечує базову роботу з системою через браузер. Він виконує дві головні ролі:

– надає розробникам зручний вхід до сервісу та генерацію API-ключів;
– дає адміністратору інструменти для експлуатації агрегатора без прямого втручання в код чи базу даних.

Вебінтерфейс реалізовано з використанням шаблонізатора Twig, що є стандартним рішенням для Symfony-застосунків.

Для стилізації сторінок застосовано CSS-фреймворк Bootstrap, який забезпечує адаптивну верстку та сучасний зовнішній вигляд без значних витрат на розробку власних стилів.

Публічний інтерфейс користувача починається з головної сторінки (див. рис. 3.4), де розміщено стислий опис призначення сервісу та посилання на реєстрацію й вхід.

Головна сторінка слугує точкою входу для нових користувачів, надаючи загальне уявлення про можливості системи та умови доступу до API. Форми реєстрації та автентифікації реалізовано засобами безпеки Symfony. Після успішного входу користувач переходить до персонального дашборда (/dashboard), де може керувати своїми API-ключами та переглядати статистику використання.

У користувацькому дашборді (див. рис. 3.5) реалізовано такі можливості:

– перегляд списку вже згенерованих API-ключів;
– створення нового ключа з одноразовим показом його значення;
– відкликання (деактивація) застарілих або скомпрометованих ключів;
– швидкий перехід до інтерактивної документації Swagger UI для тестування REST API з використанням виданого ключа.

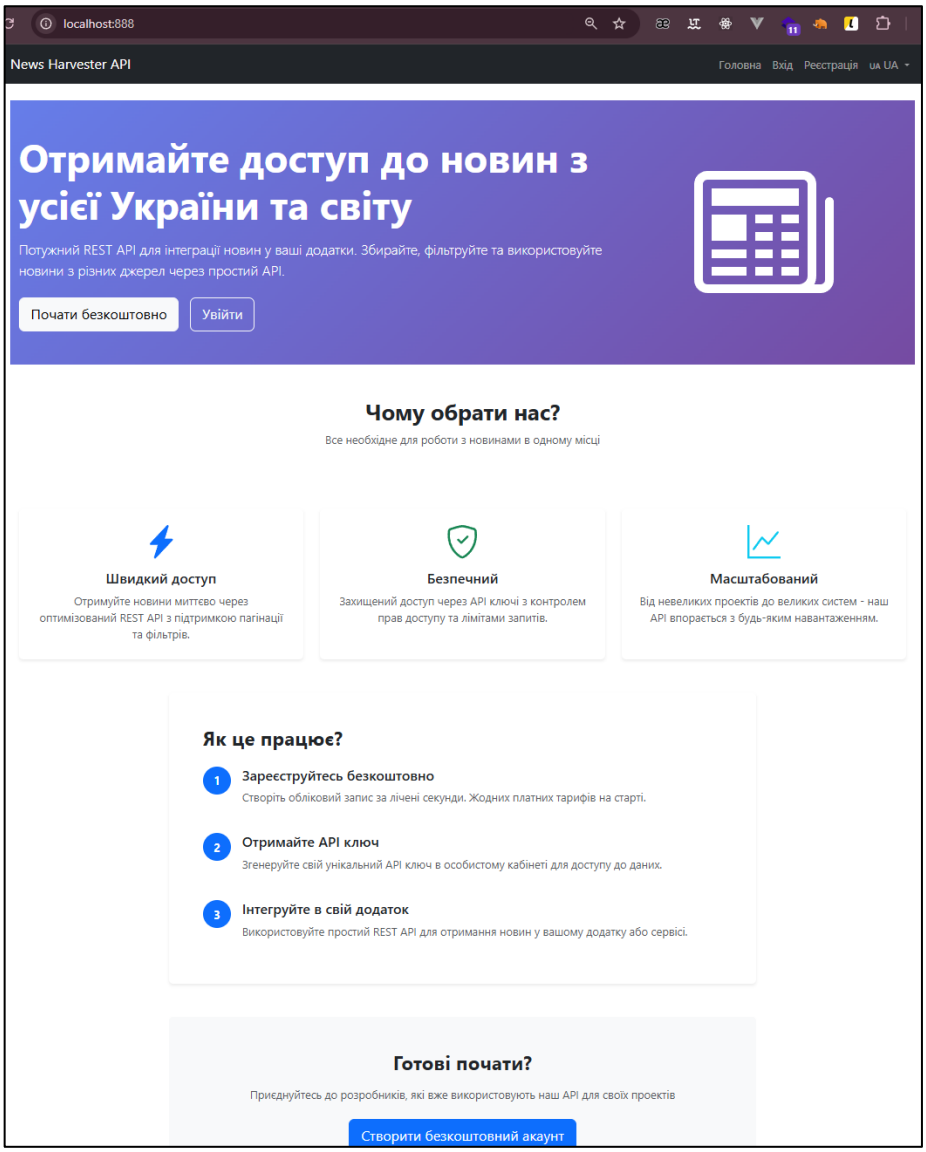


Рисунок 3.4 – Публічна головна сторінка вебінтерфейсу агрегатора новин

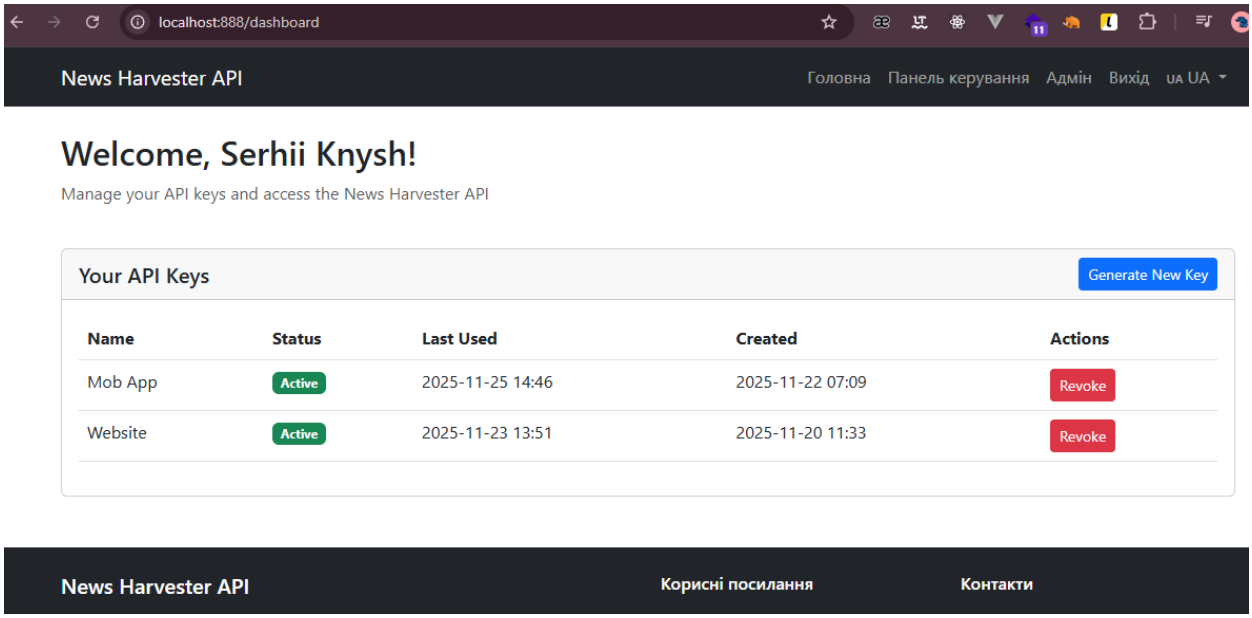


Рисунок 3.5 – Дашборд користувача з переліком API-ключів

Адміністративні функції зосереджено в адмін-панелі (/admin), доступній лише користувачам із відповідною роллю. Адмін-панель структуровано за розділами:

– управління користувачами – перегляд зареєстрованих акаунтів, за потреби блокування доступу;
– управління API-ключами – перегляд і відкликання ключів, аналіз їхнього використання;
– управління джерелами та фідами – створення, редагування, деактивація джерел і прив'язаних до них стрічок;
– перегляд новин – доступ до імпортованих новин з індикацією стану AI-обробки (наявність перекладів, тегів, резюме) та можливістю запускати AI-операції вручну;
– керування AI-моделями – реєстрація моделей, редагування параметрів, активація/деактивація, вибір моделі за замовчуванням для перекладу, резюмування та авто-тегування.

На рисунку 3.6 зображено інтерфейс розділу керування AI-моделями, що демонструє список зареєстрованих моделей та доступні операції над ними.

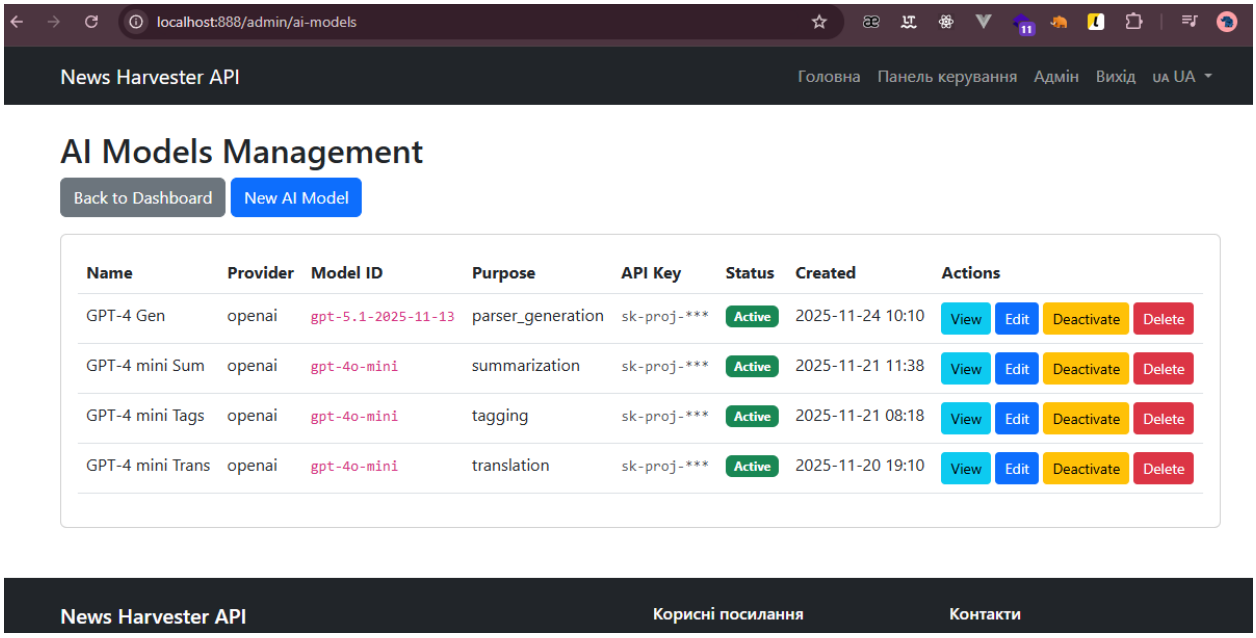


Рисунок 3.6 – Розділ керування AI-моделями в адміністративній панелі REST-модуля

Окремим елементом адміністративного інтерфейсу є майстер (wizard) додавання нових джерел (див. рис. 3.7). Його мета – спростити підключення нового сайту до агрегатора з використанням AI-допомоги для початкового налаштування парсера.

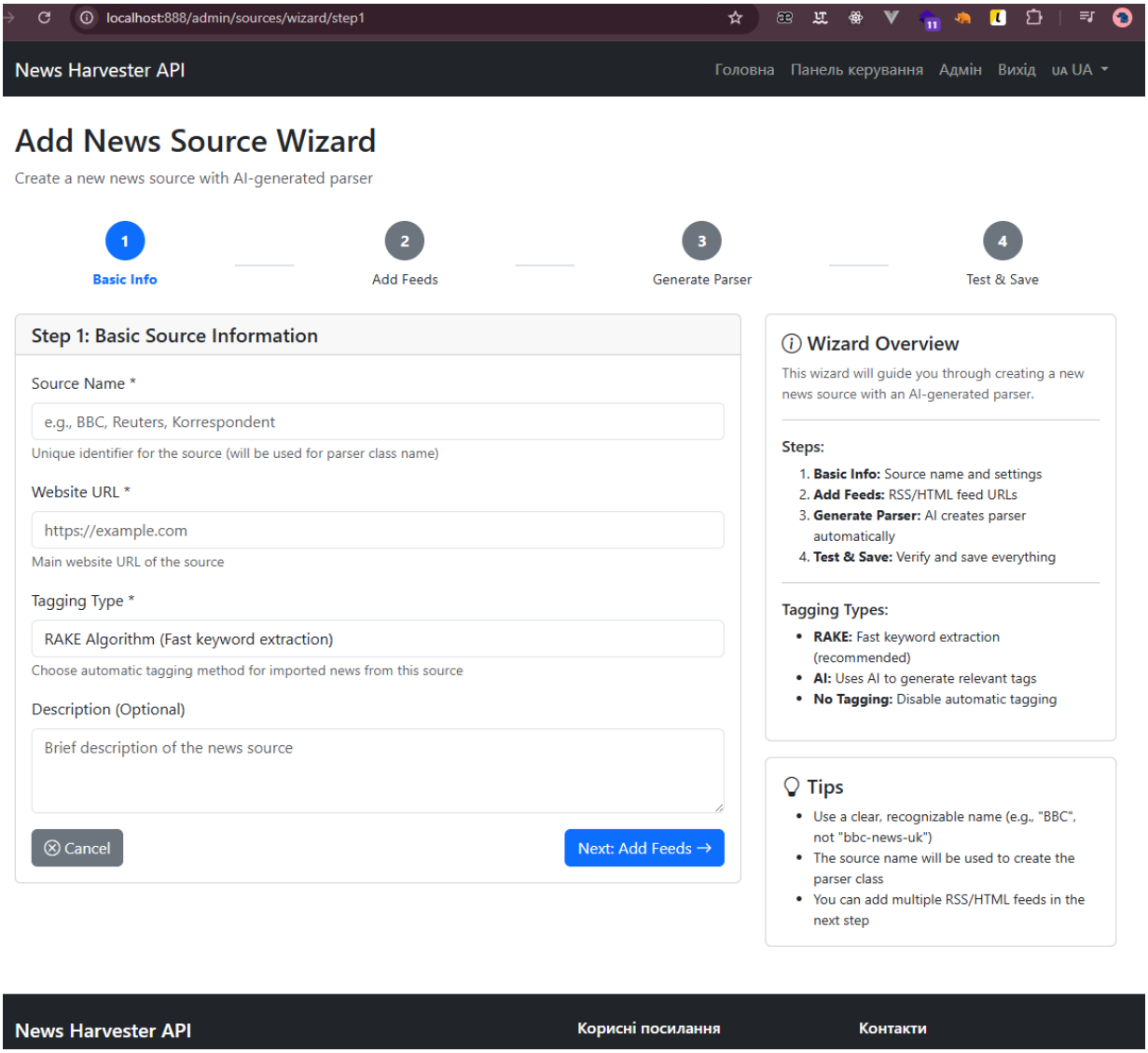


Рисунок 3.7 – Майстер додавання нового ресурсу

Процес складається з кількох кроків:

– введення базової інформації, як назва джерела, базова URL-адреса, опис, тип джерела (RSS чи HTML);
– налаштування посилань і категорій, де виконується додавання конкретних URL-адрес стрічок або сторінок та їх прив’язка до внутрішніх категорій новин;

– AI-аналіз структури документа, тут система завантажує зразок сторінки, AI-модель пропонує початкові селектори для виділення заголовка, тексту, дати публікації та інших полів;
– генерація шаблону парсера і тестування, де на основі зібраних даних створюється шаблон класу парсера для модуля Harvester, після чого адміністратор може виконати тестовий імпорт і за потреби уточнити налаштування.

У сукупності вебінтерфейси користувача, адміністратора та майстер додавання джерел роблять REST-модуль самодостатнім інструментом експлуатації системи. Розробник отримує простий спосіб реєстрації та генерації ключів доступу до API, а адміністратор – засоби керування джерелами, новинами й AI-конфігураціями без необхідності роботи з командним рядком чи безпосередньо з базою даних.

Такий підхід знижує поріг входження для нових учасників проєкту та мінімізує ризик помилок, пов’язаних із ручним редагуванням конфігураційних файлів або SQL-запитів. Усі критичні операції супроводжуються валідацією вхідних даних на рівні форм, а зміни в налаштуваннях набувають чинності одразу після збереження без потреби перезапуску застосунку. Завдяки цьому REST-модуль може використовуватися як у процесі розробки для швидкого тестування нових джерел, так і в продуктивному середовищі для повсякденного адміністрування агрегатора новин.

3.4 Організація бази даних

Організація сховища даних у розробленій системі підпорядкована принципу розподілу відповідальності між окремими підсистемами. На відміну від узагальнених теоретичних моделей, розглянутих у першому розділі, у цьому підрозділі описано практичну структуру реляційних баз даних PostgreSQL, що безпосередньо використовуються модульним агрегатором новин.

У межах Symfony-застосунку виділено дві окремі бази даних PostgreSQL:

– **news-harvester** – основне сховище доменних сутностей агрегатора (новини, джерела, категорії, теги) та артефактів AI-обробки;
– **news-rest** – база даних модуля REST API, яка містить облікові записи користувачів вебінтерфейсу, API-ключі та журнали звернень до API.

Такий поділ дозволяє ізолювати предметні дані (новини та пов’язані з ними метадані) від технічних даних, що стосуються автентифікації, авторизації й експлуатації REST API. Це полегшує супровід системи, дає змогу незалежно розвивати модулі Harvester і REST та у майбутньому спрощує можливий перехід до мікросервісної архітектури.

Окремо від цих двох баз існують зовнішні клієнтські системи (наприклад, бекенд мобільного застосунку на Laravel та вебклієнт на Next.js), які мають власні сховища й взаємодіють з агрегатором виключно через REST API, не отримуючи прямого доступу до баз даних *news-harvester* та *news-rest*.

3.4.1 База даних *news-harvester*: модель новин та AI-метадані

База даних news-harvester є ядром сховища для всіх даних, пов’язаних із новинним контентом і його AI-обробкою. Вона спроєктована таким чином, щоб:

– підтримувати уніфіковане подання новин незалежно від джерела;
– зберігати структуру, зручну для подальшої роботи REST API і клієнтських застосунків;
– забезпечити розширюваність щодо нових типів метаданих і AI-операцій.

Логічну структуру бази даних news-harvester, включно з основними таблицями news, news_content, довідниками (sources, categories, tags) та зв’язками між ними, представлено на рисунку 3.8.

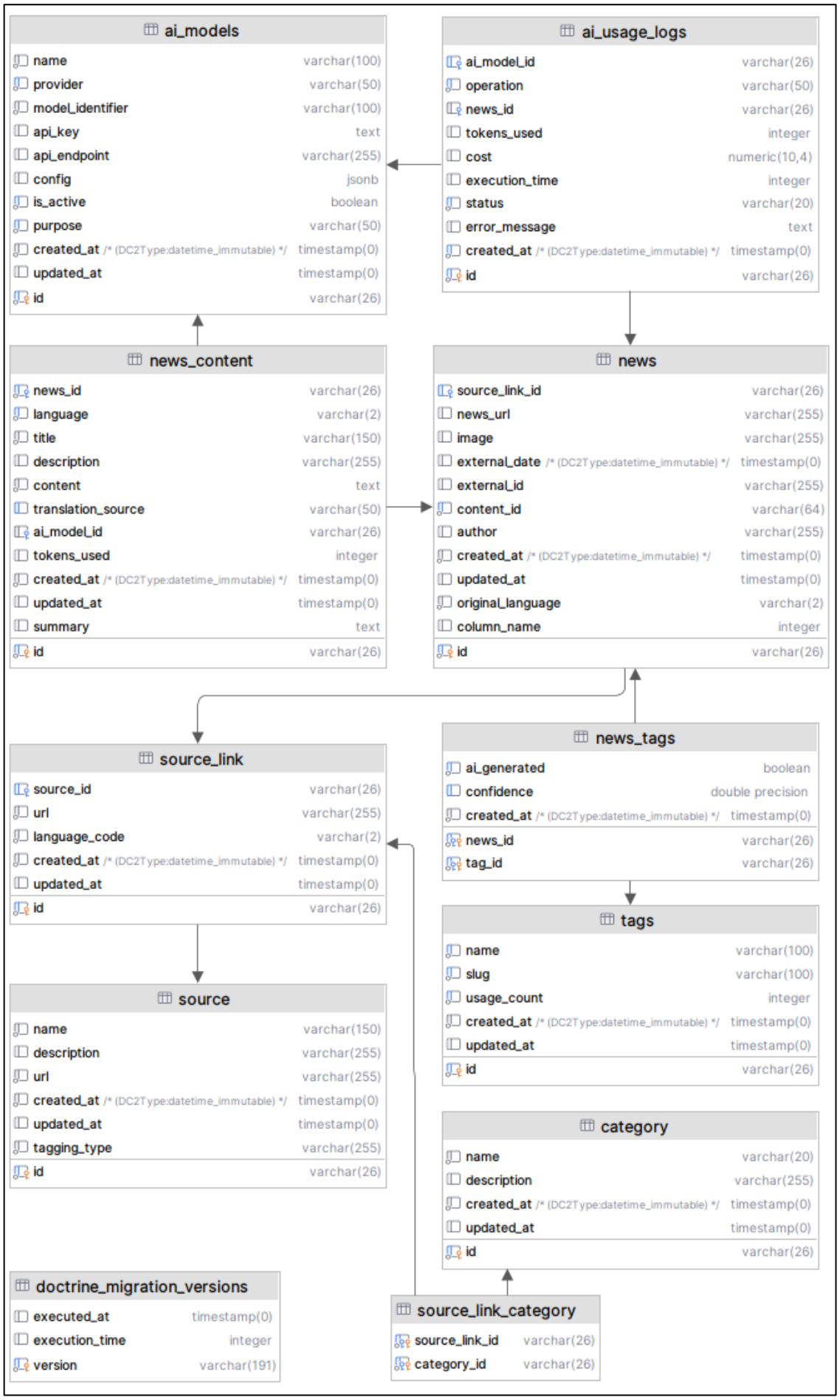


Рисунок 3.8 – Логічна схема бази даних news-harvester

Центральне місце у схемі посідає таблиця news, у якій зберігаються нормалізовані метадані новини, але не повний текстовий вміст. Типовий запис news містить:

– первинний ключ *id* (ULID), що забезпечує глобальну унікальність і зручний розподіл ідентифікаторів у розподілених сценаріях;
– основні «легкі» атрибути новини (наприклад, оригінальний заголовок, технічний ідентифікатор з джерела, мову за замовчуванням);
– поле *url* із посиланням на оригінальну публікацію;
– часові мітки *published_at, created_at, updated_at*;
– зовнішні ключі *source_id* та *category_id,* які пов’язують новину з джерелом і тематичною категорією.

Текстовий вміст новини винесено в окрему таблицю *news_content*. Це дозволяє:

– зберігати **оригінальний текст** новини окремим записом;
– додавати **переклади** на інші мови, згенеровані AI-модулем;
– розміщувати поряд із повним текстом додаткові варіанти вмісту (наприклад, коротке резюме).

Типовий запис у таблиці *news_content* містить:

– зовнішній ключ *news_id*, який пов’язує запис із конкретною новиною з таблиці *news*;
– поле *language*, що визначає мову цього варіанта вмісту (наприклад, uk, en);
– ознаку типу вмісту (оригінальний текст, переклад, резюме тощо);
– текстові поля для заголовка, короткого опису та повного тексту відповідною мовою.

Таким чином, одна й та сама новина може мати кілька записів у *news_content*: один – для оригіналу, інші – для перекладів та/або стиснених версій, згенерованих AI-модулем. Це полегшує підтримку мультимовності та дозволяє відокремити «важкі» текстові поля від основної таблиці *news*, що

позитивно впливає на продуктивність типових запитів (наприклад, вибірки списків новин). Кожен запис у *news_content* містить ідентифікатор мови та тип контенту (оригінал, переклад, резюме), що дає змогу клієнтським застосункам запитувати саме ту версію тексту, яка відповідає мовним уподобанням користувача. При цьому додавання нових мов або типів обробки не потребує змін у структурі основної таблиці новин.

Таблиці *sources* та *categories* виконують роль довідників. У *sources* зберігається інформація про джерела новин (назва, домен, мова за замовчуванням, статус активності), а в *categories* – перелік тематичних категорій (політика, економіка, спорт тощо), до яких належать новини. Використання довідникових таблиць забезпечує нормалізацію даних, спрощує фільтрацію новин за джерелом або категорією та полегшує подальше розширення системи новими джерелами без дублювання інформації.

Для гнучкішої класифікації використовується окремий блок таблиць, пов’язаний із тегами:

– таблиця *tags* – довідник тегів з унікальними назвами або «slug»-ідентифікаторами;
– проміжна таблиця *news_tags*, яка реалізує зв’язок «багато-до-багатьох» між новинами та тегами.

Така організація дозволяє поєднувати фіксовані категорії з більш деталізованими тегами, які можуть генеруватися як вручну, так і за допомогою AI. Теги зберігаються в окремій таблиці tags і пов’язуються з новинами через проміжну таблицю news_tags, що реалізує відношення «багато-до-багатьох». Це забезпечує гнучкість у класифікації контенту: категорії відповідають за загальну тематичну рубрикацію, тоді як теги дозволяють точніше описувати зміст конкретної новини ключовими словами чи іменованими сутностями.

Окремий набір таблиць стосується AI-обробки новин. На концептуальному рівні виділяються:

– таблиці, що зберігають результати AI-операцій (переклади, резюме, набори тегів), де частина цих результатів відображається у

відповідних записах *news_content* (наприклад, перекладений текст або резюме), а додаткові технічні атрибути можуть фіксуватися в окремих таблицях;

– журнали використання AI-моделей, де фіксуються виклики, типи операцій, час виконання та базові технічні метрики;
– таблиця *ai_models* – містить конфігурацію доступних AI-моделей (тип операцій – переклад, резюмування, авто-тегування; параметри генерації; статус активності; провайдер – наприклад, OpenAI).

Така структура забезпечує прозорість роботи AI-підсистеми та дозволяє аналізувати ефективність різних моделей, відстежувати обсяги спожитих ресурсів і виявляти можливі проблеми в обробці контенту.

Усі зв’язки між таблицями реалізовано за допомогою зовнішніх ключів із відповідними обмеженнями цілісності. Для поліпшення продуктивності пошуку новин застосовуються індекси за полями *published_at*, *source_id*, *category_id*, мовою та іншими полями, що активно використовуються у фільтрах REST API.

3.4.2 База даних *news-rest*: користувачі, API-ключі та журнали запитів

Друга база даних, *news-rest*, належить модулю REST API й використовується для зберігання технічної інформації, пов’язаної з доступом до сервісу та його експлуатацією. Вона не містить самих новин, а відповідає за:

– облікові записи користувачів, які працюють із вебінтерфейсом;
– API-ключі, видані цим користувачам;
– журнали звернень до REST API та інші службові дані.

У центрі цієї схеми знаходиться таблиця users, у якій зберігаються:

– унікальний ідентифікатор користувача;
– облікові дані (логін, email, хеш пароля);

– ролі й статус (активний, заблокований);
– часові мітки створення та оновлення запису.

З таблицею *users* пов’язана таблиця *api_keys*, яка реалізує механізм аутентифікації, описаний у підрозділі 3.3.3. Кожен рядок у *api_keys* містить:

– унікальний токен API (рядок довільної довжини);
– посилання на користувача (user_id);
– статус ключа (активний, відкликаний, заблокований);
– дату створення, останнього використання та, за потреби, строк дії;
– додаткові атрибути (наприклад, назву або опис ключа, щоб відрізняти його між різними клієнтськими застосунками одного користувача).

Для забезпечення прозорості та можливості аудиту запитів використовується таблиця *api_request_logs*. Вона фіксує:

– час звернення;
– endpoint і HTTP-метод;
– ідентифікатор використаного API-ключа;
– IP-адресу клієнта;
– код відповіді сервера та короткий статус (успішно, помилка автентифікації, помилка авторизації тощо).

Логічну структуру бази даних *news-rest* доцільно окремо подати у вигляді ER-діаграми на рисунку 3.9.

За потреби схема доповнюється службовими таблицями безпеки (наприклад, для зберігання сесій або підтвердження email), однак вони не впливають на основний контракт REST API й можуть розглядатися як технічна інфраструктура.

Клієнтські застосунки (мобільний сервіс на Laravel та вебклієнт на Next.js) використовують ці бази даних опосередковано, звертаючись до агрегатора виключно через REST API без прямого доступу до *news-harvester* та *news-rest.*

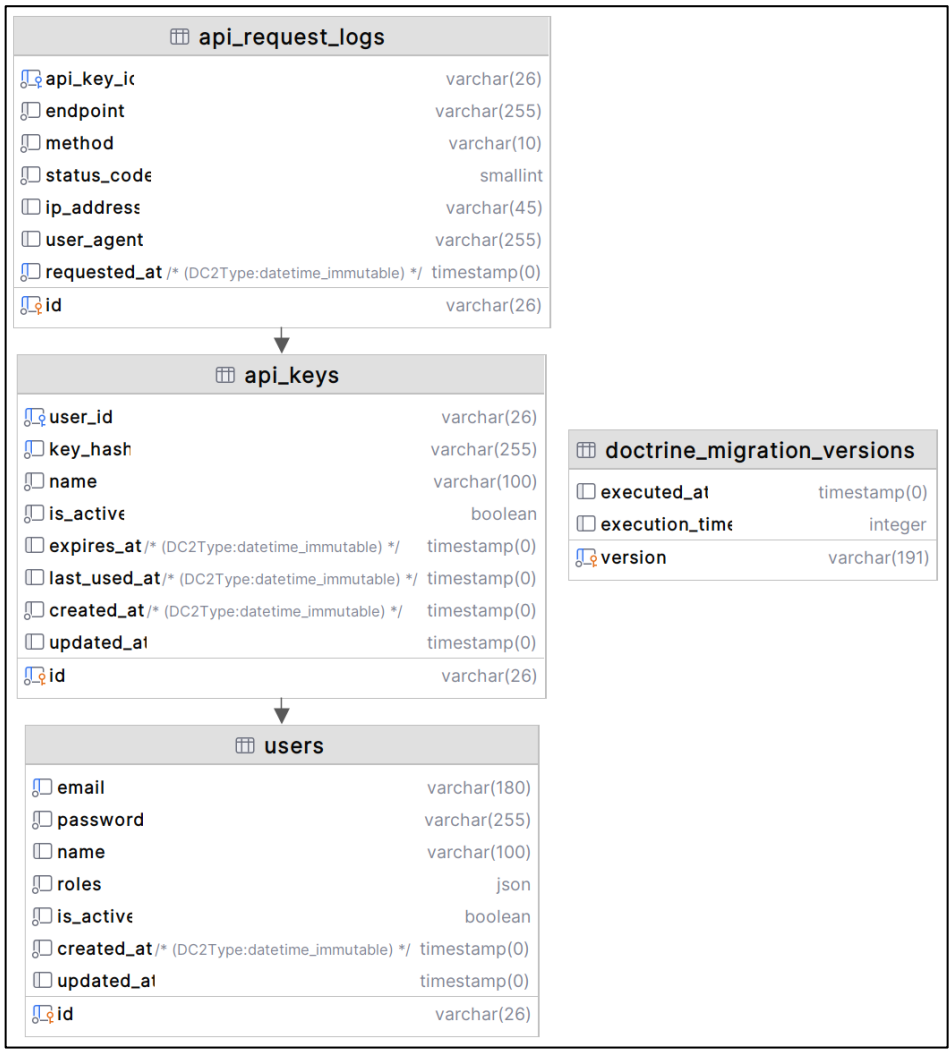


Рисунок 3.9 – Логічна схема бази даних *news-rest*

3.5 Використання Docker для локального середовища розробки

Для забезпечення відтворюваного середовища розробки та спрощення розгортання агрегатора новин використано контейнеризацію на базі Docker [6] та Docker Compose. Усі основні компоненти системи – вебзастосунок Symfony, бази даних PostgreSQL та допоміжні сервіси – запускаються в окремих контейнерах, конфігурація яких описана у спільному файлі docker-compose.yml. Такий підхід гарантує ідентичність середовища на різних машинах розробників і усуває проблему «працює на моєму комп’ютері», оскільки всі залежності та версії програмного забезпечення фіксуються в образах контейнерів.

Типова конфігурація локального середовища (див. рис. 3.10) включає такі сервіси:

– контейнер додатка – PHP-FPM і Nginx, у якому розгортається модульний моноліт Symfony з усіма модулями (Harvester, REST API, вебінтерфейс адміністратора);
– контейнер postgres-harvester – екземпляр PostgreSQL із базою даних news-harvester, що зберігає доменні сутності новин, джерел, категорій, тегів і дані про AI-обробку;
– контейнер postgres-rest – екземпляр PostgreSQL із базою news-rest, у якій розміщено облікові записи користувачів REST-модуля, API-ключі й журнали запитів;
– додаткові сервіси (за потреби) – наприклад, Redis для кешування або черг фонового виконання задач.

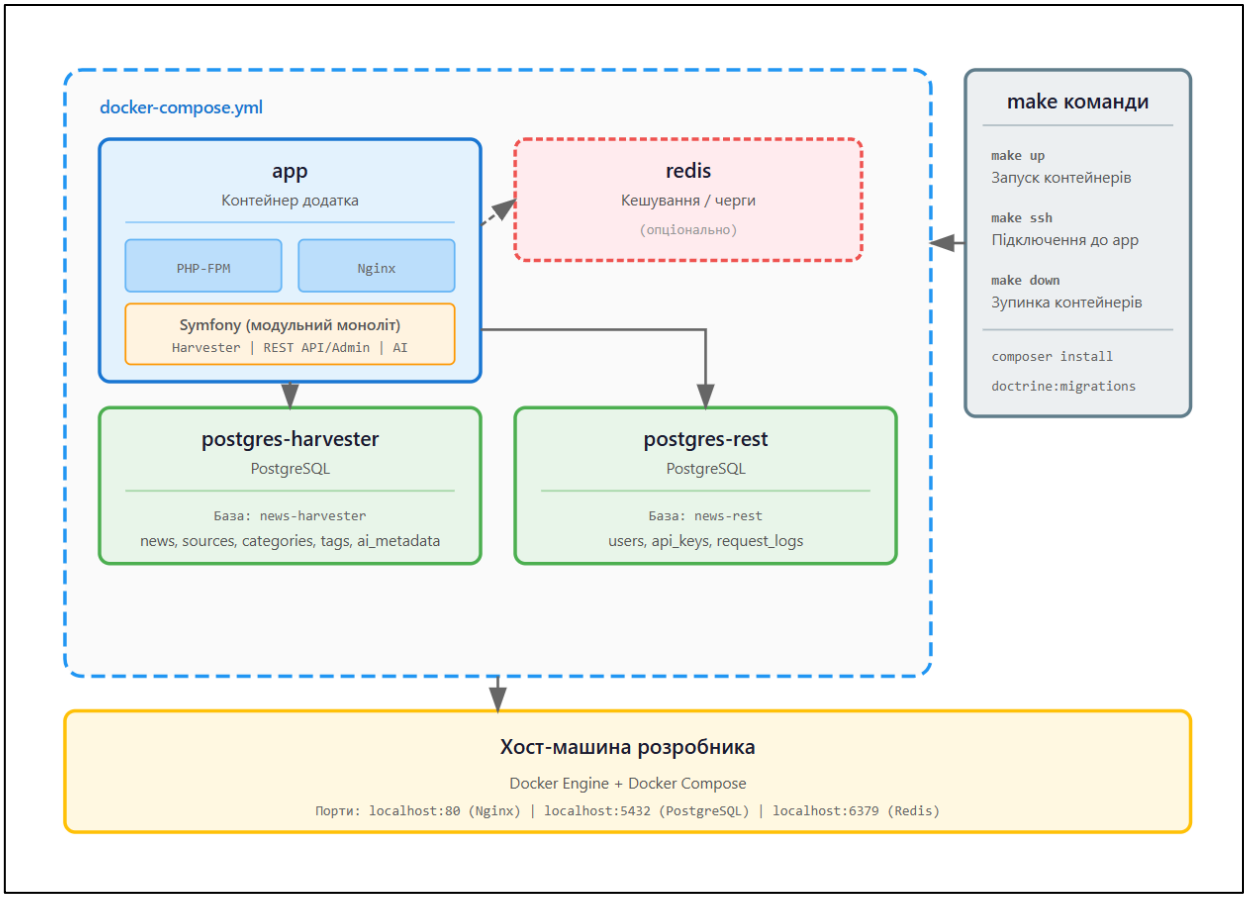


Рисунок 3.10 – Схема контейнерів Docker для локального середовища розробки агрегатора новин

Кожен контейнер ізольовано працює у власному середовищі, а взаємодія між ними здійснюється через внутрішню мережу Docker. Це дозволяє контейнеру додатка звертатися до баз даних за їхніми іменами сервісів (postgres-harvester, postgres-rest), абстрагуючись від конкретних IP-адрес. Конфігураційні параметри, такі як облікові дані баз даних, ключі API зовнішніх сервісів та інші секрети, передаються через змінні оточення, що спрощує адаптацію середовища до різних умов розгортання.

Для зручності роботи описано набір цільових команд make, які автоматизують типові операції розробника:

– make up – запуск усіх контейнерів у фоновому режимі;
– make ssh – підключення до контейнера застосунку для виконання команд PHP та Symfony;
– make down – зупинка й видалення запущених контейнерів;
– первинна ініціалізація проєкту всередині контейнера (встановлення залежностей і застосування міграцій БД):
– composer install;
– php bin/console doctrine:migrations:migrate.

Контейнеризація дає кілька важливих переваг:

– ідентичне середовище для всіх учасників проєкту незалежно від операційної системи розробника;
– швидке розгортання системи на новій машині – достатньо встановити Docker, клонувати репозиторій і виконати make up;
– спрощене перенесення в продакшн – конфігурація, відпрацьована локально, може бути використана як основа для розгортання на VPS або в хмарній інфраструктурі.

У продакшн-сценарії конфігурація Docker-довкілля доповнюється окремими контейнерами для зовнішніх сервісів, які взаємодіють з агрегатором: User API на базі Laravel та вебсайту-ридера на Next.js. При цьому новинний агрегатор залишається ядром системи, яке надає єдиний REST API для всіх клієнтів.

3.6 Клієнтські застосунки – споживачі REST API

Розроблений модульний агрегатор новин виступає ядром екосистеми, яке надає єдиний REST API для різних клієнтських застосунків. На момент підготовки роботи реалізовано два основні клієнти, що демонструють практичне використання API:

– вебклієнт на Next.js, який відображає новини в браузері;
– мобільний застосунок на Flutter з окремим бекендом User API на Laravel для облікових записів, коментарів і лайків.

Обидва клієнти взаємодіють з агрегатором виключно через REST API модуля Harvester, використовуючи згенеровані API-ключі. При цьому бекенд User API є самостійною зовнішньою системою, яка не має прямого доступу до баз даних агрегатора й комбінує отримані з нього новинні дані з власними соціальними даними.

3.6.1 Вебклієнт на Next.js

Вебсайт news-web-client побудований на базі фреймворку Next.js і виконує роль «ридера» новин. Він не має власного окремого бекенда та працює як клієнт до REST API агрегатора, використовуючи API-ключ, згенерований у вебінтерфейсі REST-модуля. Next.js забезпечує серверний рендеринг (SSR) та статичну генерацію сторінок (SSG), що позитивно впливає на швидкість завантаження та індексацію контенту пошуковими системами. Для стилізації інтерфейсу застосовано бібліотеку Tailwind CSS, яка дозволяє швидко створювати адаптивні макети без написання окремих CSS-файлів.

Основні екрани вебсайту:

– головна сторінка зі списком новин і полем для пошуку/фільтрації, де під час завантаження сторінки клієнт звертається до кінцевих точок (endpoints) на кшталт GET /api/v1/news, передаючи параметри пагінації та фільтри (наприклад, джерело або категорію);

– сторінка джерел новин, де відображається перелік доступних джерел, а під час вибору конкретного джерела вебклієнт формує запити до API з відповідним фільтром, фактично використовуючи джерело як критерій відбору новин;
– детальна сторінка новини, яка отримує новинний запис за ідентифікатором (GET /api/v1/news/{id}) та дозволяє перемикати мову відображення (за наявності вже згенерованих перекладів контент завантажується з відповідних записів у news_content; за потреби користувач може ініціювати переклад через REST API, який, у свою чергу, використовує AI-модуль).

На рисунках 3.11–3.13 наведено приклади основних екранів вебклієнта, що ілюструють типові сценарії взаємодії користувача із системою.

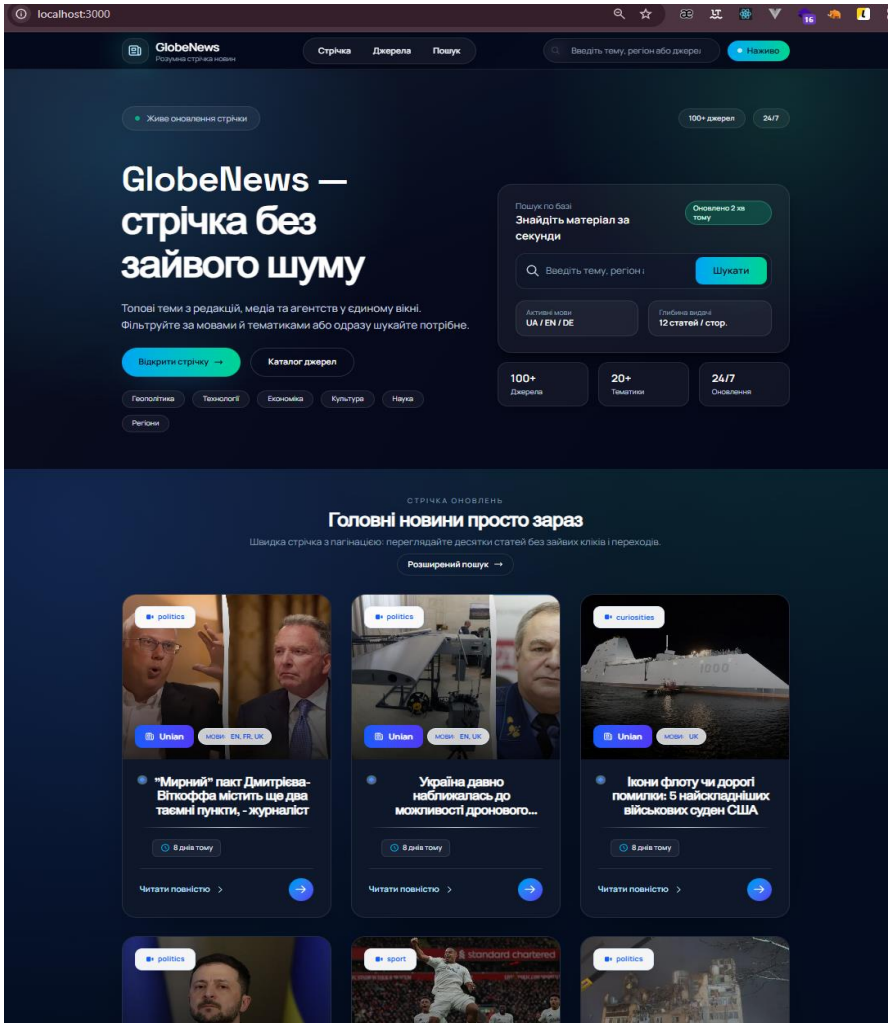


Рисунок 3.11 – Головна сторінка вебклієнта Next.js зі списком новин і полем фільтрації та hero

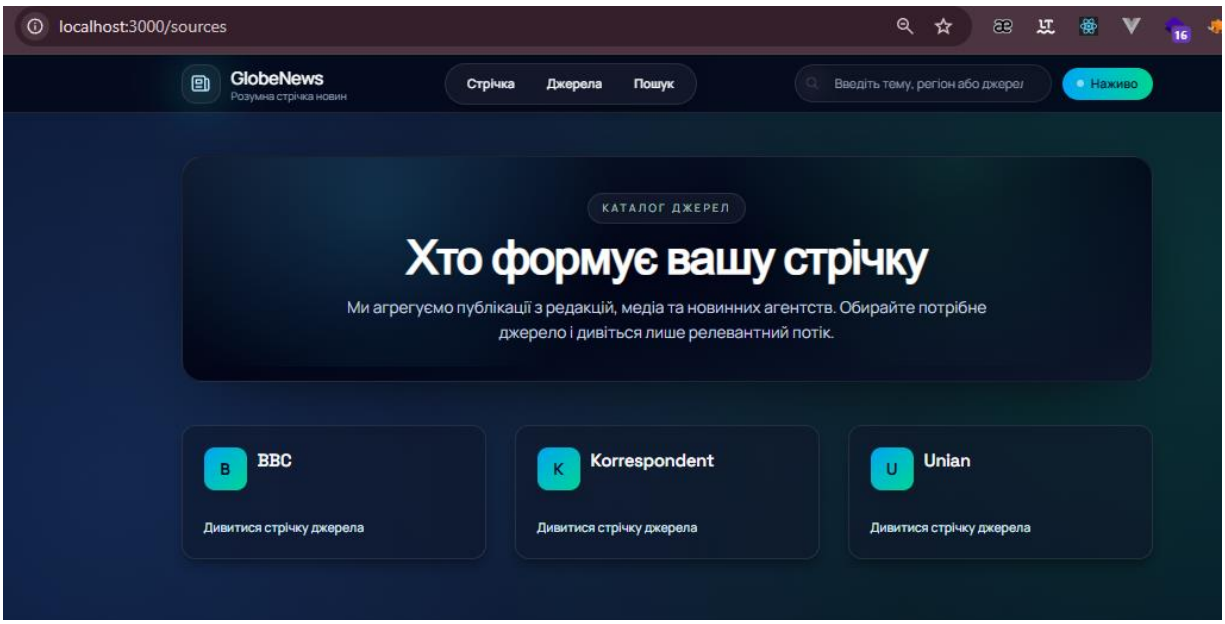


Рисунок 3.12 – Сторінка джерел новин із вибором джерела як фільтра

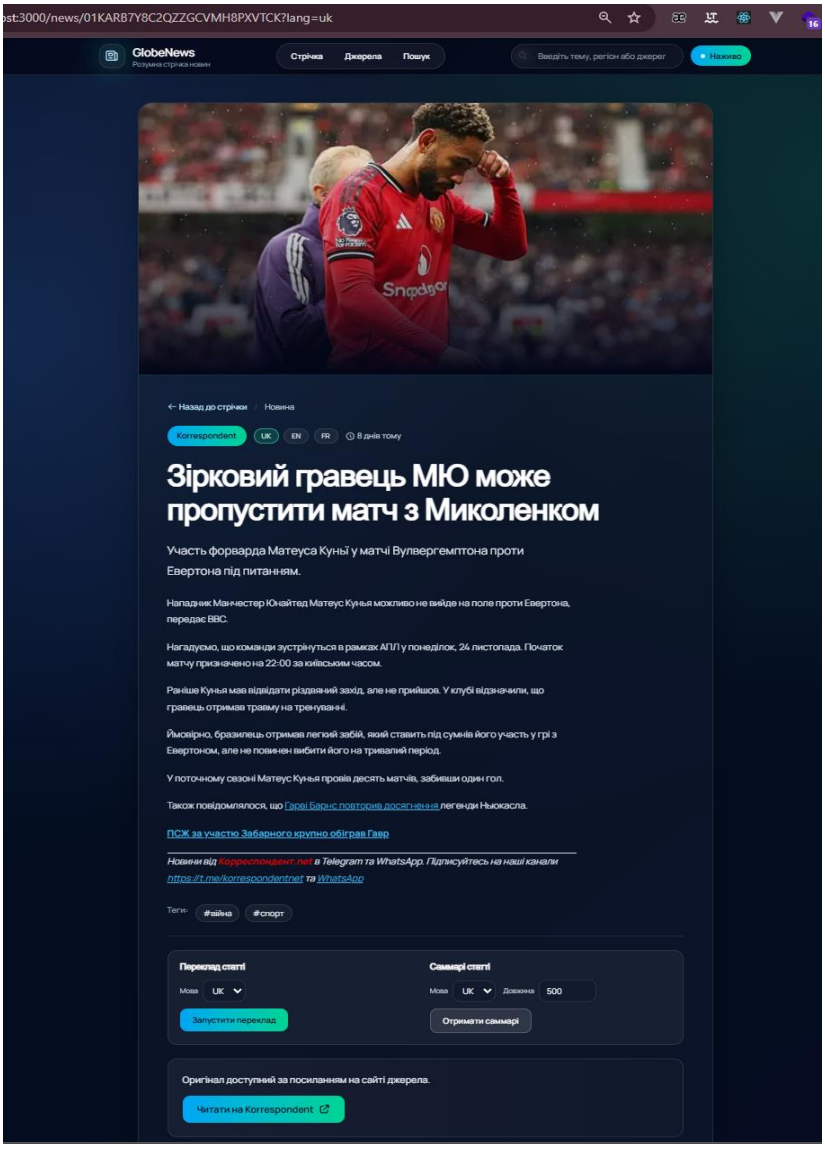


Рисунок 3.13 – Детальна сторінка новини

На рівні клієнта реалізовано просту логіку роботи з REST API: запити виконуються з передачею API-ключа в заголовку авторизації, а отримані дані відображаються у вигляді списків, карток новин та детальних сторінок. Додаткове кешування й оптимізація запитів можуть реалізовуватися засобами Next.js та бібліотек для роботи з даними, але ці аспекти не є ключовими для даної кваліфікаційної роботи.

3.6.2 Мобільний додаток на Flutter

Мобільний застосунок news-mobile-app реалізований на Flutter і наразі орієнтований на платформу Android (з перспективою подальшого розширення на iOS). Його завдання – надати користувачеві зручний мобільний доступ до новин, отриманих із агрегатора, а також базові соціальні функції, що реалізовані в окремому бекенді.

З погляду архітектури мобільний клієнт працює із двома HTTP-сервісами:

– Harvester REST API (Symfony) – джерело новинного контенту; через нього завантажуються списки новин, детальна інформація, переклади та резюме;
– User API (Laravel) – окремий бекенд, відповідальний за реєстрацію й автентифікацію користувачів, а також зберігання коментарів, лайків та інших соціальних даних (для нього також підготовлена документація у форматі Swagger).

Основні сценарії взаємодії користувача з мобільним застосунком охоплюють перегляд новин, автентифікацію та соціальну взаємодію. Після запуску застосунок виконує запит до Harvester API для завантаження останніх новин і відображає їх у вигляді стрічки з можливістю пролистування та переходу до детального перегляду. Для зручності пошуку передбачено відповідне поле, яке формує запити з фільтрами до REST API.

Користувач може створити обліковий запис або увійти до існуючого через User API. Отримані токени автентифікації використовуються виключно у взаємодії з Laravel-бекендом і не передаються агрегатору новин, що забезпечує розмежування відповідальності між сервісами.

Інтерфейс мобільного клієнта представлено на рисунках 3.14 та 3.15. На рисунку 3.14 зображено головну сторінку зі списком новин та полем пошуку у світлій і темній темах оформлення, а на рисунку 3.15 – детальну сторінку новини з функціями коментування, лайків та перекладу контенту.

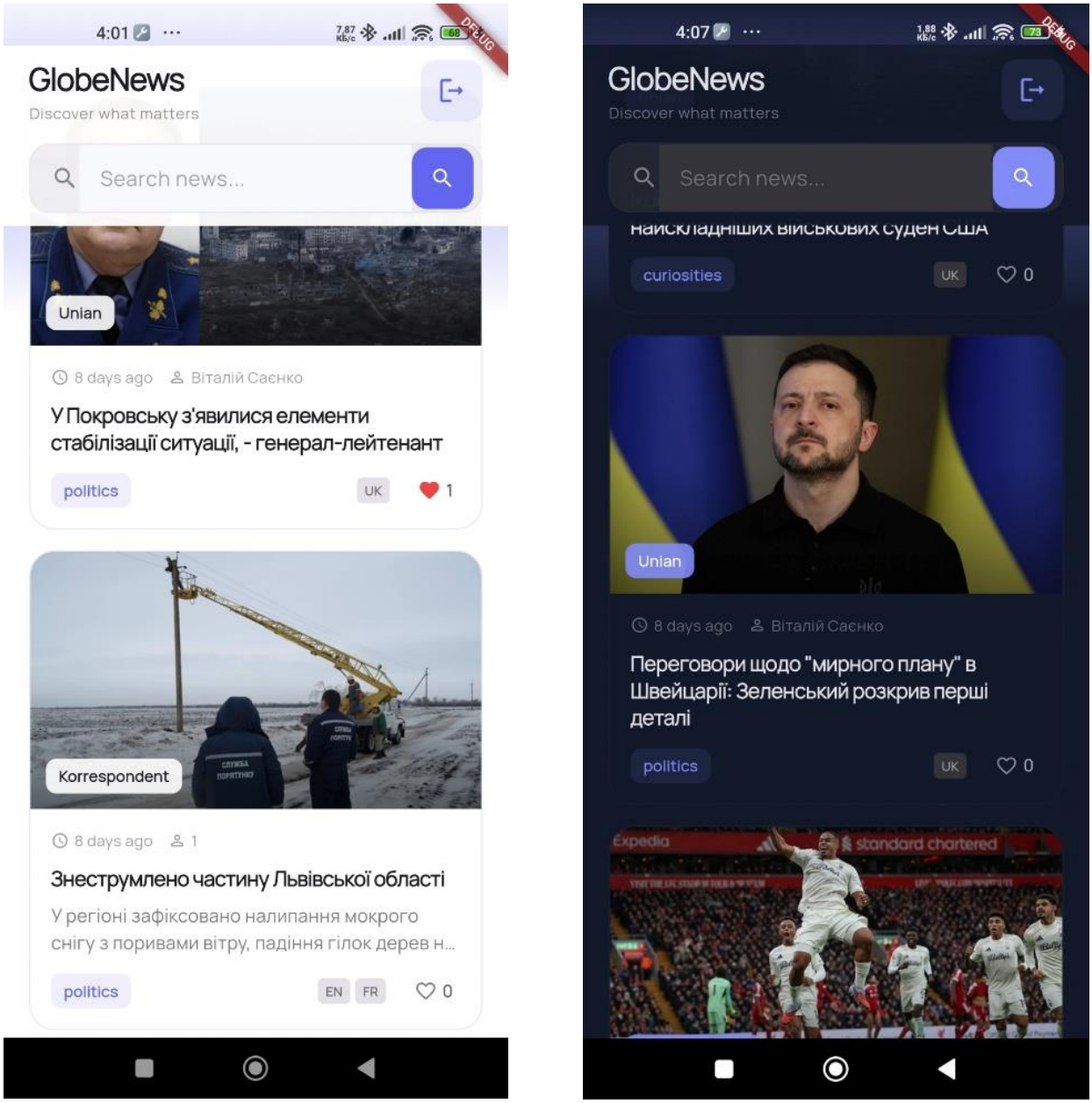


Рисунок 3.14 – Головна сторінка мобільного застосунку Flutter зі списком новин і пошуком (світла та темна тема)

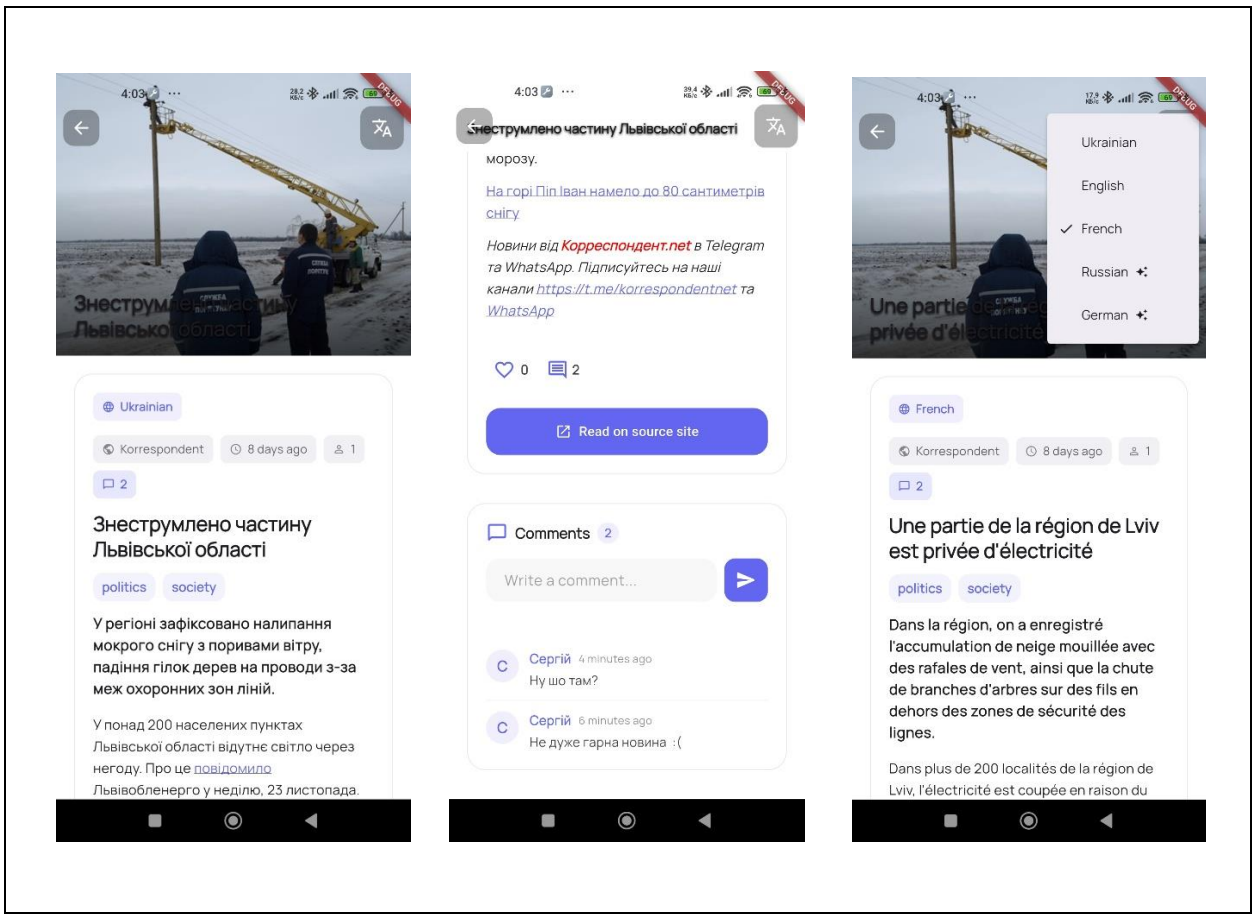


Рисунок 3.15 – Детальна сторінка новини з коментарями, лайками та перекладом в мобільному застосунку

На детальній сторінці новини мобільний клієнт завантажує контент із Harvester API, тоді як коментарі, лайки та іншу соціальну інформацію отримує з User API. Додавання нового коментаря або лайка відправляється на бекенд мобільного застосунку, де зберігається в його власній базі даних, не змінюючи структуру сховища агрегатора.

Такий поділ дозволяє зберегти агрегатор новин максимально незалежним від конкретних сценаріїв соціальної взаємодії в мобільному клієнті. З погляду архітектури агрегатор і User API пов’язані лише через HTTP-контракт: бекенд мобільного застосунку зберігає посилання на новини за їхніми ідентифікаторами з Harvester API й об’єднує дані двох джерел у відповідях для застосунку.

3.7 Тестування функціональності та продуктивності

У другому розділі було розглянуто інструменти забезпечення якості коду (PHPUnit, Behat, статичні аналізатори, GitLab CI/CD) та їхню роль у процесі розроблення. У цьому підрозділі зосереджено увагу на тому, як ці засоби застосовано безпосередньо в розробленому агрегаторі новин: які саме частини системи покрито тестами, як організовано запуск тестів та яким чином результати інтегруються в CI/CD-процес.

3.7.1 Автоматизоване тестування (PHPUnit)

Для перевірки коректності роботи основних компонентів системи було створено низку модульних та інтеграційних тестів на базі PHPUnit [14]. Вони охоплюють насамперед ті частини застосунку, які безпосередньо впливають на правильність обробки новин, взаємодію з базою даних та стабільність REST API.

Серед ключових груп тестів можна виділити:

– **тести для CQRS-хендлерів імпорту новин** – перевіряють сценарії, коли з джерела надходить змішаний набір новин (частина з них уже присутня в базі, а частина є новою), моделюють відповіді парсерів і перевіряють, що дублікати не додаються повторно, а нові записи коректно зберігаються в *news* та *news_content* із відповідними зв’язками;

– **тести для парсерів окремих джерел** – для таких джерел, як, наприклад, новинні сайти *unian* чи *korrespondent*, використовуються збережені HTML/RSS-фрагменти як фікстури (це дозволяє перевіряти коректність CSS/XPath-селекторів: виділення заголовка, дати публікації, основного тексту, зображень тощо навіть у разі змін на стороні джерела);

- **тести REST-контролерів** – перевіряють формати JSON-відповідей, структуру полів *data* та *meta*, поведінку пагінації, повернення коректних HTTP-кодів статусу (200, 400, 404, 422 тощо) та обробку типових помилкових ситуацій (некоректні параметри, відсутня новина, невалідний API-ключ);
- **тести сервісів AI-модуля** – у цих тестах реальний клієнт зовнішнього провайдера (OpenAI) замінюється на тестовий дублер (mock/stub) через механізм впровадження залежностей, що дозволяє емулювати різні варіанти відповідей сервера та дає змогу верифікувати коректність побудови промптів, точність парсингу JSON-відповідей у внутрішні DTO й збереження результатів у базу даних.

Завдяки такому розподілу тестів досягається баланс між модульним і інтеграційним рівнем: бізнес-логіка CQRS-хендлерів і сервісів перевіряється ізольовано, тоді як інтеграційні тести REST-контролерів і роботи з базою даних дають упевненість у коректності «наскрізних» сценаріїв.

Запуск тестів інтегровано в процес GitLab CI/CD. Після кожного пушу до основних гілок репозиторію виконується окремий етап test, у якому піднімається тимчасове середовище (контейнери застосунку та баз даних) і запускається повний набір PHPUnit-тестів разом із перевірками статичних аналізаторів. У разі невдалого проходження тестів або порушення правил, визначених PHP_CodeSniffer, PHPMD, PHPStan чи Deptrac, зміни не можуть бути злито до основної гілки. Це дисциплінує процес розробки, забезпечує стабільність головної гілки й підтримує єдині стандарти якості коду в усьому проєкті.

3.7.2 Поведінкове тестування (Behat)

Поведінкове тестування (Behavior-Driven Development, BDD) з використанням Behat [5] застосовується для перевірки наскрізних сценаріїв

роботи агрегатора мовою бізнес-орієнтованих описів. На відміну від модульних тестів PHPUnit, що зосереджені на окремих компонентах, Behat-сценарії описують поведінку системи з точки зору зовнішнього споживача REST API або користувача вебінтерфейсу.

На поточному етапі створено базовий набір сценаріїв, які охоплюють найважливіші кейси:

– отримання списку новин через REST API з валідним API-ключем;
– застосування фільтрації за джерелом і перевірка, що у відповіді повертаються лише новини з обраного джерела;
– обробка типових помилок автентифікації (наприклад, відсутній або невалідний API-ключ);
– базові сценарії роботи з користувачем і API-ключами у вебінтерфейсі.

Типові сценарії формалізуються у форматі Gherkin, що робить їх зрозумілими не лише розробникам, а й іншим учасникам проєкту. Приклад спрощеного сценарію отримання списку новин може виглядати так, як представлено на рисунку 3.16.

```
Feature: News API
  In order to access news articles
  As an API client
  I need to be able to retrieve news through the API

  Background:
    Given there is a test user with API key
    And there are 10 news articles in the database

  Scenario: Get list of news articles with authentication
    Given I have a valid API key
    When I add "Authorization" header equal to "Bearer test-api-key-behat-12345"
    And I send a "GET" request to "/api/v1/news"
    Then the response status code should be 200
    And the response should be in JSON
    And the response should be paginated
    And the JSON node "data" should have 10 elements

  Scenario: Get list of news articles without authentication
    Given I do not have an API key
    When I send a "GET" request to "/api/v1/news"
    Then the response status code should be 401
    And the response should be in JSON
    And the JSON node "error" should exist
```

Рисунок 3.16 – Приклад спрощеного BDD-сценарію у форматі Gherkin

Запуск Behat-сценаріїв інтегровано в процес розробки поряд із виконанням PHPUnit-тестів. Наразі їх кількість є порівняно невеликою й охоплює насамперед базову функціональність, однак навіть цей мінімальний набір дозволяє автоматично перевіряти коректність ключових користувацьких сценаріїв після змін у коді.

У подальшому планується розширити набір поведінкових тестів за рахунок сценаріїв, пов’язаних з AI-обробкою (запуск перекладу, резюмування, авто-тегування) та інтеграцією з клієнтськими застосунками. Це дасть змогу використовувати Behat як більш повноцінний інструмент перевірки вимог і живу документацію для всієї системи.

3.8 Висновки з розділу 3

У розділі 3 було описано архітектуру та реалізацію модульного агрегатора новин на базі Symfony. Система побудована як модульний моноліт із використанням принципів DDD, гексагональної архітектури та CQRS, що дозволило чітко розділити доменну логіку та інфраструктурні компоненти.

Показано, що модуль Harvester забезпечує повний цикл роботи з новинними ресурсами: завантаження HTML/RSS-документів, парсинг, нормалізацію, дедуплікацію та збереження в базі даних, а також подальшу AI-обробку (переклад, резюмування, авто-тегування). Модуль REST API надає публічний HTTP-інтерфейс до цих даних, документований за допомогою OpenAPI/Swagger, і доповнений вебінтерфейсом користувача й адміністратора, включно з майстром додавання нових джерел із використанням AI.

Організація даних базується на двох окремих базах PostgreSQL: news-harvester для доменних сутностей та результатів AI-обробки й news-rest для користувачів, API-ключів і журналів запитів. Для розгортання й локальної розробки використано Docker і Docker Compose, що забезпечує відтворюваність середовища.

Реалізовані клієнти – вебклієнт на Next.js та мобільний застосунок на Flutter з окремим бекендом на Laravel – демонструють практичне використання REST API агрегатора у різних сценаріях. Якість реалізації підтримується завдяки використанню автоматизованих тестів на PHPUnit та інтеграції перевірок у конвеєр GitLab CI/CD.

У сукупності ці рішення формують цілісну, розширювану систему, готову до подальшого масштабування та інтеграції з новими типами джерел і клієнтських застосунків.

4 АНАЛІЗ РЕЗУЛЬТАТІВ ТА ПЕРСПЕКТИВИ

4.1 Оцінка ефективності системи

Ефективність розробленого агрегатора доцільно розглядати комплексно, спираючись на кілька ключових критеріїв. Першим із них є коректність збору та обробки даних. Практичні запуски модуля Harvester на наборі обраних джерел засвідчили, що система успішно завантажує RSS- і HTML-стрічки, виконує парсинг основних полів (заголовка, дати публікації, тексту, зображень) та проводить дедуплікацію записів. При цьому нормалізовані новини зберігаються уніфіковано в базі даних news-harvester, що дозволяє надалі однаково обробляти матеріали з різних джерел.

Важливим аспектом є зручність доступу до накопиченої інформації. Реалізований REST API надає можливість виконувати пошук та отримувати новини з фільтрацією за джерелом, категорією, мовою та іншими параметрами. Завдяки наявності документації у форматі OpenAPI/Swagger та вебінтерфейсу для генерації API-ключів, доступ до даних стає зручним як для власних клієнтських застосунків, так і для сторонніх розробників.

Окремої уваги заслуговують архітектурні рішення та якість реалізації. Використання гексагональної архітектури та CQRS дозволило відокремити доменну логіку від інфраструктурних деталей, таких як джерела даних чи зовнішні AI-сервіси. Це забезпечує модульність і розширюваність системи: нові джерела або клієнти можуть додаватися без кардинальних змін коду, а перехід до мікросервісів можливий поетапно. Надійність реалізації та спрощення супроводу досягаються завдяки модульним тестам на PHPUnit, використанню статичних аналізаторів (PHPStan, Deptrac) та автоматизованим перевіркам у GitLab CI/CD.

У сукупності проведені експерименти із запуском імпорту та тестуванням REST API показали, що система здатна стабільно обробляти дані

з кількох джерел і забезпечує прийнятний час відповіді для веб- і мобільних клієнтів у типовому сценарії використання.

4.2 Аналіз масштабованості та продуктивності

Завдяки модульній архітектурі та використанню Docker система має потенціал для подальшого масштабування й оптимізації продуктивності.

З погляду масштабованості можна виділити такі напрями:

– горизонтальне масштабування API – контейнер із Symfony-застосунком (модулі Harvester і REST API) може бути запущений у кількох екземплярах за reverse proxy (наприклад, Nginx), що дозволяє розподіляти навантаження між інстансами;
– винесення окремих модулів в окремі сервіси – зокрема, AI-модуль може бути перетворений на окремий сервіс із HTTP-інтерфейсом, а модуль імпорту – на самостійну службу, що працює за розкладом і через черги;
– оптимізація імпорту – використання асинхронних HTTP-запитів, пакетної обробки новин і черг задач дозволяє краще контролювати навантаження на зовнішні джерела та зменшувати час повного циклу імпорту.

Для підвищення продуктивності в майбутньому доцільно:

– додати кешування «гарячих» даних (наприклад, списків останніх новин) за допомогою Redis або аналогічних рішень;
– рознести сховища читання й запису, використовуючи репліки бази даних для операцій читання через REST API;
– винести ресурсоємні операції (імпорт, AI-переклад, генерування резюме) у фонові задачі із використанням системи черг.

У межах магістерської роботи основний акцент був зроблений на коректності реалізації логіки збору й наданні доступу до даних. Детальні навантажувальні тести на великих обсягах даних та повномасштабні сценарії

горизонтального масштабування розглядаються як подальший етап розвитку проєкту.

4.3 Обмеження та труднощі дослідження

У процесі розробки системи виникла низка об’єктивних обмежень і технічних труднощів, які вплинули на фінальний обсяг функціоналу та глибину експериментів.

Насамперед, обмежені часові рамки та ресурси не дозволили в повному обсязі реалізувати всі заплановані компоненти. Зокрема, розробка мобільного клієнта обмежилася базовими сценаріями взаємодії з API, а такі аспекти, як складні механізми офлайн-роботи, побудова повноцінного CI/CD для production-середовища та кластеризація бази даних, залишилися поза межами поточної ітерації.

Суттєвим викликом стала залежність підсистеми збору даних від зовнішніх ресурсів. Стабільність роботи парсерів прямо залежить від незмінності DOM-структури сайтів-донорів. Будь-які зміни у верстці або посилення захисту від автоматизованого збору даних вимагають оперативного втручання та оновлення конекторів, а в окремих випадках — тимчасової відмови від інтеграції.

Інтеграція з AI-сервісами також супроводжувалася обмеженнями: масштабне використання LLM лімітується фінансовими витратами та швидкістю відповіді API, що вимагало впровадження механізмів контролю квот. Тестування системи проводилося переважно в локальному середовищі; комплексні навантажувальні випробування (stress testing) для перевірки поведінки системи при великій кількості одночасних користувачів не входили до обсягу роботи через обмежені обчислювальні потужності.

Попри зазначені виклики, вдалося реалізувати стабільне ядро системи з гнучкою архітектурою, яке демонструє життєздатність концепції та слугує надійним фундаментом для подальшого масштабування проєкту.

4.4 Перспективи розвитку та впровадження

Подальший розвиток системи може відбуватися в кількох напрямках.

Одним із пріоритетних завдань є завершення та розширення User API з реалізацією повних соціальних сценаріїв (коментарі, лайки, «обране», профілі користувачів) на окремому Laravel-сервісі з використанням сучасних механізмів автентифікації, зокрема JWT. Паралельно доцільно розвивати AI-модуль, додаючи автоматичне тегування, розширене резюмування та експерименти з рекомендаціями новин, а також побудувати дашборди для моніторингу використання моделей, якості результатів і витрат.

Перспективним напрямком є виділення окремих мікросервісів для складного парсингу на базі Python. Такий підхід дозволить використовувати потужні бібліотеки екосистеми Python, зокрема BeautifulSoup, Scrapy та Selenium, для обробки динамічного контенту, що генерується JavaScript, а також для роботи з ресурсами, які потребують емуляції браузера. Мікросервісна архітектура забезпечить незалежне масштабування парсерів залежно від навантаження та спростить підключення нових джерел без впливу на основне ядро системи.

Для підвищення ефективності пошуку новин доцільно інтегрувати спеціалізовані пошукові рушії, такі як Elasticsearch або Apache Solr. Ці системи забезпечують повнотекстовий пошук із підтримкою морфологічного аналізу, фасетної фільтрації та релевантного ранжування результатів. Синхронізація даних між PostgreSQL та пошуковим індексом може здійснюватися через механізм подій або періодичну індексацію, що дозволить користувачам швидко знаходити новини за ключовими словами, фразами чи їх комбінаціями навіть у великих обсягах контенту.

З інфраструктурної точки зору наступним кроком є розгортання в промисловому середовищі з налаштуванням VPS або хмарної інфраструктури, використанням Docker Compose або оркестрації контейнерів, reverse proxy (Nginx), SSL-сертифікатів (Let’s Encrypt), а також систем моніторингу та

логування. Клієнтські застосунки – вебклієнт на Next.js та мобільний застосунок на Flutter – можуть бути доведені до повноцінного продакшн-рівня з подальшою публікацією в Google Play та, за потреби, в App Store.

Перспективним напрямком є також розширення переліку джерел шляхом підключення нових локальних і міжнародних новинних порталів різними мовами, що дозволить збільшити корисність агрегатора й посилити ефект від AI-функцій перекладу та резюмування.

Завдяки модульній архітектурі та вже спроєктованій інфраструктурі ці кроки можуть реалізовуватися поетапно, без радикальної перебудови системи.

4.5 Висновки з розділу 4

У четвертому розділі було оцінено ефективність розробленого агрегатора новин, проаналізовано його масштабованість і продуктивність, окреслено основні обмеження та труднощі дослідження, а також визначено перспективні напрями розвитку. Проведений аналіз показує, що обрана архітектура, реалізований функціонал Harvester, REST API та клієнтських застосунків утворюють життєздатну основу для подальшого перетворення розробленої системи на повноцінну новинну платформу з підтримкою AI-функціональності.

ВИСНОВКИ

У магістерській роботі вирішено науково-прикладне завдання розроблення модульного агрегатора новин із підтримкою вебскрейпінгу, REST API та AI-обробки контенту на базі фреймворку Symfony [16] та СУБД PostgreSQL [15]. На основі аналізу існуючих підходів до побудови новинних агрегаторів і сервісів обробки текстових даних було обґрунтовано вибір доменно-орієнтованого проєктування [1], гексагональної архітектури [4] та підходу CQRS як основи для реалізації системи.

У роботі спроєктовано й реалізовано структуру модульного моноліту, до складу якого входять модуль Harvester для збору та нормалізації новин, модуль REST API для зовнішнього доступу до даних, вебінтерфейси користувача й адміністратора, а також модуль підтримки AI-функціональності. Запропонована архітектура забезпечує чітке розділення відповідальностей, ізоляцію доменної логіки від інфраструктурних компонентів та можливість поетапного розвитку окремих модулів.

Розроблено й описано схему двох баз даних PostgreSQL: news-harvester для зберігання доменних сутностей (новини, джерела, категорії, теги, контент різними мовами, результати AI-обробки) та news-rest для зберігання користувачів, API-ключів і журналів звернень до REST API. Такий поділ підвищує прозорість доступу до даних, спрощує супровід і створює передумови для подальшого переходу до мікросервісної архітектури.

Практично реалізовано повний цикл обробки новин: завантаження HTML/RSS-документів, парсинг та нормалізацію структур, дедуплікацію записів, збереження уніфікованих доменних моделей у базі даних, а також AI-обробку (переклад, резюмування, автоматичне тегування) із використанням конфігурованих моделей провайдера OpenAI. Результати цієї обробки інтегровані як у REST API, так і в адміністративний вебінтерфейс, що дозволяє поєднувати автоматичний і ручний режими роботи з контентом.

На основі розробленого REST API створено два клієнтські застосунки: вебклієнт на Next.js [11], який виконує роль «ридера» новин у браузері, та мобільний застосунок на Flutter [7] з окремим бекендом User API на Laravel для облікових записів, коментарів і лайків. Це демонструє практичну застосовність розробленого агрегатора як ядра екосистеми, до якого можуть підключатися різні зовнішні сервіси без прямого доступу до бази даних.

Для забезпечення відтворюваності середовища розробки й спрощення розгортання використано контейнеризацію на базі Docker [6] та Docker Compose. Якість реалізації підтримується за рахунок використання модульних та інтеграційних тестів на PHPUnit [14], статичних аналізаторів коду та інтеграції перевірок у конвеєр GitLab CI/CD, що знижує ризик регресій і полегшує подальший супровід системи.

Отримані результати підтверджують, що запропонований підхід до побудови агрегатора новин з AI-підтримкою є життєздатним і придатним до практичного використання. Подальший розвиток системи може бути пов’язаний із розширенням переліку джерел, поглибленням функціональності AI-модуля, впровадженням повномасштабних навантажувальних випробувань та розгортанням комплексу в промисловому середовищі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Еванс Е. Предметно-орієнтоване проектування (DDD): структуризація складних програмних систем. Київ : Науковий Світ, 2022. 560 с.
2. Закон України «Про авторське право і суміжні права» від 01.12.2022 № 2811-IX. Верховна Рада України. URL: https://zakon.rada.gov.ua/laws/show/2811-20 (дата звернення: 12.07.2025).
3. Закон України «Про захист персональних даних» від 01.06.2010 № 2297-VI. Відомості Верховної Ради України. 2010. № 34. Ст. 481. URL: https://zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 10.07.2025).
4. Мартін Р. Чиста архітектура: мистецтво розробки програмного забезпечення. Харків : Фабула, 2024. 368 с.
5. Behat Documentation. *The BDD framework for PHP*. URL: https://docs.behat.org/en/latest/ (дата звернення: 02.11.2025).
6. Docker Documentation. *Docker overview*. URL: https://docs.docker.com/get-started/overview/ (дата звернення: 06.11.2025).
7. Flutter Documentation. *Build apps for any screen*. URL: https://docs.flutter.dev/ (дата звернення: 25.11.2025).
8. General Data Protection Regulation (GDPR). Regulation (EU) 2016/679. *Official Journal of the European Union*. URL: https://gdpr-info.eu/ (дата звернення: 23.09.2025).
9. Guzzle Documentation. *PHP HTTP client*. URL: https://docs.guzzlephp.org/en/stable/ (дата звернення: 15.10.2025).
10. Laravel Documentation. *The PHP Framework for Web Artisans*. URL: https://laravel.com/docs (дата звернення: 30.11.2025).
11. Next.js Documentation. *The React Framework for the Web*. URL: https://nextjs.org/docs (дата звернення: 28.11.2025).
12. OpenAI API Documentation. *Text generation models*. URL: https://platform.openai.com/docs/guides/text-generation (дата звернення:

20.11.2025).

13. OpenAPI Initiative. *OpenAPI Specification v3.1.0*. URL: https://spec.openapis.org/oas/v3.1.0 (дата звернення: 18.09.2025).
14. PHPUnit Documentation. *The PHP Testing Framework*. URL: https://phpunit.de/documentation.html (дата звернення: 19.11.2025).
15. PostgreSQL Documentation. *The world's most advanced open source database*. URL: https://www.postgresql.org/docs/ (дата звернення: 29.08.2025).
16. Symfony Documentation. *The Symfony Framework*. URL: https://symfony.com/doc/current/index.html (дата звернення: 11.10.2025).
17. Vernon V. Implementing Domain-Driven Design. Upper Saddle River, NJ : Addison-Wesley, 2013. 656 p.
18. Web Scraping Best Practices and Tools. *ZenRows Blog*. URL: https://www.zenrows.com/blog/web-scraping-best-practices (дата звернення: 03.11.2025).