МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: ВЕБ-ПАТФОРМА ДЛЯ АНАЛІЗУ RSS-СМУГ

Здобувача вищої освіти другого (магістерського) рівня,
ІІ курсу, групи КМІ-24-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма: «Інформаційні управляючі системи і технології в економіці»



Лаговського Олеся
Володимировича

Науковий керівник



Ніжегородцев Владислав
Олександрович,
канд. пед. наук, доцент

Керівник
освітньо-професійної
програми



Омельчук Антон
Анатолійович, канд. техн. наук

Допущено до захисту за рішенням кафедри протокол від «02» січня 2026 № 10

Завідувач кафедри



Філоненко Михайло
Миколайович,
канд. фіз.-мат. наук, доцент

Ірпінь – 2026

Реєстрація № 12 19.01.2026 _____
(дата) (підпис відповідальної особи)

Результати перевірки до захисту 02.01.2026 _____
(до захисту, на доопрацювання) (дата) (підпис)

Результат захисту 5 (відмінно) 10.02.2026 _____
(оцінка) (дата) (підпис)

Голова комісії: _____ О.С. Дюбанов
(підпис)

Члени комісії:

_____ А.Ю. Горбовий
(підпис)

_____ М.М. Філоненко
(підпис)

_____ І.Д. Погорєловська
(підпис)

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки вбалах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії,секретаря.
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст кваліфікаційної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	29		10.02.2026 Дюбанов О.С. Горбовий А.Ю. Філоненко М.М. Погорєловська І.Д. Суховецька Л.П.
2. Дотримання вимог щодо оформлення	від 0 до 10	8		
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	5		
4. Рецензент	від 0 до 5	5		
РАЗОМ БАЛІВ	від 0 до 50	47		

Оцінювання кваліфікаційної роботи науковим керівником - 48 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100- бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національноюшкалою	Дата	П.І.Б.: голови, членів комісії, секретаря,
95	A	5	10.02.2026	Дюбанов О.С. Горбовий А.Ю. Філоненко М.М. Погорєловська І.Д. Суховецька Л.П.

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	3 (задовільно)
50-59	E	
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань 12 «Інформаційні технології»
Спеціальність 122 «Комп'ютерні науки»
Освітня програма «Інформаційні управляючі системи і технології в економіці»

Затверджую
Завідувач кафедри _____ Філоненко М.М.
«11» грудня 2024

Завдання на кваліфікаційну роботу
здобувачу вищої освіти
Лаговському Олесю Володимировичу

1. **Тема кваліфікаційної роботи**: Веб-платформа для аналізу RSS-смуг
Затверджена наказом ректора від «10» грудня 2024 № 1852.

2. Строк здачі закінченої роботи «02» січня 2026.

3. **Мета роботи:** створення веб-застосунку з модульною архітектурою, призначеного для автоматизації процесів моніторингу, централізованого збору, агрегування та аналітичної обробки інформаційних потоків у форматі RSS-смуг у режимі реального часу.

Об'єкт дослідження: процес моніторингу, агрегації та аналітичного опрацювання структурованої інформації з відкритих джерел мережі Інтернет через механізм RSS-смуг.

Предмет дослідження: методи, архітектурні підходи та програмні засоби розробки модульних веб-платформ для збору, структурування та аналізу динамічних інформаційних потоків у форматі RSS.

Орієнтовна база дослідження: технічна документація та академічні публікацій.

4. **Вихідні дані до роботи**: моделі та алгоритми, притаманні додаткам модульного типу.

5. **Перелік графічного матеріалу**: рисунки, таблиці які розміщені в додатках.

6. **Зміст кваліфікаційної роботи:**

Розділ 1. Обґрунтування вибору програмного забезпечення для реалізації сучасного додатку

Аналіз сучасних методів обробки та синдикації RSS-смуг. Огляд сучасних інструментів роботи з каналами, методами їх компіляції, валідації та обробки. Висвітлення актуальності такої розробки на момент написання роботи. Аналіз подібних рішень, які були розроблені раніше. Детальний огляд інструментарію утиліт, які можуть бути використані у реалізації архітектурі проекту.

Розділ 2 Особливості то проблеми створення програмних модулів для обробки RSS-смуг.

Описані особливості формату RSS, а також їх вимоги до архітектури програмного забезпечення. Аргументована позиція, щодо використання додатку модульного типу, описані особливості цього формату і відповідні вимоги до архітектури проєкту.

Розділ 3. Проєктування та розробка веб-платформи для обробки RSS-смуг.

Проєктування архітектури додатку. Реалізація моделі. Тестування додатку. UML діаграми.

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційноїроботи	Строк виконання етапів роботи	
		За планом	Фактично
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 15.12.2024	До 15.12.2024
2	**І розділ** (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	28.05.2025	28.05.2025
3	**ІІ розділ** (аналіз експериментальної частини про методики досліджень)	08.09.2025	08.09.2025
4	**ІІІ розділ** (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	17.11.2025	17.11.2025
5	**Висновки**	08.12.2025	08.12.2025
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	22.12.2025	22.12.2025
7	Отримання відгуку керівника	20.01.2026	20.01.2026
8	Отримання зовнішньої рецензії	23.01.2026	23.01.2026

8. Дата видачі завдання «11» грудня 2024

9. Керівник кваліфікаційної роботи _____ Ніжегородцев В.О.

10. Гарант освітньо-професійної програми _____ Омельчук А.А.

11. Завдання до виконання отримав здобувач вищої освіти _____ Лаговський О.В.

АНОТАЦІЯ

Веб-платформа для аналізу RSS-смуг

У сучасних інформаційних системах зростає потреба в автоматизованому отриманні, агрегації та обробці даних з різноманітних джерел. RSS-формат, незважаючи на свій тривалий час існування, залишається універсальним та зручним інструментом розповсюдження контенту, однак на ринку програмного забезпечення відсутні гнучкі автономні модульні рішення, які легко інтегруються у сторонні вебдодатки.

Мета роботи полягає у створенні веб-застосунку з модульною архітектурою, призначеного для автоматизації процесів моніторингу, централізованого збору, агрегування та аналітичної обробки інформаційних потоків у форматі RSS-смуг у режимі реального часу.

Об'єкт дослідження: процес моніторингу, агрегації та аналітичного опрацювання структурованої інформації з відкритих джерел мережі Інтернет через механізм RSS-смуг.

Предмет дослідження: методи, архітектурні підходи та програмні засоби розробки модульних веб-платформ для збору, структурування та аналізу динамічних інформаційних потоків у форматі RSS.

У роботі використано методи системного аналізу, принципи проєктування модульних і багатоконтейнерних систем, REST-орієнтовану архітектуру, технології контейнеризації Docker, а також методи захисту даних і автентифікації користувачів.

Наукова новизна одержаних результатів полягає у розробці архітектури автономного модульного програмного рішення для роботи з RSS-каналами, яке реалізує розмежування відповідальностей між мікромодулями, ізольоване середовище виконання та уніфікований інтерфейс інтеграції з іншими вебдодатками.

Практичне значення. Розроблено функціональний програмний модуль у вигляді вебплатформи з підтримкою REST API та багатоконтейнерної інфраструктури. Запропоноване рішення може бути використане як складова частина комерційних інформаційних систем, аналітичних платформ або новинних агрегаторів.

Кваліфікаційна робота викладена на 65 сторінках, містить вступ, три розділа, висновки, список використаних джерел та додатки.

Ключові слова: RSS-канали, модульна вебплатформа, програмний модуль, вебдодаток, Docker, REST API, обробка інформації, контейнеризація.

Рік виконання кваліфікаційної роботи: 2025.

Рік захисту роботи: 2026

ABSTRACT
Web platform for RSS feed analysis

In modern information systems, the need for automated retrieval, aggregation, and processing of data from various sources is increasing. The RSS format, despite its long period of existence, remains a universal and convenient tool for content distribution; however, the software market lacks flexible autonomous modular solutions that are easily integrated into third-party web applications.

The purpose of the work is to create a web application with a modular architecture, intended for automation of monitoring processes, centralized collection, aggregation, and analytical processing of information flows in the RSS feed format in real time.

The object of the research: the process of monitoring, aggregation, and analytical processing of structured information from open sources of the Internet network through the RSS feed mechanism.

The subject of the research: methods, architectural approaches, and software tools for developing modular web platforms for collecting, structuring, and analyzing dynamic information flows in the RSS format.

The work uses methods of system analysis, principles of designing modular and multi-container systems, REST-oriented architecture, Docker containerization technologies, as well as methods of data protection and user authentication.

The scientific novelty of the obtained results lies in the development of the architecture of an autonomous modular software solution for working with RSS channels, which implements separation of responsibilities between micro-modules, an isolated execution environment, and a unified integration interface with other web applications.

Practical significance. A functional software module in the form of a web platform with support for REST API and multi-container infrastructure has been developed. The proposed solution can be used as a component of commercial information systems, analytical platforms, or news aggregators.

The qualification work is presented on 65 pages, contains an introduction, 3 sections, conclusions, a list of used sources, and appendices.

Key words: RSS channels, modular web platform, software module, web application, Docker, REST API, information processing, containerization.

Year of completion of the qualification work: 2025.

Year of defense of the work: 2026.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ 3
ВСТУП 5
РОЗДІЛ 1. ОБҐРУНТУВАННЯ ВИБОРУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕАЛІЗАЦІЇ СУЧАСНОГО ДОДАТКУ 10
1.1. Аналіз актуальності предметної області 10
1.2. Актуальність розробки RSS 13
1.3. Аналіз існуючих рішень 16
1.4. Вибір інструментів для реалізації 20
1.5. Висновки до розділу 27
РОЗДІЛ 2. ОСОБЛИВОСТІ ТА ПРОБЛЕМИ СТВОРЕННЯ ПРОГРАМНИХ МОДУЛІВ ДЛЯ ОБРОБКИ RSS-СМУГ 29
2.1. Особливості архітектури RSS формату 29
2.2. Характерні особливості програмних модулей 32
2.3. Особливості використання Docker-контейнерів 33
2.4. Розділення середовищ для розробки та користувацького користування 34
2.5. Висновки до розділу 36
РОЗДІЛ 3. ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБ-ПЛАТФОРМИ ДЛЯ ОБРОБКИ RSS СМУГ 37
3.1. Загальна схема програмного продукту 37
3.2. Інформаційне забезпечення програмного продукту 41
3.3. Реалізація модулів системи 45
3.4. Сворення демо моделі 54
3.5. Тестування програмного продукту 55
3.6. Керівництво користувача 58
3.7. Висновки до розділу 59
ВИСНОВКИ 61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 63

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API — Application Programming Interface, інтерфейс програмної взаємодії між компонентами системи.

CMS — Content Management System, система керування вмістом вебресурсів.

CRUD — Create, Read, Update, Delete, базові операції роботи з даними.

JWT — JSON Web Token, механізм автентифікації та передачі захищених даних між клієнтом і сервером.

MVC — Model–View–Controller, архітектурний шаблон побудови програмних застосунків.

PDO — PHP Data Objects, бібліотека для безпечної роботи з базами даних у мові програмування PHP.

REST — Representational State Transfer, архітектурний стиль побудови вебсервісів.

REST API — програмний інтерфейс, побудований відповідно до принципів REST.

RSS — Really Simple Syndication, формат для розповсюдження та агрегації інформаційного контенту.

SQL — Structured Query Language, мова структурованих запитів до баз даних.

UTC — Coordinated Universal Time, універсальний координований час.

URL — Uniform Resource Locator, уніфікований ідентифікатор ресурсу в мережі Інтернет.

JSON — JavaScript Object Notation, текстовий формат обміну структурованими даними.

HTTP — HyperText Transfer Protocol, протокол передавання даних у мережі Інтернет.

HTTPS — захищена версія протоколу HTTP з використанням шифрування.

SSL/TLS — Secure Sockets Layer / Transport Layer Security, криптографічні протоколи захисту з’єднань.

DDOS — Distributed Denial of Service, тип атаки, спрямований на перевантаження сервісу.

CI/CD — Continuous Integration / Continuous Deployment, підхід до автоматизації розробки та розгортання ПЗ.

ВД – веб-додаток.

ВП – веб-платформа.

ПЗ – програмне забезпечення.

ПР – програмне рішення.

ВСТУП

RSS-технологія залишається одним із найбільш ефективних механізмів поширення структурованої інформації в мережі Інтернет, забезпечуючи оперативну передачу новин, статей, оголошень та інших типів контенту в уніфікованому форматі. Попри зростання популярності соціальних мереж та месенджерів, RSS зберігає свою актуальність завдяки стандартизованості, відкритості та відсутності алгоритмічної фільтрації контенту. Формат широко використовується новинними агентствами, блоговими платформами, корпоративними порталами, державними установами та аналітичними системами.

Водночас існуючі інструменти для роботи з RSS-каналами мають суттєві обмеження. Більшість комерційних рішень є закритими системами, що не дозволяють інтеграцію з власними веб-застосунками. Наявні відкриті рішення часто обмежуються базовим відображенням стрічок без можливостей глибокого аналізу, класифікації, фільтрації за складними критеріями або побудови аналітичних звітів.

Сучасні інформаційні системи потребують гнучких модульних рішень, які можна легко інтегрувати у існуючу інфраструктуру без необхідності повної перебудови архітектури. Це особливо важливо для корпоративних середовищ, де веб-застосунки повинні взаємодіяти з різними типами баз даних, системами автентифікації та зовнішніми сервісами. Розробка модульної веб-платформи (ВП) для обробки RSS-каналів, яка забезпечує незалежність від операційного середовища основного застосунку та надає стандартизований програмний інтерфейс, є актуальним завданням для підвищення ефективності роботи з інформаційними потоками.

Мета роботи полягає у розробці модульної ВП для автоматизованого збору, обробки, аналізу та візуалізації даних RSS-каналів із забезпеченням гнучкої інтеграції у сторонні інформаційні системи через стандартизований програмний інтерфейс.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

1. Проаналізувати сучасний стан технології RSS, обґрунтувати актуальність створення спеціалізованої платформи для комплексної обробки RSS-каналів та визначити основні сфери її застосування.

2. Дослідити існуючі ПР для роботи з RSS-стрічками, виявити їх функціональні обмеження, переваги та недоліки, визначити вимоги до проектованої системи.

3. Обґрунтувати вибір технологічного стеку для реалізації модульної ВП з урахуванням вимог масштабованості, портативності, безпеки та зручності інтеграції.

4. Розробити архітектуру багатоконтейнерної системи на основі Docker із забезпеченням ізоляції компонентів, ефективної взаємодії між ними та можливості розгортання на різних операційних системах.

5. Спроектувати структуру реляційної бази даних для зберігання RSS-каналів, публікацій, метаданих користувачів та забезпечення швидкого доступу до історичних даних.

6. Реалізувати серверну частину платформи з використанням PHP 8 та фреймворку Slim для побудови RESTful API з дотриманням принципів модульності та розширюваності.

7. Розробити систему автентифікації та авторизації на основі JWT-токенів із розмежуванням прав доступу для користувачів та адміністраторів.

8. Впровадити механізми захисту від SQL-ін'єкцій, CSRF-атак та інших загроз безпеці веб-застосунків через використання параметризованих запитів та валідації вхідних даних.

9. Створити алгоритми автоматизованого оновлення RSS-каналів, фільтрації контенту за різними параметрами, уніфікації часових міток та аналізу інформаційних потоків.

10. Розробити демонстраційний модуль для обчислення метрик соціального узгодження на основі аналізу контенту RSS-публікацій із практичним застосуванням у фінансових та аналітичних системах.

11. Провести функціональне тестування всіх компонентів платформи,

перевірити стійкість до типових атак та підготувати керівництво користувача для розгортання і інтеграції системи.

Об'єкт дослідження: процеси автоматизованої обробки структурованих інформаційних потоків у форматі RSS у сучасних веб-застосунках та інформаційних системах.

Предмет дослідження: архітектурні рішення, методи та програмні засоби побудови модульних ВП для агрегування, фільтрації, класифікації та аналітичного опрацювання RSS-смуг із використанням контейнерних технологій та RESTful API.

Методи дослідження. У роботі використано методи системного аналізу для дослідження предметної області, визначення вимог до системи та декомпозиції складної задачі обробки RSS, порівняльний аналіз при виборі інструментальних засобів, архітектурне моделювання для побудови діаграм послідовностей, об'єктно-орієнтованого програмування для побудови модульної архітектури, патерни проектування ПЗ, методи контейнеризації додатків на основі Docker, методи криптографічного захисту даних через JWT, методи параметризованих запитів для захисту від SQL-ін'єкцій, методи парсингу та валідації XML-структур, методи функціонального тестування ПЗ, методи нормалізації баз даних та статистичні методи аналізу інформаційних потоків.

Наукова новизна одержаних результатів полягає у наступному:

1. Запропоновано архітектурне рішення для побудови модульної ВП обробки RSS-каналів на основі багатоконтейнерної системи, що забезпечує незалежність від операційного середовища основного застосунку та дозволяє інтегрувати платформу у різнорідні інформаційні системи без модифікації їхньої інфраструктури.

2. Розроблено метод розподіленого тестування модульних компонентів ВП через систему змінних середовища Docker, що дозволяє автоматично перемикатися між продуктивним та тестовим режимами роботи без зміни програмного коду.

3. Удосконалено механізм обробки різнорідних RSS-потоків шляхом розробки уніфікованого алгоритму конвертації часових міток із різних часових поясів у єдиний формат UTC, що забезпечує коректне сортування та фільтрацію публікацій.

4. Запропоновано метод обчислення коефіцієнта соціального узгодження для кількісної оцінки інформаційного фону навколо визначеної теми на основі аналізу контенту RSS-публікацій.

Практичне значення одержаних результатів. Розроблена модульна ВП може застосовуватися в інформаційних системах для автоматизованого агрегування новинних потоків, в аналітичних системах та OSINT-інструментах для моніторингу відкритих джерел, у фінансових платформах для прогнозування поведінки ринків, у корпоративних порталах для збору галузевих новин, у державних інформаційних системах для інформування громадян, а також в освітніх проектах як навчальний матеріал. Використання Docker-контейнерів забезпечує легке розгортання на різних операційних системах, модульна структура дозволяє розширювати функціональність без ризику порушення роботи існуючих компонентів.

Особистий внесок здобувача. Всі основні результати роботи отримані автором особисто. Здійснено аналіз предметної області та існуючих рішень, обґрунтовано вибір технологічного стеку, розроблено архітектуру багатоконтейнерної системи, спроектовано структуру бази даних, реалізовано серверну частину платформи з RESTful API, впроваджено систему автентифікації та механізми захисту, створено алгоритми обробки RSS-каналів, розроблено демонстраційний модуль аналізу, проведено тестування та підготовлено документацію.

Апробація результатів роботи.

Лаговський О.В. Особливості структурних програмних рішень з технологією RSS/ В.В. Лаговський, О.В. Лаговський, В.О. Ніжегородцев, М.М. Філоненко, Л.В. Скасків // Вісник східноукраїнського національного університету імені Володимира Даля № 2 (288) 2025. - С.5-13.

Структура та обсяг роботи. Дипломна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Повний обсяг роботи становить 65 сторінки, включаючи 7 рисунків, 8 додатки. Список використаних джерел містить 38 найменувань.

РОЗДІЛ 1. ОБҐРУНТУВАННЯ ВИБОРУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕАЛІЗАЦІЇ СУЧАСНОГО ДОДАТКУ

1.1. Аналіз актуальності предметної області

RSS є технологією, що забезпечує формування спеціалізованих смуг даних, які можуть бути відображені у структурованому та зручному для сприйняття форматі [3]. Ключовою особливістю RSS-стрічок виступає їх чітко визначена структура. Кожна стрічка містить окремі елементи – публікації, або пости, які, у свою чергу, складаються з низки контейнери-зованих полів. Формат цих полів є стандартизованим та незмінним, що істотно спрощує процес опрацювання даних та робить RSS одним із найбільш придатних форматів для використання в об’єктно орієнтованих мовах програмування. Саме універсальність і структурованість RSS сприяли його широкому розповсюдженню та інтеграції у сучасні інформаційні системи.

Важливим аспектом роботи з RSS є спосіб зберігання отриманих даних. З огляду на потенційно великий обсяг стрічок та потребу в їх постійному оновленні, використання оперативної пам’яті для збереження інформації є недоцільним як з технічної, так і з економічної точки зору. Надмірне споживання ресурсів не приносить жодних переваг та може негативно впливати на масштабованість системи. Саме тому при побудові інструментів для аналізу RSS виникає потреба інтеграції механізмів роботи з базами даних.

Аналізуючи специфіку структури RSS-стрічок, можна зробити висновок, що такий формат або легко конвертується у форму, оптимізовану для зберігання, або навіть може бути використаний без попередньої трансформації. Особливо це актуально для ієрархічних, або документо-орієнтованих баз даних, які здатні зберігати XML-структури у первинному вигляді. Такий підхід забезпечує збереження повної відповідності між вихідними та збереженими даними, мінімізує ризики викривлення інформації

під час конвертації та дозволяє реалізувати функції, що потребують роботи саме з початковою XML-структурою.

Отже, формат RSS у поєднанні з відповідними методами зберігання даних створює гнучке та надійне середовище для формування, опрацювання та подальшого аналізу структурованих інформаційних потоків у сучасних веб-системах.

У свою чергу, реляційні бази даних, хоча й потребують певної конвертації та розподілу інформації між окремими таблицями, також здатні забезпечувати повноцінне зберігання всіх необхідних даних. Попри те, що на перший погляд реляційні сховища можуть здаватися менш зручними для організації збереження RSS-стрічок, детальніший аналіз їх практичного використання показує протилежне. Оскільки найбільш поширеними операціями над RSS-даними є фільтрація та композиція, то у процесі їх виконання дані все одно приводяться до структури, оптимальної для реляційних баз даних, що робить такий спосіб їх зберігання придатним та ефективним.

Додатки, призначені для роботи з RSS-смугами, зазвичай реалізовують більшість із таких функцій:

– функція перегляду наповнення окремих постів або цілої RSS-смуги. Зазвичай ця можливість реалізується не як окрема частина додатка, а через прямий доступ до даних, що зберігаються у базі, однак у низці випадків вона залишається необхідною.

– функція композиції смуг. Під композицією мається на увазі можливість формування нової RSS-стрічки на основі вже існуючих, шляхом комбінування вибраних постів або їхніх частин.

– функція фільтрації смуг і постів. Фільтрація може здійснюватися за різними параметрами контейнерів, такими як автор, дата публікації, мова, ключові слова або інші текстові характеристики.

В окремих випадках, коли технологія RSS використовується не як допоміжний інструмент, а як центральний елемент інформаційної системи, розглядаються також додаткові функції:

– функція створення та наповнення смуг у ручному режимі. Вона є необхідною для форумів, новинних порталів та інших ресурсів, де контент формується безпосередньо у межах системи.

– функція редагування постів та смуг. На відміну від автоматизованої обробки, ця можливість потрібна насамперед у випадках, коли додаток виконує роль повноцінної платформи для формування контенту. Існують і модифіковані варіації цієї функції — наприклад, механізми автоматичної цензури або інструменти машинного перекладу для багатомовних систем, які інтегруються в процес фільтрації чи постобробки даних.

З урахуванням специфіки сучасного ринку ПЗ та зростаючого попиту на інструменти для автоматизованої обробки даних, можна очікувати, що найбільший інтерес до застосування такого роду додатків буде спостерігатися саме серед користувачів Linux-систем. Це передусім пов’язано з високою популярністю операційних систем сімейства Linux у професійному середовищі розробників. Спрощені механізми взаємодії з утилітами, широкі можливості конфігурації програмного середовища та розширена гнучкість у модифікації інфраструктури створюють оптимальні умови для вирішення задач у сфері розробки, що й забезпечує актуальність відповідного ПЗ.

На основі дослідження, проведеного веб-ресурсом W3Techs, можна зробити висновок, що на сьогодні найбільш поширеними Linux-дистрибутивами є Ubuntu та її базова система — Debian [5]. Це зумовлює необхідність приділити особливу увагу забезпеченню сумісності розроблюваного додатка саме з цими платформами. Попри те, що таке обмеження на перший погляд може здаватися суттєвим, у контексті широкої популярності зазначених дистрибутивів серед розробників та їх розвиненої екосистеми інструментів воно, навпаки, створює значні переваги та спрощує реалізацію поставлених задач.

Беручи до уваги те, що результатом кваліфікаційної роботи є не кінцевий користувацький продукт, а базова кнострукція ВП для подальшого розвитку та інтеграції у ширші системи, зазначені особливості набувають особливої ваги. Неможливо наперед передбачити конфігурацію користувацького середовища, так само як і було б недоцільно обмежувати роботу додатка лише однією платформою. Це суперечило б концепції створення універсального, гнучкого та зручного у використанні ПР, придатного для подальшої інтеграції в різні інформаційні системи.

1.2. Актуальність розробки RSS

Формат RSS-стрічок у сучасних інформаційних системах набув значного поширення та фактично став невід'ємним елементом більшості веб-застосунків, орієнтованих на динамічне оновлення контенту. Його можна зустріти у блогових платформах, рекламних модулях, форумах, цифрових журналах, новинних агрегаторах та багатьох інших рішеннях, у яких ключову роль відіграє регулярна зміна даних. У ряді випадків RSS застосовується разом із спорідненими форматами на кшталт Atom, однак ідеологічна структура таких форматів є подібною: обидва забезпечують уніфікований спосіб передачі структурованого контенту. Відповідно, інструменти, що дозволяють ефективно опрацьовувати RSS-стрічки, інтегрувати їх у складнішу систему та здійснювати динамічний аналіз даних, мають високу цінність у сучасній індустрії розроблення веб-рішень.

Разом із тим, сам формат RSS не забезпечує користувацької взаємодії, механізмів навігації або засобів представлення інформації кінцевому користувачеві. Для практичного застосування він повинен функціонувати у комплексі з іншими системними компонентами, зокрема інтерфейсами подання, базами даних, модулями обробки контенту та засобами фільтрації. Це обумовлює потребу у створенні сукупного інструментарію, який би

дозволяв не лише отримувати RSS-дані, а й виконувати їх аналіз, узагальнення, структурування, зберігання та відображення.

Така інтеграційна специфіка визначає необхідність модульності та розширюваності ВП, що розробляється. Модульний характер системи забезпечує можливість багаторазового використання її компонентів у різних типах застосунків, а також спрощує адаптацію під особливості конкретних сценаріїв роботи. У контексті сучасного ринку веб-технологій, де застосунки часто поєднують різнорідні джерела даних, подібна архітектура є ключовою.

Потреба у ВП для аналізу RSS-смуг обумовлена широким спектром сфер, у яких такі стрічки є базовим механізмом передачі даних:

– рекламні платформи. Автоматизація показу рекламних блоків потребує регулярного оновлення інформації про продукти, що практично неможливо забезпечити вручну. RSS-стрічка дозволяє передавати структуровані дані — зображення, описи, відомості про постачальника чи продавця — в єдиному уніфікованому форматі. Якщо ж магазин генерує RSS на своїй стороні, весь процес стає повністю автоматизованим. Також формат спрощує фільтрацію на основі метаданих та поведінкових даних користувача.

– форумні системи. Форумна модель взаємодії передбачає швидке поширення великої кількості коротких повідомлень. RSS особливо підходить для такого типу контенту, оскільки дозволяє передавати дані про автора, текст повідомлення, час публікації та іншу супутню інформацію. Простота структури елементів RSS сприяє застосуванню механізмів модерації та цензурування без додаткових технічних ускладнень.

– блоги, новинні ресурси та соціальні платформи. Структура RSS природно співвідноситься з логічною структурою таких застосунків: заголовок, короткий опис, зображення, автор, дата публікації. Через це багато сучасних контент-платформ або використовують RSS-смуги як первинний спосіб передачі даних, або забезпечують їх автоматичну генерацію.

– веб-системи для комерційних продуктів. Як у рекламних застосунках, так і при побудові внутрішніх каталогів товарів RSS є ефективним способом передачі стандартизованих даних для подальшої обробки веб-інтерфейсом.

– спеціалізовані та державні веб-застосунки. З переходом численних інституцій до цифрової форми взаємодії виникла потреба у швидкому поширенні офіційної інформації. RSS-стрічки дозволяють забезпечити оперативне інформування громадян про зміни у нормативних документах чи важливі події, що робить формат придатним для державних інформаційних систем.

– аналітичні системи та інструменти OSINT. RSS-стрічки відіграють важливу роль у системах відкритої розвідки (OSINT), де критичною є здатність оперативно отримувати й аналізувати великі обсяги даних із відкритих джерел. Офіційні сайти, новинні ресурси, міжнародні організації, дослідницькі лабораторії та інші структури часто публікують оновлення саме у форматі RSS, що дозволяє автоматизувати процес збору інформації. На основі цього формуються інструменти для виявлення тенденцій, фіксації появи певних ключових слів, моніторингу регіональних чи політичних змін.

– фінансові та прогнозні платформи. У сфері інвестицій та фондових ринків RSS використовується для автоматичного збору новин, аналітичних звітів та публікацій, які можуть впливати на динаміку цін на акції чи інші фінансові інструменти. Для алгоритмічних трейдингових систем або платформ, що формують прогнози ринкової поведінки, RSS є зручним способом швидкого отримання інформації з різних джерел без потреби у складних інтеграціях. У поєднанні з методами обробки природної мови RSS стає важливим джерелом даних для оцінки настроїв ринку (sentiment analysis), побудови індикаторів очікувань та виявлення потенційних ризиків ще до того, як вони вплинуть на ціну активів.

Усі наведені приклади демонструють не лише широке застосування RSS-формату, але й актуальність створення ВП, здатної здійснювати комплексний аналіз RSS-смуг, обробляти структури даних різного рівня

складності та забезпечувати інтеграцію з іншими компонентами інформаційної системи. Саме така платформа створює передумови для формування гнучкого інструментарію, що може бути використаний у широкому спектрі сучасних веб-застосунків.

1.3. Аналіз існуючих рішень

Inoreader — це професійний RSS-агрегатор, що забезпечує розширені можливості управління та фільтрації стрічок. Система підтримує великі обсяги джерел, використання тегів, пошук по всьому контенту та інтеграцію з API для зовнішніх ПР. Inoreader орієнтований як на індивідуальних користувачів, так і на корпоративні середовища, дозволяючи ефективно організовувати інформаційні потоки та проводити базову аналітику.

Переваги:

– забезпечує розширені можливості організації та структурування RSS-стрічок;

– має потужну систему тегів і пошуку по всьому масиву даних;

– підтримує функції експорту та імпорту, що полегшує перенесення профілю між платформами;

– передбачає можливість роботи з великою кількістю джерел без зниження продуктивності.

Недоліки:

– не надає доступу до внутрішніх алгоритмів обробки та сортування;

– має обмежену інтеграцію з зовнішніми ПР без використання платних API;

– більшість аналітичних функцій доступна лише у платній версії;

– використання у корпоративному середовищі потребує окремої ліцензії.

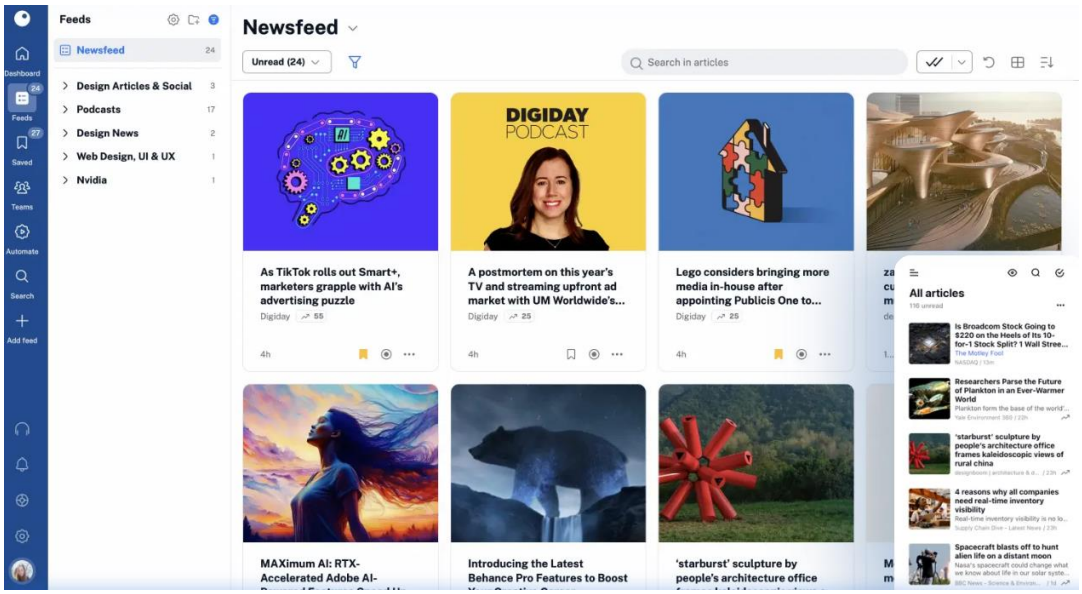


Рис. 1.1. Додаток Inoreader

NewsBlur — це персональний RSS-агрегатор із частково відкритим кодом. Система дозволяє користувачам читати, організовувати та сортувати новини, а також створювати власні папки та фільтри для персоналізації інформаційних потоків. NewsBlur підтримує синхронізацію між різними клієнтськими пристроями та надає базові аналітичні функції для роботи з контентом, хоча глибокий аналіз і розширена інтеграція з іншими системами обмежені.

Переваги:

– підтримує функцію “розумного читання”, що дозволяє автоматично визначати релевантний контент;

– є частково відкритим, оскільки має відкритий серверний компонент;

– підтримує зручну синхронізацію даних між різними клієнтами;

– пропонує розширені можливості кастомізації відображення стрічок.

Недоліки:

– клієнтська частина залишається закритою, що ускладнює розширення функціональності;

– функції аналізу та фільтрації мають обмежений характер;

– відсутні механізми інтеграції з власними веб-застосунками;

– продуктивність системи помітно знижується при роботі з великими обсягами джерел.



Рис. 1.2. Додаток NewsBlur

Meltwater — комерційна платформа для медіамоніторингу та аналітики, що працює з різними типами інформаційних джерел, включаючи RSS. Система дозволяє збирати новини, оцінювати тональність публікацій, аналізувати тренди та генерувати аналітичні звіти. Meltwater орієнтований на корпоративне використання, надаючи можливості моніторингу бренду, конкурентного середовища та суспільної думки.

Переваги:

– створена для професійного медіамоніторингу та має розвинені інструменти аналітики;

– підтримує роботу з RSS як з одним із основних джерел даних;

– забезпечує ефективну візуалізацію інформаційних потоків і трендів;

– має інтегровані алгоритми аналізу тональності.

Недоліки:

– являє собою повністю закрите комерційне рішення без можливості розширення;

– не призначена для використання як модуль у сторонніх системах;

– не підтримує роботу з довільними RSS-структурами;

– вартість підписки є занадто високою для малих проєктів або індивідуальних розробників.

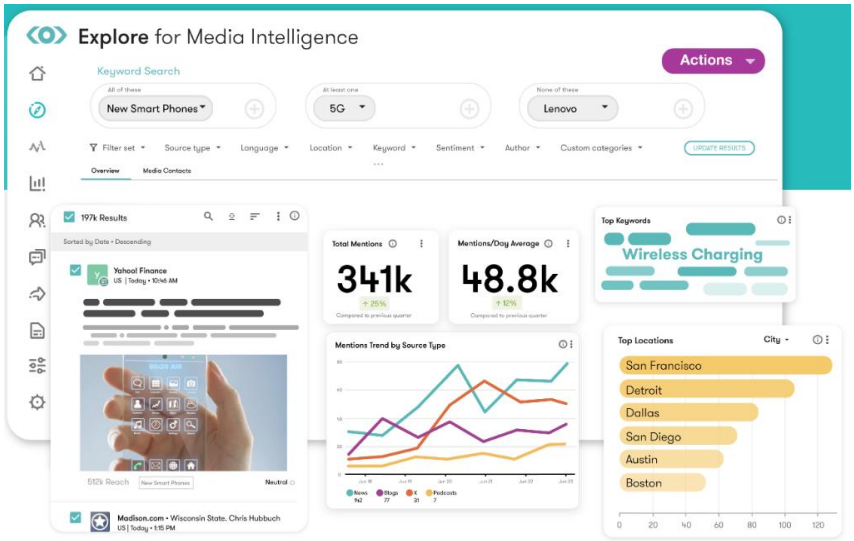


Рис. 1.3. Додаток Meltwater

SpiderFoot (OSINT) - інструмент відкритої розвідки з модульною архітектурою, призначений для автоматизованого збору, аналізу та класифікації даних із відкритих джерел. RSS-стрічки використовуються як один із каналів отримання інформації. Платформа підтримує інтеграцію з різними базами даних та системами візуалізації, що дозволяє реалізовувати складні аналітичні сценарії для оцінки загроз, моніторингу інформаційного середовища та прогнозування подій.

Переваги:

– дозволяє здійснювати автоматизовану обробку даних із великої кількості джерел, включаючи RSS;

– має модульну архітектуру та може розширюватися додатковими плагінами;

– надає широкі можливості для налаштування сценаріїв збору даних;

– доступний у вигляді відкритого ПЗ.

Недоліки:

– механізми роботи з RSS обмежені та не оптимізовані для глибокої аналітики;

– інтерфейс орієнтований переважно на технічних користувачів;

– інтеграція результатів у зовнішні системи потребує окремих розробницьких рішень;

– швидкість обробки значних обсягів RSS-потоків є недостатньою для реального часу.

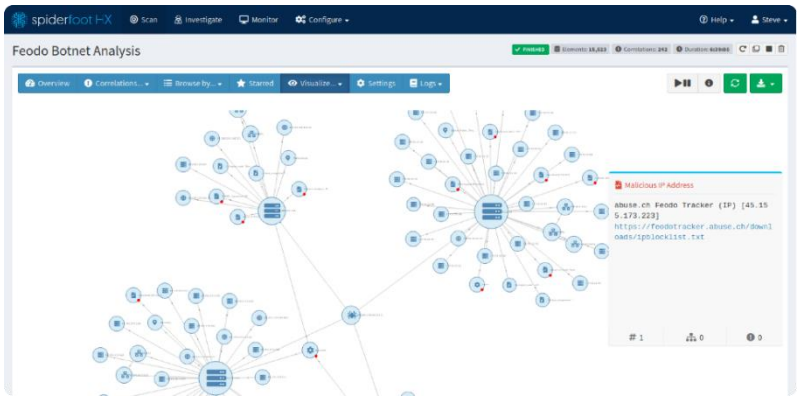


Рис. 1.4. Додаток SpiderFoot

1.4. Вибір інструментів для реалізації

PHP — це широко використовувана мова сценаріїв загального призначення з відкритим вихідним кодом, яка особливо ефективна для розробки ВД та динамічних ВП. Синтаксис мови базується на традиційних мовах програмування, таких як C, Java та Perl, і дозволяє створювати модульні рішення, що легко інтегруються у складні веб-системи. Основна мета використання PHP полягає в тому, щоб забезпечити швидку реалізацію динамічних функцій, включаючи обробку зовнішніх даних, автоматизоване формування контенту та інтеграцію з базами даних.

В контексті ВП для аналізу RSS-стрічок PHP дозволяє ефективно реалізувати модульну архітектуру, де окремі компоненти відповідають за різні функції: завантаження стрічок, їх збереження, фільтрацію, класифікацію та візуалізацію даних. Завдяки вбудованим інструментам для роботи з HTTP-запитами, XML та JSON, PHP дозволяє безпосередньо підключатися до зовнішніх джерел інформації, завантажувати RSS-дані та готувати їх для подальшого аналізу.

PHP підтримує об’єктно-орієнтоване програмування (ООП), що є ключовим при розробці ВП з модульним підходом. Використання об’єктів та класів дозволяє створювати повторно використовувані компоненти, легко підтримувати код та розширювати функціональність платформи. Це забезпечує швидке додавання нових аналітичних алгоритмів або нових видів інтеграції з іншими веб-сервісами без необхідності повної переробки існуючої архітектури. Окремо варто відзначити, що об’єктно-орієнтований код значно полегшує тестування та модифікацію платформених модулів, що важливо для систем, які працюють з великими потоками динамічних даних.

Таким чином, вибір PHP для розробки ВП для аналізу RSS-стрічок є обґрунтованим як з точки зору простоти реалізації, так і з точки зору підтримки модульного підходу. Мова забезпечує ефективну інтеграцію з веб-середовищем, можливості роботи з різними форматами даних і створення повторно використовуваних компонентів, що дозволяє побудувати гнучку, масштабовану та аналітично орієнтовану платформу.

Окремо слід підкреслити, що PHP є слабо типізованою мовою, тобто вона функціонує як динамічно типізована, проте водночас дозволяє встановлювати обмеження на типи змінних. Така особливість поєднує позитивні аспекти як динамічної, так і статичної типізації. У процесі розробки ВП для аналізу RSS-стрічок часто виникає проблема непередбачуваності даних, що надходять від зовнішніх джерел, таких як реляційні бази даних або ретрансляції RSS. Механізм динамічної типізації дозволяє приймати в якості аргументу будь-який тип даних та обробляти його, спираючись на властивості

отриманого об’єкта, з подальшою можливістю приведення даних до статичного типу.

Такий підхід забезпечує коректну взаємодію модулів ВП та дозволяє ефективно налагоджувати механізми обробки різнорідних потоків інформації. Крім того, він зменшує кількість необхідних тестів і підвищує надійність ПР, що особливо важливо у системах, які виконують аналітику великих обсягів даних та забезпечують інтеграцію з різними зовнішніми сервісами.

Важливим уточненням є вибір версії мови PHP, а саме PHP8. Ця версія є на даний момент стабільною і підтримуваною, а також має низку ключових поліпшень порівняно з попередніми релізами. Зокрема, покращена читабельність і зрозумілість коду сприяє більш ефективній розробці модульних компонентів ВП та зручності подальшого супроводу та модифікації системи.

Slim — це шаблон (framework) для мови PHP, спеціалізований на розробці ВД та реалізації RESTful API. Він забезпечує створення зрозумілого та структурованого програмного інтерфейсу для взаємодії з ВП. Використовуючи прості HTTP-запити, можна передавати платформі всі необхідні дані для подальшої обробки та аналізу, що робить Slim ідеальним інструментом для побудови модульних компонентів платформи.

Особливою перевагою Slim є механізм middleware, який буде використаний при реалізації поставлених задач. Middleware дозволяє виділити окремі функції платформи, наприклад безпеку, персоналізацію або збереження даних, у незалежний шар, який можна за потреби відключити або модифікувати без впливу на основний код платформи. Це забезпечує високий рівень ізоляції компонентів і полегшує подальше розширення функціональності або інтеграцію нових модулів.

Враховуючи, що кожна ВП має власний набір функцій і вимог, передбачити конкретні компоненти, які будуть використовуватися разом із нашим модулем для аналізу RSS-стрічок, неможливо. Використання Slim дозволяє створювати гнучку архітектуру, де нові модулі можна легко

підключати, відключати або змінювати без ризику порушення роботи інших частин системи. Таким чином, цей framework сприяє створенню масштабованої, безпечної та легко підтримуваної ВП з модульною структурою.

Composer — це пакетний менеджер для мови PHP, який забезпечує ефективну організацію взаємодії файлів та класів у рамках ВП. Основною його функцією є спрощення процесу імпортування зовнішніх бібліотек та внутрішніх класів, що дозволяє розробникам створювати роздроблену систему компонентів зі складною ієрархією без необхідності вручну узгоджувати файли між собою.

Використання Composer є критично важливим у великих ВД, таких як платформи для аналізу RSS-стрічок, оскільки написання всього коду в одному файлі ускладнює командну роботу, унеможливлює ефективне використання систем контролю версій та обмежує загальну оглядовість архітектури. Composer дозволяє автоматично завантажувати залежності та підтримувати актуальні версії бібліотек, що значно підвищує стабільність і масштабованість платформи.

Завдяки цьому пакетному менеджеру розробники можуть легко інтегрувати сторонні бібліотеки для роботи з HTTP-запитами, обробки XML і JSON, а також реалізації аналітичних алгоритмів для RSS-даних, створюючи гнучкі та повторно використовувані модулі ВП. Крім того, Composer сприяє стандартизації структури проекту та зменшує ризик конфліктів між компонентами, що особливо важливо для платформ, які мають модульну архітектуру та передбачають подальше розширення функціональності.

Docker — це відкрита платформа, розроблена однойменною компанією, яка дозволяє створювати програмні контейнери. Контейнери виконують роль ізольованого середовища, сформованого на основі конфігураційних файлів, і забезпечують передбачувану роботу додатків незалежно від операційної системи.

Використання Docker у ВП для аналізу RSS-стрічок підвищує універсальність і портативність ПР. Завдяки контейнерам платформа може працювати на MacOS, Windows та більшості дистрибутивів Linux без необхідності додаткового налаштування під конкретне середовище. Це дозволяє створювати єдину версію додатку, ігноруючи особливості кожної окремої операційної системи, що спрощує розгортання та підтримку ПР.

Крім того, Docker значно спрощує процес розробки, оскільки кожен розробник може працювати у власному ізольованому середовищі без потреби вручну узгоджувати версії бібліотек, налаштування серверів або конфігурації системи. Узгодженість версій та середовища забезпечується шаблонами контейнерів, що дозволяє зосередитися на розробці аналітичних модулів платформи і взаємодії з потоками RSS-даних, не витрачаючи час на вирішення проблем сумісності.

Завдяки таким можливостям Docker, ВП набуває високої масштабованості, стійкості до змін середовища та простоти інтеграції нових модулів, що є важливими аспектами при побудові аналітичних систем із модульною архітектурою.

Docker Compose — це інструмент для визначення та запуску багатоконтейнерних програм, який дозволяє значно спростити процес розробки та розгортання контейнерів. Compose надає можливість зібрати всі конфігурації, що стосуються мереж, томів та інших служб, в один файл формату YAML. Це забезпечує зручне та централізоване управління системою, а за допомогою однієї команди можна створити та запустити всі служби, визначені у файлі конфігурації.

Для ВП з модульною архітектурою, що аналізує RSS-стрічки, Docker Compose спрощує процес запуску та встановлення платформи для користувачів. У конфігураційному файлі можна модифікувати важливі параметри, такі як розташування томів для окремих контейнерів, дані профілів доступу до бази даних та порти, що прослуховуються службами платформи.

Особливу увагу слід приділити тому, що централізовані конфігураційні файли легко адаптувати для різних етапів тестування та експлуатації платформи. Наприклад, під час перевірки або розробки модулів безпеки немає потреби створювати контейнери баз даних. Крім того, завдяки різним конфігураціям можна розділити середовища на тестові та реальні бази даних, що не лише підвищує безпеку даних, а й забезпечує оптимальну швидкодію системи під час розробки та тестування.

Таким чином, Docker Compose є важливим інструментом для побудови гнучкої, масштабованої та безпечної ВП з модульною архітектурою, що працює з потоками RSS-даних та іншими аналітичними сервісами.

Makefile — це спеціальна утиліта для автоматизації роботи з bash-терміналами, яка дозволяє значно спростити виконання комплексних команд та процедур. У контексті ВП для аналізу RSS-стрічок Makefile дозволяє автоматизувати запуск Docker-контейнерів, встановлення необхідних мов програмування, бібліотек та інших залежностей. При правильному налаштуванні, комплексні задачі, такі як підготовка зовнішнього середовища, конфігурація серверу та запуск платформи, можуть виконуватися однією командою, що зменшує ризик помилок і підвищує ефективність розробки та розгортання платформи.

Nginx — це веб-сервер і утиліта для реалізації зворотних проксі-серверів, яка дозволяє створювати ізольовану та ефективно керовану екосистему для серверної частини ВП. У поєднанні з Docker-контейнерами, Nginx забезпечує організовану та безпечну роботу всіх внутрішніх служб, зокрема серверного інтерфейсу на PHP. Ця утиліта дозволяє ефективно керувати запитами, балансувати навантаження між компонентами платформи та підтримувати стабільну роботу ВП при високих обсягах обробки RSS-даних.

Використання Makefile та Nginx у комплексі з Docker та Docker Compose забезпечує високу автоматизацію, портативність та масштабованість ВП.

Visual Studio Code (VS Code) — це середовище розробки ПР, створене компанією Microsoft. Воно підтримує перевірку синтаксису для більшості популярних мов програмування, включаючи PHP, що робить його ефективним інструментом для створення ВП з модульною архітектурою.

Основною перевагою VS Code є його зручність при розробці багатомовних проектів, що особливо важливо для платформ, які інтегрують різні технології та модулі для обробки RSS-даних. Крім того, відкритий вихідний код середовища дозволяє створювати незалежні розширення, які можуть автоматизувати специфічні завдання, полегшити роботу з бібліотеками або забезпечити інтеграцію з іншими сервісами, наприклад, базами даних, системами контролю версій або інструментами для тестування.

Завдяки цим можливостям, VS Code забезпечує гнучке та ефективне середовище для розробки, підтримки та модифікації ВП, що дозволяє розробникам зосередитися на логіці обробки RSS-стрічок та аналітичних функціях, не витрачаючи зайвий час на налаштування робочого середовища.

Postman — це інструмент для тестування та налагодження API, який широко використовується у веб-розробці. Він дозволяє створювати запити до серверних компонентів, перевіряти їх правильність та відстежувати відповіді, що особливо важливо для платформ, які взаємодіють із зовнішніми сервісами, такими як RSS-стрічки.

Для ВП з модульною архітектурою Postman забезпечує ефективне тестування API, яке реалізується через шаблон Slim і PHP. Використовуючи Postman, розробник може перевіряти роботу окремих модулів платформи, наприклад, обробку запитів до бази даних, фільтрацію постів, формування аналітичних звітів або інтеграцію з зовнішніми RSS-джерелами.

Особливою перевагою є можливість автоматизованого тестування, що дозволяє створювати сценарії для перевірки взаємодії різних компонентів платформи без необхідності ручного запуску запитів. Крім того, Postman підтримує експорт колекцій тестів і їх інтеграцію з системами CI/CD, що підвищує надійність та стабільність роботи ВП при розгортанні оновлень.

Завдяки своїй гнучкості та функціональності, Postman є незамінним інструментом для налагодження та підтримки ВП, забезпечуючи ефективну перевірку роботи всіх модулів і їх взаємодії із зовнішніми потоками RSS-даних.

1.5. Висновки до розділу

У ході написання першого розділу був проведений детальний аналіз предметної області кваліфікаційної роботи, під час якого розглянуто базові принципи функціонування технології RSS-стрічок, її основні переваги та сфери застосування. Результати цього аналізу дозволили обґрунтувати актуальність теми розробки ВП для аналізу RSS-стрічок.

Було встановлено, що на сучасному етапі технологія RSS має широке розповсюдження та використовується у різних сферах, включаючи блоги, форуми, цифрові журнали, соціальні мережі, рекламні платформи та спеціалізовані додатки для аналітики. Водночас існує ряд невирішених задач, зокрема потреба у єдиній платформі з модульною архітектурою, яка забезпечувала б ефективну обробку, фільтрацію та аналітичне представлення даних з RSS-стрічок.

Аналіз конкуруючих веб-рішень показав наявність суттєвих обмежень існуючих платформ, серед яких: обмежена гнучкість інтеграції з іншими системами, відсутність доступу до вихідного коду, обмежені можливості фільтрації даних та відсутність підтримки певних операційних систем. Ці недоліки були враховані при визначенні вимог до проектованої платформи та її модульної архітектури.

З урахуванням потреби у створенні універсального ВД для аналізу RSS-стрічок, було проведено оцінку сучасних інструментів та технологій, здатних забезпечити ефективну реалізацію модульного підходу. В результаті було обрано наступний стек технологій:

- основна мова програмування: PHP8 — об’єктно-орієнтована мова зі слабкою та динамічною типізацією, що дозволяє ефективно працювати з різнорідними даними RSS-стрічок та забезпечує повторне використання коду;
- шаблон для розробки API: Slim v4 — спрощує реалізацію модульного серверного інтерфейсу та підтримує middleware для ізоляції функцій;
- пакетний менеджер: Composer — забезпечує злагоджену роботу файлів та класів, спрощує управління залежностями та підтримку ієрархії модулів;
- платформа для контейнеризації: Docker — дозволяє створити уніфіковане середовище для роботи ВП, забезпечуючи кросплатформеність та стабільність;
- інструмент для управління багатоконтейнерними конфігураціями: Docker Compose — спрощує запуск і тестування всієї системи через один файл конфігурації;
- утиліта для автоматизації завдань у терміналі: Makefile — дозволяє автоматизувати запуск платформи та підготовку середовища однією командою;
- веб-сервер і проксі: Nginx — забезпечує ефективну організацію запитів, балансування навантаження та хостинг PHP-сервера;
- середовище розробки: V S Code — надає зручний інтерфейс для роботи з багатомовними проектами та підтримує створення розширень для специфічних задач;
- інструмент для тестування API: Postman — забезпечує налагодження та автоматизоване тестування взаємодії між модулями та зовнішніми RSS-джерелами.

Використання вищенаведених технологій дозволяє створити модульну ВП для аналізу RSS-стрічок, яка буде легко масштабованою, інтегрованою з іншими сервісами та універсальною для використання на різних операційних системах.

РОЗДІЛ 2. ОСОБЛИВОСТІ ТА ПРОБЛЕМИ СТВОРЕННЯ ПРОГРАМНИХ МОДУЛІВ ДЛЯ ОБРОБКИ RSS-СМУГ

2.1. Особливості архітектури RSS формату

RSS-смуга — це файл формату XML, що має чітко визначену ієрархічну структуру елементів. Головним кореневим елементом документа є тег <rss>, всередині якого містяться всі інші структурні елементи. Цей елемент має один обов'язковий атрибут — version, який визначає версію стандарту RSS. Наявність цього атрибуту зумовлена тим, що на сьогодні існує декілька варіацій RSS-форматів, кожен з яких має свої особливості синтаксису. Відповідно, наявність атрибуту version дозволяє ПЗ коректно інтерпретувати структуру файлу залежно від використаного стандарту.

Наступним за ієрархією є елемент <channel> (у перекладі — канал), який не містить обов'язкових атрибутів. Формально структура RSS допускає наявність декількох каналів у межах одного файлу, проте така модель на практиці вважається недоцільною. Навіть під час об'єднання кількох інформаційних потоків, розробники зазвичай формують окремий об'єднаний канал, що спрощує подальшу обробку даних. Єдиним частковим винятком можуть бути випадки мультомовних стрічок, хоча і тут частіше застосовують створення окремих файлів для кожної мови.

Елемент <channel> містить кілька обов'язкових дочірніх елементів:

<link> — посилання на веб-сторінку з візуальним відображенням вмісту каналу. Якщо ресурс не має веб-інтерфейсу, у цьому елементі наводиться посилання на сам RSS-файл.

<title> — назва каналу.

<description> — короткий опис каналу, який може містити інформацію про тематику або призначення стрічки.

Окрім обов’язкових полів, у канал можуть входити численні необов’язкові елементи. З огляду на специфіку поставленої задачі, більшість з них не матиме практичного значення та може бути відкинута під час обробки. Проте деякі з додаткових елементів залишаються важливими навіть у цьому контексті. Наприклад, елемент:

<language> — визначає мову каналу. Його використання відкриває можливість автоматизованої фільтрації новин за мовою або ж автоматичного застосування механізмів перекладу при обробці контенту. Крім того, фільтрація за мовою може бути адаптивною, якщо веб-застосунок має доступ до метаданих користувача, зокрема до мови його операційної системи.

Елемент <item> є складовою частиною каналу RSS та використовується для опису окремого повідомлення або публікації. На відміну від інших елементів, він може повторюватися необмежену кількість разів, оскільки кожен <item> відповідає окремій сутності контенту.

Подібно до елемента <channel>, кожний <item> містить базові структурні елементи — <link>, <title> та <description>, які відповідно містять посилання на публікацію, її назву та короткий опис.

До найпоширеніших необов’язкових елементів <item> належать:

<author> — містить інформацію про автора публікації (електронну адресу або ініціали);

<category> — визначає категорію чи тематику публікації, що може бути використано для подальшої фільтрації;

<pubDate> — містить дату та час публікації;

<source> — вказує первинне джерело публікації, тобто RSS-канал, з якого спочатку була отримана інформація.

Останні два елементи мають особливе значення. Під час композиції кількох RSS-каналів у єдиний потік назву каналу, з якого походить публікація, зазвичай переносять до елемента <source>, тоді як сам <channel> залишається унікальним для об’єднаної стрічки. Таким чином забезпечується збереження походження кожного окремого запису.

Особливу складність викликає обробка часу, оскільки різні мови програмування та системи використовують власні формати дат і часових міток. Для усунення неоднозначності RSS 2.0 визначає стандартний формат представлення дати в елементі <pubDate>. Він має вигляд:

"DDD, dd MMM YYYY HH:ii:ss T"

Розшифрування формату: DDD — трилітерне позначення дня тижня англійською мовою (Mon, Tue, …); dd — двосимвольний номер дня місяця; MMM — трилітерне позначення місяця (Jan, Feb, …); YYYY — чотиризначний номер року; HH — година у 24-годинному форматі; ii — хвилини (двосимвольне позначення); ss — секунди; T — позначення часової зони. Останній параметр є найваріативнішим, оскільки може бути поданий у вигляді стандартних абревіатур, зсуву відносно UTC або альтернативних позначень, залежно від джерела.

Таким чином, елемент <item> є ключовим структурним блоком RSS-стрічки, який містить усю суттєву інформацію про окрему публікацію, включаючи її метадані та контекст походження.

Підсумуємо вищеописане діаграмою залежностей об’єктів (рис. 2.1)

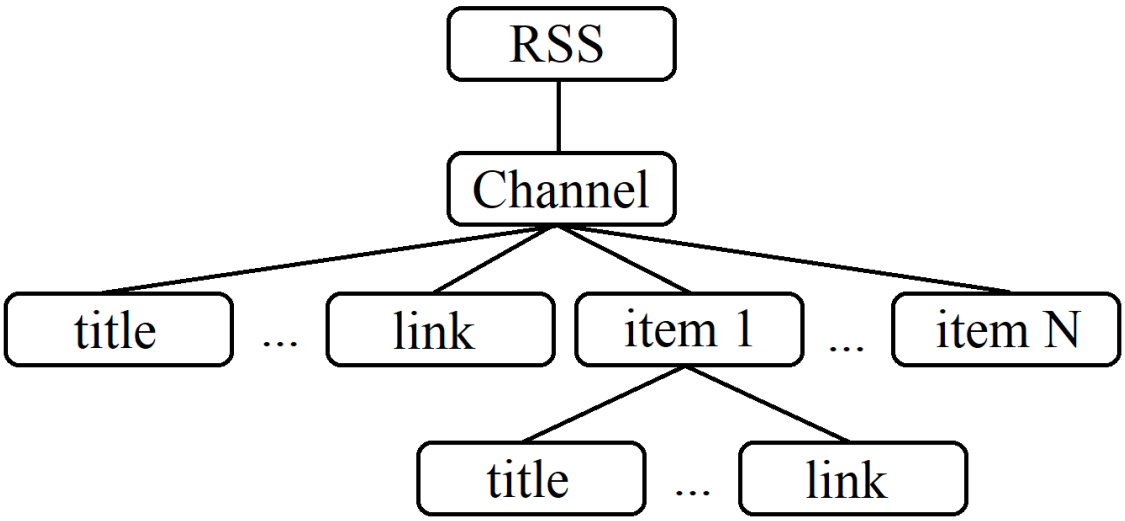


Рис. 2.1. Діаграма залежностей об’єктів в RSS-смузі

2.2. Характерні особливості програмних модулей

Масштабність спроєктованого ПЗ є одним із ключових питань під час створення сучасних додатків. Для того щоб знайти раціональну відповідь, необхідно коректно визначити умови та сценарії використання ПР.

Великі сервісні системи здатні охоплювати широкий спектр проблем, однак вони мають істотний недолік — високу залежність від зовнішніх технологій. Чим більший застосунок, тим більше інструментів, фреймворків та бібліотек він потребує. Це ускладнює його застосування як допоміжного рішення для локальної задачі, оскільки вимагає адаптації як коду, так і середовища для забезпечення сумісності. Саме тому подібні системи використовують лише у випадках, коли їх функціональність є критично необхідною.

На відміну від великих сервісів, додатки модульного типу орієнтовані на значно менший набір задач, проте мають суттєву перевагу — простоту інтеграції. Під час їх розробки питання адаптивності зазвичай беруть на себе автори модуля: вони забезпечують універсальність інтерфейсів, мінімальні залежності й чітку документацію. Саме тому модульні рішення найчастіше застосовуються для компактних, локальних задач.

На основі цього можна визначити ключові особливості архітектури модульної ВП:

– вузька спеціалізація та точкова функціональність;

– мінімальне та раціональне використання технологій та сторонніх утиліт;

– наявність зрозумілого програмного інтерфейсу для взаємодії з іншими частинами системи.

Окремої уваги потребують автономні модулі. Вони функціонують у власному середовищі та отримують команди від основного застосунку, на відміну від інтегрованих модулів, що працюють лише як частина великої системи. Такий підхід виправданий, коли модуль виконує незалежні процеси

або повинен залишатися постійно активним. Найпоширеніший приклад — база даних, яка хоч і не обчислює код постійно, але регулярно обробляє запити, і тому повинна бути доступною безперервно.

Для реалізації ефективної модульної архітектури необхідно визначити головну задачу модуля та сформулювати набір його кінцевих функцій, уникаючи надмірно загальних цілей, що можуть потребувати зайвих залежностей, бібліотек або каркасів. Особливо важливо забезпечити незалежність модуля від конкретних версій мов програмування чи середовищ, оскільки це ускладнює його використання та суперечить принципам модульності.

При проєктуванні програмного інтерфейсу взаємодії з модулем небажано використовувати спеціалізовані сторонні застосунки або важкі бібліотеки для передачі інформації. Достатньо застосувати механізм прямого зв’язку між застосунками або стандартні методи комунікації, наприклад роботу через локальні порти. Окремо слід зазначити, що обмін даними через файлову систему є небажаним, оскільки створює ризики зниження продуктивності та збільшення затримок.

2.3. Особливості використання Docker-контейнерів

Docker — це інструмент, що надає можливість запускати ізольовані робочі середовища, відомі як контейнери. У таких середовищах можна змінювати системні параметри та конфігурації без ризику вплинути на основну операційну систему. Завдяки цьому розробники отримують простір для експериментів та створення специфічних умов, які у звичайній системі було б складно або небезпечно відтворити.

Однією з ключових переваг Docker є можливість формувати шаблони контейнерів і використовувати їх на різних пристроях. Це забезпечує однакові налаштування незалежно від середовища, що значно полегшує роботу в команді та зменшує витрати часу на конфігурацію робочих станцій. Крім того,

можливість швидко створювати різні варіанти середовищ є корисною під час тестування ПР.

В контексті модульної архітектури Docker допомагає усунути проблеми сумісності між модулем і основним застосунком. Контейнер можна підготувати таким чином, щоб він працював незалежно від конкретної версії системи чи мовних бібліотек, що позбавляє необхідності адаптувати модуль під кожне окреме середовище.

Оскільки контейнери спілкуються з головною системою через мережеві порти, вони природно підходять для реалізації автономних компонентів, які функціонують окремо від основного застосунку. За бажанням взаємодію можна організувати через один порт, що дає змогу уникнути зайвої складності у мережевій інфраструктурі.

Додаткові можливості з’являються при використанні контейнера з Nginx, який дозволяє створити внутрішню схему маршрутизації. Завдяки цьому кілька контейнерів можуть працювати як єдина система, обмінюючись даними між собою всередині групи, але виходячи назовні через один спільний інтерфейс. Такий підхід полегшує побудову складних модульних рішень, де потрібні внутрішні служби, зокрема й власні бази даних, без надмірного навантаження на канали взаємодії з основною системою.

2.4. Розділення середовищ для розробки та користувацького користування

Під час розробки серверних систем часто виникає труднощі, пов’язані з оновленням чи розширенням уже працюючого API. Пряме тестування нових фрагментів коду на активному середовищі неможливе: запуск додаткового екземпляра того ж застосунку зазвичай потребує значних ресурсів, а робота другого інстансу з продуктивними даними створює додаткові ризики. Якщо ж обидві копії програми взаємодіятимуть із реальною базою клієнтів, збільшується ймовірність витоку інформації або пошкодження даних. Навіть

якщо ці дані дублювати, загроза безпеці не зникає, а навантаження на інфраструктуру все одно росте. Крім того, така архітектура значно ускладнює написання коректних тестів та аналіз поведінки системи при внесенні локальних змін — часто складно зрозуміти, який саме компонент спровокував помилку.

Для усунення таких проблем широко використовують Docker. Завдяки контейнерам можна підняти окрему тестову базу даних, яка працює так само, як і основна, але не містить реальної інформації. Це спрощує перевірку функціональності й дає можливість ізолювати тестування окремих частин застосунку. Для імітації взаємодії з іншими модулями можуть застосовуватися спеціальні сценарії, які повертають прогнозовані відповіді. Без додаткових інструментів створення таких механізмів означало б наявність різних версій коду для тестування та робочого середовища, що ускладнює розробку.

Оптимальним рішенням стає використання системних змінних. Вони дозволяють програмі автоматично перемикатися між тестовим та реальним режимом роботи. Наприклад, конфігураційна змінна може задавати, до якої саме бази слід підключатися — тестової чи продуктивної. За аналогічним принципом модуль може визначати, чи потрібно звертатися до реального алгоритму або ж до імітованого сценарію, що спрощує процес перевірки. Таким чином, один і той самий код може використовуватися у двох режимах — повному та спрощеному — лише з урахуванням значень змінних середовища.

Docker Compose, у свою чергу, дозволяє поєднувати кілька контейнерів у спільну інфраструктуру і керувати нею через один конфігураційний файл. Завдяки цьому можна легко створювати різні варіанти середовищ для тестування, розробки або розгортання — і запускати їх однією командою. Це суттєво зменшує складність роботи з багатокомпонентними системами, що взаємодіють між собою, і саме тому такі інструменти стали стандартом у сучасній розробці модульних серверних застосунків.

2.5. Висновки до розділу

Проаналізовано принципи роботи з RSS-стрічками та окреслено ключові труднощі, що виникають під час створення серверних застосунків. Розглянуто специфіку взаємодії між частинами багатомодульних систем та проблеми, з якими стикається розробник у таких умовах. Описано набір технологій і підходів, які використовуватимуться під час розробки програмного модуля для їх подолання.

При вивченні структури RSS-файлів було сформульовано особливості організації елементів та принципи побудови ієрархії даних. Детально досліджено механізми обробки RSS-записів, що стануть основою для імплементації відповідного функціоналу. Окремий акцент зроблено на роботі з датами: розглянуто формат зберігання часу, його особливості та методи коректного перетворення. Отримані відомості стануть базою для реалізації механізму уніфікації часових поясів у межах одного каналу.

Проаналізовано архітектуру ПР, побудованих у форматі набору модулів. Визначено інструменти, що застосовуються в сучасній розробці для організації взаємодії між модулями, та окреслено параметри, які відрізняють модульну структуру від сторонніх сервісів. Розглянуто поняття незалежності модулів, наведено їхні переваги та потенційні обмеження, а також обґрунтовано, чому певні способи комунікації є більш ефективними.

Окрему увагу приділено інструментам контейнеризації, зокрема Docker, Docker Compose. Пояснено принципи їх роботи та ролі у створенні модульних систем. Підкреслено актуальність використання цих технологій та описано проблеми, які вони дозволяють усунути.

Розглянуто практичний приклад застосування контейнерів для оптимізації командної роботи. Окреслено труднощі тестування багатомодульних застосунків та наведено можливі шляхи їх подолання. Обґрунтовано доцільність створеня окремих середовищ для етапів розробки та тестування програмного модуля.

РОЗДІЛ 3. ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБ-ПЛАТФОРМИ ДЛЯ ОБРОБКИ RSS СМУГ

3.1. Загальна схема програмного продукту

Під час роботи зі сторонніми компонентами замовники зазвичай очікують повного доступу до вихідного коду. Це пов’язано з необхідністю подальшого розвитку ВП: додавання нових можливостей, підтримки актуальності функціоналу та адаптації до змін бізнес-логіки. Тому читабельність та структурованість коду має не менше значення, ніж перелік його функцій.

Щоб взаємодія з платформою була простою, а процес доопрацювання - передбачуваним, важливо дотримуватися загальних стандартів оформлення коду, логічної структури каталогів та правил документування. Документація має пояснювати не лише призначення окремих частин системи, але і контекст їх виконання у межах багатомодульної ВП.

Перед формуванням фінальної архітектури слід узагальнити вимоги.

1. Платформа повинна мати модульну будову, тобто кожен компонент має бути ізольованим та здатним інтегруватися з іншими елементами системи. Для контейнеризації та розмежування середовищ застосовуються Docker, Docker Compose та Nginx. Це потребує коректно організованої структури файлів, де конфігурації розміщені у передбачених каталогах.

2. Обмін даними між модулями та основними сервісами здійснюється через HTTP-інтерфейси. Для реалізації API обрано PHP у поєднанні з фреймворком Slim, що вимагає створення окремого серверного середовища.

3. Платформа використовує власну систему зберігання даних. Для цього застосовується MySQL, яку необхідно налаштувати з урахуванням роботи у контейнеризованому середовищі.

4. Система повинна підтримувати легке розширення та модифікування. Код має бути поділений на логічні мікросервіси або субмодулі з мінімальними залежностями, щоб внесення змін не призводило до порушення роботи інших частин платформи.

5. Платформа повинна мати режим розробки. Перемикання між робочим та дев-режимом реалізується через системні змінні у контейнерах Docker, що потребує відповідних змін у налаштуваннях контейнерів та програмному коді.

Опираючись на ці вимоги отримуємо наступну діаграму послідовностей дій при обробці запиту користувача (рис. 3.1.)

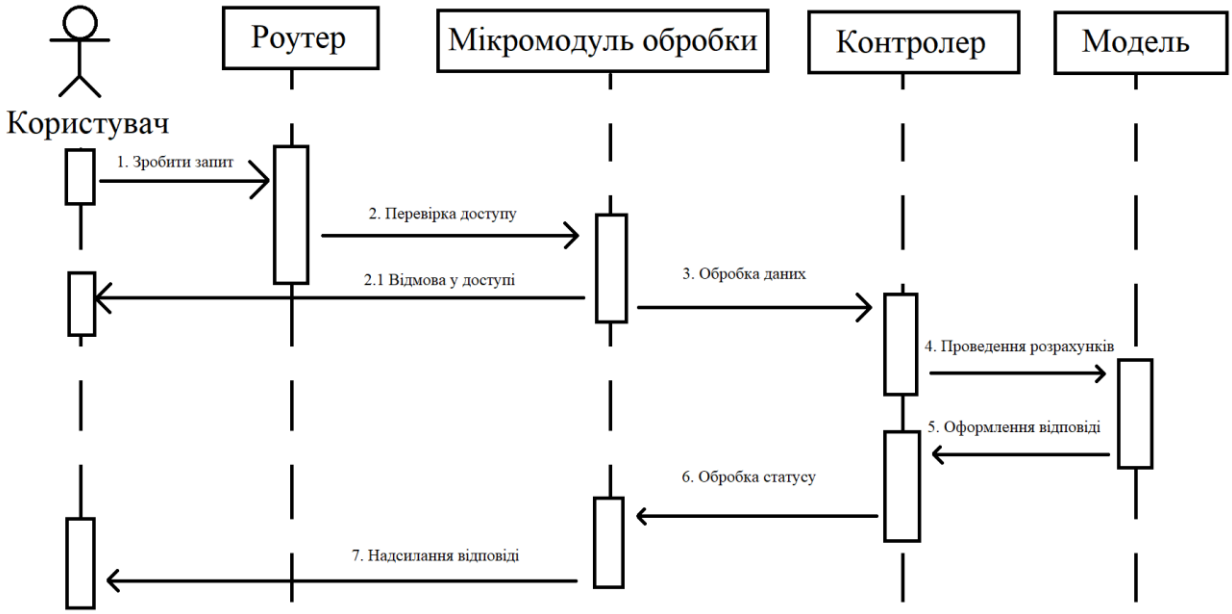


Рис. 3.1. Діаграма послідовностей дій при обробці запиту користувача

Розглянемо структуру платформи.

1. Візуальний шар - компонент, що відповідає за формування та відображення кінцевого контенту. Оскільки ключове завдання системи полягає в опрацюванні RSS-потоків, цей шар не є критично необхідним. Він задіюється лише у випадках, коли потрібне додаткове налаштування або

формування користувацького RSS-каналу, який у майбутньому може використовуватися зовнішніми клієнтами.

2. Маршрутизатор - елемент, який приймає вхідний HTTP-запит, визначає його тип та передає далі у відповідні частини платформи. Він забезпечує коректне перенаправлення запитів між внутрішніми модулями.

3. Блок супровідної логіки - проміжний рівень, де розміщені допоміжні механізми: авторизація, валідація, обробники виключень, засоби безпеки та інші операції, що не належать до основної бізнес-логіки, але впливають на якість взаємодії між клієнтом і платформою.

4. Контролер - частина, яка перетворює зовнішній запит у формат, придатний для подальшої обробки. Контролер також формує відповідь: отримує технічні дані від бізнес-логіки і переводить їх у структуру, зручну для передачі через API.

5. Модель - центральний елемент системи та місце, де виконується основна бізнес-логіка платформи. Тут відбувається обробка даних, застосування алгоритмів та взаємодія із внутрішніми сервісами. Модель працює виключно з технічними об’єктами та структурами, не враховуючи візуальне подання.

6. Основна база даних - сховище, у якому містяться всі робочі дані платформи. Таблиці мають оптимізовану структуру, орієнтовану на швидке виконання запитів у продакшн-середовищі.

7. Тестове сховище даних - окремий екземпляр бази, призначений для моделювання та перевірки роботи модулів без впливу на реальні дані. Має спрощену або скорочену структуру, що дозволяє швидко формувати тестові сценарії.

Таким чином, логіка платформи розділена на набір ізольованих компонентів, і кожен із них відповідає лише за власний сегмент роботи. Подібний підхід суттєво полегшує підтримку системи, спрощує читання коду та дозволяє змінювати або розширювати окремі частини без ризику зачепити інші модулі. Крім того, така організація забезпечує можливість повністю

переписати будь-який мікромодуль - за умови, що його інтерфейси взаємодії з іншими частинами платформи залишаються стабільними.

Щоб реалізувати подібну логіку на практиці, структура модулів має бути відтворена й у файловій системі проєкту. Оскільки платформа будується на PHP, її розгортання вимагає формування кількох контейнерів, що зумовлює свої особливості у формуванні архітектури директорій. У кореневій теці зберігаються конфігураційні файли, необхідні для запуску середовища, а також каталоги, що відповідають за окремі мікрокомпоненти системи.

Враховуючи загальну логіку побудови платформи, можна виокремити три ключові групи директорій: блок візуальної частини, API-підсистему та шар, що забезпечує взаємодію між ними. Кожен із цих елементів містить окрему директорію з docker-конфігураціями. Ті компоненти, що працюють разом із nginx, мають додаткові каталоги, у яких розміщуються відповідні конфігураційні файли. За потреби SQL-скрипти для створення таблиць або допоміжних структур розміщуються окремо, щоб забезпечити прозоре та передбачуване розгортання баз даних.

Якщо певна частина платформи виконує власний програмний код, її структура має бути поділена на відкриті (публічні) елементи та внутрішню логіку, що не повинна бути доступною зовні. Така сегментація дозволяє підтримувати безпеку, забезпечує захист внутрішніх механізмів і спрощує подальші зміни в системі.

Під час формування внутрішньої структури каталогу з виконуваним кодом першочергово слід враховувати особливості мови програмування, доступні механізми завантаження класів та прийняті галузеві стандарти. Оскільки у межах кваліфікаційної роботи серверна частина реалізована мовою PHP, логіка проєкту має бути узгоджена з принципами роботи Composer — стандартного інструмента для автоматичного підключення залежностей та автозавантаження класів.

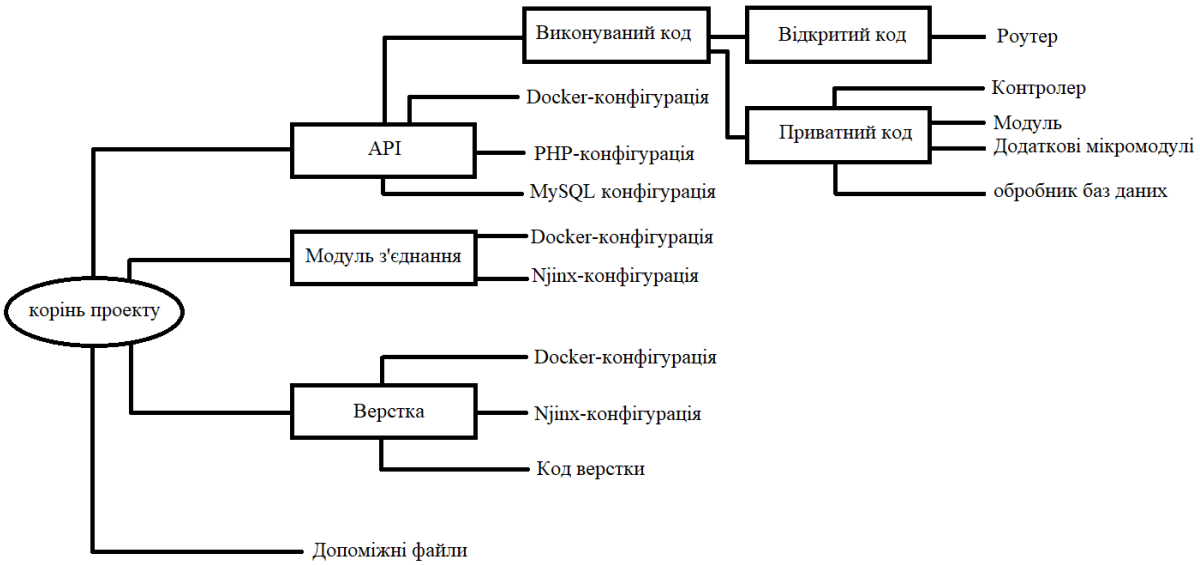


Рис. 3.2. Схематичне зображення ієрархії файлів проекту

Composer орієнтується на набір рекомендацій PSR, серед яких ключову роль відіграє стандарт PSR-4. Він визначає спосіб зіставлення простору імен класів зі структурою директорій. За цим підходом шлях до файлу має повторювати ієрархію namespace, а назва класу збігається з назвою файлу. Це означає, що кожен файл зберігає лише одну класову сутність, що унеможливлює накопичення декількох реалізацій в одному місці.

Як наслідок, навіть фінальний рівень структури каталогу містить значну кількість файлів, де кожний відповідає за окремий об’єкт або невеликий фрагмент логіки. Таке дрібне розділення є необхідним для дотримання принципів декомпозиції та контролю відповідальності, що лежать в основі сучасної модульної архітектури. Завдяки цьому код платформи стає прогнозованим, прозорим і простим для розширення без порушення стабільності інших компонентів.

3.2. Інформаційне забезпечення програмного продукту

Для коректного функціонування створюваної ВП необхідно забезпечити роботу спеціалізованого програмного середовища, у межах якого серверна

частина зможе стабільно обробляти запити клієнтів. Кожна серверна технологія потребує власного способу виконання коду - інтерпретації або компіляції. У випадку PHP робота виконується через інтерпретатор, який передає отримані скрипти на обробку рушію Zend Engine, де вони компілюються у проміжний байткод.

У стандартній конфігурації Zend Engine не передбачений для прямої взаємодії з операційною системою, тому він зазвичай працює у зв'язці з веб-сервером. Враховуючи вимоги до модульності та ізоляції компонентів у межах кваліфікаційної роботи, доцільно розгорнути PHP-середовище всередині docker-контейнера, використовуючи Nginx як серверний шар. Це рішення дає низку переваг: Nginx забезпечує гнучке керування портами, виконує роль внутрішнього проксі та дозволяє будувати ізольовані контейнери, кожен з яких обслуговує окрему частину API.

Завдяки цьому різні компоненти платформи можуть запускатися у власних контейнерах, а Nginx виступає сполучною ланкою між ними, забезпечуючи безпечну маршрутизацію та контроль середовища виконання. Такий підхід значно спрощує масштабування, обслуговування та незалежний розвиток кожного з модулів.

Для реалізації зазначеної архітектури необхідно сформувати набір docker-контейнерів, кожен з яких відповідає за окремий функціональний сегмент платформи. Першим створюється контейнер, що забезпечує роботу візуальної частини системи. Саме через нього користувач отримує доступ до інтерфейсу та надсилає запити до внутрішнього API, який опрацьовує основну логіку та повертає підготовлені дані.

Контейнер клієнтського інтерфейсу базується на мінімалістичному образі nginx:1.17-alpine, як і більшість інших серверних компонентів платформи (додаток Г).

Налаштування додає стандартний заголовок безпеки, що запобігає завантаженню сторінок платформи в iframe сторонніх ресурсів, зменшуючи ризик атак типу clickjacking.

Контейнер отримує власну конфігурацію Nginx, яка дозволяє коректно оброблювати запити, розподіляти їх між PHP-інтерпретатором та серверною логікою, а також забезпечувати належну безпеку. Детальніше розглянемо кожну директиву, яка використовується у файлі конфігурації, та механізм її роботи. (додаток Д).

Контейнер-посередник виконує роль центральної точки маршрутизації, через яку проходять усі зовнішні запити перед тим, як потрапити до внутрішніх мікросервісів. Його основне завдання — приймати HTTP-трафік на відкритому порту, застосовувати до нього встановлені правила та перенаправляти дані до відповідних контейнерів API. Такий підхід дозволяє ізолювати внутрішню логіку, приховати справжню структуру системи та підвищити загальну безпеку.

Попри те, що Nginx у контейнері використовує ту ж базову конфігурацію, що й попередні середовища, блок location у цьому випадку має інші інструкції, оскільки цей контейнер працює не як файловий сервер і не як PHP-обробник, а саме як HTTP-проксі. Розглянемо детальніше кожну директиву (додаток Е).

Наступний елемент архітектури — контейнер, який забезпечує обробку бізнес-логіки додатку за допомогою рушія PHP-FPM. Технологія PHP-FPM дає можливість керувати пулом робочих процесів, які незалежно обробляють запити. На відміну від класичної моделі CGI, де кожному запиту відповідає окремий процес, PHP-FPM підтримує довготривалі процеси з внутрішнім менеджером, що дозволяє значно підвищити продуктивність системи і зменшити затрати ресурсів при високому навантаженні.

Такий контейнер фактично є обчислювальним ядром системи: він не відповідає за мережеву взаємодію напряму, але виконує основні обчислення, а також можливі звернення до баз даних.

На рівні Docker цей контейнер має дещо складнішу конфігурацію, оскільки потребує налаштування як самої мови, так і встановлення обов’язкових залежностей (додаток Є)

Контейнер PHP-CLI призначений для виконання командного рядку PHP та обробки задач, які не потребують безпосередньої взаємодії з веб-сервером. Такий контейнер особливо корисний для створення фонового коду, виконання cron-задач, міграцій баз даних, резервного копіювання, а також запуску unit-тестів. Він дозволяє ізолювати процеси, які можуть працювати поза HTTP-запитами, не навантажуючи основний сервер.Конфігурація цього контейнера представлена в додатку Є.

Контейнер MySQL відповідає за збереження та обробку реляційних даних системи. У рамках даного модульного рішення він використовується для зберігання RSS-постів, каналів, а також даних додаткового мікромодуля автентифікації. Це дозволяє ізолювати інформацію від інших компонентів і забезпечує централізоване управління базою даних.

Для такого контейнера не потрібні додаткові локальні конфігураційні файли, оскільки його параметри можна налаштовувати безпосередньо через docker-compose. Це значно спрощує інтеграцію з іншими контейнерами та дозволяє автоматизувати запуск і керування базою даних.

Використання docker-compose є ключовим для організації роботи всієї системи контейнерів. Цей інструмент дозволяє:

- запускати групу контейнерів одночасно;
- визначати порядок старту контейнерів через залежності;
- передавати системні змінні, необхідні для безпеки та налаштування середовища;
- керувати томами для зберігання даних, уникаючи конфліктів між тестовими та реальними таблицями.

Усі контейнери повинні мати два обов’язкових атрибути – контекст та шлях. Крім контейнера поєднання (Gateway), кожен контейнер має власний том для зберігання даних. Для контейнера бази даних бажано виділити окрему директорію, щоб дані тестових таблиць та реальної бази не змішувалися.

Специфічні налаштування кожного контейнера:

Gateway – налаштування внутрішніх та зовнішніх портів, а також залежність від контейнерів API та frontend. Це забезпечує правильний порядок запуску та організацію маршрутизації запитів.

Api – залежить від контейнера api-php-fpm, що гарантує наявність рушія PHP для обробки запитів.

api-php-fpm – має системні змінні для режиму роботи (production/development), імені бази даних, логіну та пароля, а також шлях до секретних змінних та їх оголошення.

api-php-cli – має змінну для увімкнення режиму розробки.

mysql – налаштовується через посилання на каркас контейнера, ім’я бази даних, логін та пароль користувача.

Після налаштування всіх контейнерів доцільно створити спрощений інтерфейс для управління системою. Для цього використовується Makefile, який дозволяє виконувати ключові операції:

- завантаження та оновлення бібліотек;
- створення каркасу контейнерів;
- увімкнення, вимкнення та перезапуск серверу.

Таке рішення дозволяє ефективно керувати модульною веб-платформою, автоматизувати повторювані дії та забезпечити стабільну взаємодію між усіма компонентами системи.

3.3. Реалізація модулів системи

Першим блоком, який приймає і обробляє запит, є роутер. Його роль у системі полягає у виступанні програмним інтерфейсом модуля, тобто у визначенні того, який код обробляє конкретний запит. Завдяки використанню каркасу Slim, реалізація роутера спрощується, адже каркас вже надає вбудовані механізми маршрутизації та обробки HTTP-запитів.

Для коректної роботи модуля потрібно забезпечити обробку запитів, що стосуються RSS-каналів:

- додавання каналу до бази даних;
- редагування інформації про існуючий канал;
- видалення каналу із системи;
- отримання списку всіх каналів;
- отримання детальної інформації про конкретний канал;
- оновлення новин на каналах;
- видалення застарілих новин.

Крім того, для зручності роботи з різними користувачами, роутер обробляє запити для керування користувачами:

- додавання користувача;
- редагування профілю;
- видалення користувача;
- отримання списку всіх користувачів;
- отримання детальної інформації про конкретного користувача;
- реєстрація користувача;
- вхід у систему.

Важливо розділити запити на ті, які доступні звичайним користувачам, та ті, що доступні лише адміністраторам. Для цього використовується механізм попередньої перевірки прав доступу, який у Slim реалізується через middleware. Проміжний код виконується перед основною логікою запиту та після неї, що дозволяє обробляти дані користувача до того, як вони надійдуть у бізнес-логіку, а також контролювати результат після обробки.

Таким чином, роутер забезпечує централізовану та безпечну обробку запитів, дозволяє розмежувати права доступу та інтегрує користувацькі і адміністративні функції в єдину архітектуру модуля.

У проекті застосовано три ключові проміжні функції (middleware), які забезпечують безпеку та стабільність роботи ВП:

Перевірка дозволу користувача

Ця функція контролює, чи має запитувач доступ до функцій, доступних зареєстрованим користувачам. Для цього використовується технологія JWT (JSON Web Token). При вході у систему користувач отримує об’єкт із зашифрованим ключем доступу. Токен містить інформацію про права користувача, яка перевіряється під час виконання запиту.

Секретні змінні зберігають алгоритм шифрування та ключ для генерації токена.

Для оптимізації зберігання прав використовується побітова маска: кожен біт відповідає конкретному режиму доступу. Наприклад, двобітова маска:

- 00 – відсутність прав;
- 01 – лише перегляд каналів;
- 10 – редагування каналів без права перегляду (не типово, але можливо в системі);
- 11 – повний доступ (читання і редагування).

Порівняння ключа з очікуваною маскою відбувається через логічне «І»: якщо результат співпадає з маскою, запит дозволяється; інакше – відмова з повідомленням про помилку авторизації.

Перевірка дозволу адміністратора

Принцип роботи схожий на перевірку користувача, але спрямований на обмеження доступу до адміністративних функцій. JWT-токен також містить інформацію про права адміністратора, що дозволяє перевіряти запити перед виконанням критичних операцій, таких як редагування користувачів чи каналів, видалення даних, або управління системними параметрами.

Мікромодуль обробки помилок - цей блок відповідає за коректне представлення помилок користувачу. Принцип роботи:

Режим розробки – всі помилки відображаються у початковому вигляді, включаючи стек-трейси, що допомагає програмістам відслідковувати проблеми.

Режим реальної роботи – текст помилки приховується, користувач бачить лише узагальнене повідомлення, яке пояснює, хто спричинив помилку (сервер чи користувач), але не розкриває внутрішню логіку системи. Такий підхід дозволяє одночасно забезпечити безпеку, зручність відладки та захист внутрішньої логіки модуля.

Після проходження запиту через проміжні функції наступним етапом обробки стає контролер. Його головна роль — привести дані у формат, зручний для роботи програмного коду, та підготувати відповідь для користувача.

Контролер приймає аргументи як з тіла запиту, так і з рядка URL, і перетворює їх у словник, де ключ — назва параметра, а значення — його зміст. Це дозволяє програмі отримувати упорядковану структуру даних, незалежно від способу їх передачі, після чого він перетворює відповідь моделі у формат JSON, щоб клієнтська частина могла коректно обробити результат. Це забезпечує уніфікований спосіб передачі даних між сервером та користувачем. Для перевірки та обробки даних контролер використовує об’єкт підключення до бази даних. Цей об’єкт реалізує шаблон «одинак», тобто існує лише один екземпляр підключення на всю систему. Такий підхід: зменшує навантаження на сервер бази даних, забезпечує централізований доступ до локальних змінних підключення та дозволяє легко повторно використовувати вже створене підключення у різних контролерах. Оскільки багато функцій (обробка запитів, перетворення у JSON, робота з базою) будуть повторюватися у більшості контролерів, їх доцільно винести в базову абстракцію. При створенні нового контролера цей абстрактний клас можна наслідувати, що спрощує підтримку та розвиток коду.

Для зменшення зв’язності та підвищення зрозумілості коду логічно виділити окремі контролери під різні типи даних. Наприклад:

Контролер для роботи з RSS-каналами (додавання, редагування, видалення, отримання інформації),

Контролер користувачів (операції для звичайних користувачів),

Контролер адміністратора – спеціальний блок для роботи з користувачами:

- додавання нового користувача;
- редагування даних;
- видалення користувача;
- отримання списку всіх користувачів;
- відображення детальної інформації про конкретного користувача.

Такий підхід дозволяє створити прямолінійну структуру коду, зменшити складність взаємозв'язків між класами і підвищити масштабованість системи при подальшому розвитку ВП.

Контролер для входу в систему – обробляє процес входу користувача або реєстрації нового акаунту. Після успішної автентифікації користувач отримує JWT-ключ, який слугує для ідентифікації у системі під час подальших запитів.

Контролер користувача – забезпечує роботу з власним профілем. Користувач може редагувати власні дані та переглядати їх у зручному форматі.

Контролер адміністратора для RSS-каналів – дозволяє додавати нові канали, редагувати або видаляти існуючі, а також переглядати всі або окремі канали. Це дає змогу адміністратору управляти вмістом платформи.

Контролер оновлення новин – обробляє автоматичне оновлення та очищення новин. Хоча ці функції виконуються автоматично, запити все одно проходять через контролер, оскільки саме на цьому рівні відбувається підключення до бази даних.

Контролер відображення публікацій – відповідає за виведення контенту користувачеві. Може ініціювати показ публікацій конкретного каналу або надати доступ до всіх доступних каналів. Незважаючи на роботу з різними сутностями, у контексті платформи ці процеси тісно пов'язані між собою.

Кожен контролер пов'язаний із власною моделлю, яка реалізує логіку обробки даних і відповідає за виконання конкретних завдань.

Основні принципи роботи з базою даних

Цілісність даних – забезпечується за рахунок виконання запитів у межах одного логічного блоку. Оскільки всі операції з даними можна виконати однією командою, додаткові модифікації для підтримки цілісності не потрібні.

Захист від SQL-ін’єкцій – критично важливий для безпеки платформи. SQL-ін’єкція – це метод, при якому зловмисник вводить у форму чи запит зарезервовані директиви бази даних, обминаючи інтерфейс. Результат ін’єкції може дозволити виконати шкідливий код на сервері бази даних, отримати доступ до приватної інформації користувачів або корпоративних даних. Усі контролери повинні враховувати ці принципи при обробці запитів і взаємодії з базою даних, використовуючи безпечні методи передачі даних, параметризовані запити та механізми автентифікації.

Щоб убезпечити взаємодію з базою даних та підвищити стабільність системи, у проекті використовується механізм підготовлених запитів (prepared statements). Суть його полягає в тому, що база спочатку «підготовлює» регістри для прийому аргументів, і будь-які дані користувача зберігаються лише як значення, не виконуючись як SQL-команди. У разі невірного формату або спроби вставити шкідливу команду запит буде відхилений.

У PHP для цього застосовується бібліотека PDO. Запити створюються за допомогою методу prepare(), а значення підставляються окремо через bindParam() або execute(). Весь код роботи з базою винесений у окремий об’єкт, що забезпечує захист від ін’єкцій для всіх операцій, включно з додаванням, редагуванням та видаленням записів. Після будь-якої зміни завжди здійснюється перевірка результату – новий запис запитується або видалення перевіряється, і лише після цього користувач отримує повідомлення про успіх операції. Такий підхід дозволяє мінімізувати помилки та підвищує довіру до системи.

Стандартні функції обробки помилок PHP не дозволяють зручно класифікувати їх та надавати користувачу зрозумілі повідомлення. Для цього створено власний клас помилок, який успадковує можливості стандартного класу, але додає функції зберігання інформації про кілька проблем одночасно

та метод для відображення помилок у зрозумілому вигляді. Такий підхід дозволяє користувачу бачити, що саме сталося, без розкриття внутрішньої логіки модуля, та робить систему більш надійною й зручною у використанні.

Розділення помилок на системні та користувацькі дозволяє спростити логіку модуля обробки помилок. Тепер не потрібно додатково визначати тип помилки у коді: якщо виникає помилка внаслідок роботи самого модуля, користувач отримує загальне повідомлення про проблему на сервері, тоді як помилки, спричинені діями користувача, пояснюються детально.

Наступним рівнем абстракції є моделі. Вони реалізують конкретні задачі, поставлені контролерами, та працюють з даними. Найчастіше це взаємодія з базою даних, тому об’єкт підключення, створений на рівні контролера, передається в модель через конструктор. Часто моделі виконують і додаткові перевірки, наприклад, перевірку унікальності e-mail. Щоб спростити код і уникнути дублювання, створено базову модель, яка сама по собі не обробляє запити, але слугує батьківським класом для всіх інших моделей.

Дотримуючись принципу розподілу обов’язків, для кожної сутності створюється окремий модуль. Це означає, що кожен контролер має відповідну модель, яка виконує необхідну логіку. Винятком є модель для роботи з JWT-ключами, яка використовується проміжними мікромодулями і не прив’язана до конкретного контролера. Такий підхід дозволяє підтримувати чисту архітектуру, зменшує зв’язність між класами та спрощує модифікацію системи.

Модель адміністратора для роботи з користувачами реалізує п’ять основних методів:

add() приймає інформацію про користувача та додає його до бази даних, після чого виконує повторний запит для перевірки успішності операції і повертає помилку, якщо користувача не знайдено.

delete() видаляє користувача і також перевіряє, чи операція відбулася успішно.

show() знаходить користувача за унікальним ідентифікатором, а showAll() повертає список всіх користувачів разом із їхньою кількістю, що зручно для візуалізації та обмеження кількості відображених даних.

edit() дозволяє редагувати будь-яку кількість полів користувача, спершу визначаючи, які саме поля змінюються; додаткова перевірка даних не потрібна, оскільки вона вже виконана на рівні контролера.

Модель для роботи з JWT-ключами має чотири методи і використовує приватні змінні для зберігання масок ідентифікації адміністратора та користувача.

key() отримує шифрувальний ключ із секретної змінної середовища для шифрування та дешифрування даних.

exp() визначає час життя JWT-об’єкта, за замовчуванням один день, що забезпечує додатковий рівень безпеки.

createToken() формує об’єкт автентифікації користувача, який містить інформацію про рівень доступу та унікальний ідентифікатор.

getID() дешифрує автентифікаційний об’єкт і використовується для перевірки рівня доступу користувача та визначення його ідентифікатора у функціях, які цього потребують.

Модель для роботи з ідентифікацією користувачів містить два основні методи – для реєстрації та для входу в систему. При реєстрації користувач отримує рівень доступу «01», тобто статус читача. Зміна цього рівня доступу можлива лише через адміністратора, тому метод реєстрації не підходить для додавання модераторів або інших працівників. Для цього передбачено використання адміністративного запиту на ручне додавання користувача.

Модель для роботи з постами об’єднує всі дії, що безпосередньо стосуються постів, і містить п’ять методів.

addPost() додає новий пост до обраного каналу і повідомляє користувача про успішне виконання.

getRss() повертає список усіх доступних RSS-каналів, оскільки ця інформація тісно пов’язана з постами.

addNews() виконує автоматичне оновлення постів із визначених каналів, але його використання слід обмежувати, щоб уникнути блокування сервера через надмірну кількість запитів до джерел каналів.

clearDB() очищує базу даних від застарілих записів; максимальний час існування посту визначається приватним параметром класу і цей метод виконується після додавання нових постів.

betterConvertDate() забезпечує конвертацію різних форматів часових даних у єдиний формат, використовуючи бібліотеку PHP DateInterval. Часовий пояс встановлюється на UTC+0 для правильного сортування постів з різних джерел.

Модель адміністратора для роботи з RSS-каналами функціонує аналогічно моделі адміністратора для користувачів, але оперує даними каналів. Методи цієї моделі виконують ті ж дії: додавання, видалення, редагування та перегляд каналів, при цьому логіка їх роботи повторює описані раніше принципи.

Модель для відображення постів зосереджена на методі showNews(), який відображає пости тих каналів, що передані у запиті. Це дозволяє динамічно змінювати список відображуваних каналів. За замовчуванням пости сортуються за новизною. Оскільки виведення всіх постів може бути ресурсномістким, передбачено механізм ліміту отриманих даних. Значення ліміту зберігається у приватній змінній класу і може бути змінене при необхідності.

Модель користувача для редагування свого профілю містить методи перегляду та редагування власних даних. На відміну від моделі адміністратора, унікальний ідентифікатор користувача не передається разом із даними; контролер розшифровує JWT-об’єкт автентифікації, щоб отримати інформацію про конкретного користувача. Це гарантує, що користувач не має доступу до чужих даних, і той же механізм застосовується при перегляді профілю.

3.4. Сворення демо моделі

Важливою складовою під час створення ПЗ є підготовка демонстраційної версії, яка дозволяє потенційному користувачу оцінити працездатність, корисність та доцільність впровадження розробленого рішення. Саме ознайомлення з демо-функціоналом формує перше враження про можливості платформи та її практичну цінність.

Для демонстрації роботи модульної ВП був розроблений окремий алгоритм, що здійснює розрахунок низки соціальних метрик, пов’язаних із загальним інформаційним фоном навколо визначеної теми. Подібні метрики можуть бути застосовані у фінансових сферах — зокрема під час торгівлі облігаціями, акціями, NFT чи іншими цифровими активами, де індикатори громадської думки мають значний вплив на поведінку ринку.

До переліку розрахованих показників входять:

- Абсолютний обсяг позитивних та негативних дописів, що дозволяє оцінити загальну кількість реакцій.
- Відносне співвідношення позитивних і негативних згадувань, яке допомагає виявити домінуючий настрій у пропорційному вимірі.
- Коефіцієнт соціального узгодження, який є інтегральною метрикою для оцінки загального консенсусу.

Коефіцієнт соціального узгодження відображає узагальнену позицію спільноти щодо обраного об’єкта. Значення, що наближається до 1, свідчить про чітко позитивний інформаційний фон; значення біля –1 вказує на переважно негативну реакцію; а показник, близький до 0, означає відсутність визначеної тенденції або змішаний характер соціальних оцінок, сформований за обраними ключовими словами.

Коєфіцієнт рахується наступним чином:

$$k = \frac{g - b}{g + b} * \frac{g + b}{T} => \frac{g - b}{T}$$

Де k - кофіцієнт соціального узгодження; g - абсолютна кількість позитивних коментарів; b - абсолютна кількість негативних коментарів; T – загальна кількість коментарів

Отримана формула дозволяє звести оцінку соціального фону до одного узагальненого числового показника, що спрощує подальший аналіз та порівняння різних інформаційних потоків між собою. За рахунок нормалізації на загальну кількість повідомлень значення коефіцієнта не залежить від абсолютного обсягу даних і може коректно застосовуватися як для великих, так і для відносно невеликих вибірок.

Реалізація такого алгоритму у межах демонстраційної версії платформи дозволяє на практиці показати можливості розробленого модуля щодо аналітичної обробки RSS-контенту. Зокрема, платформа не лише виконує агрегацію та зберігання публікацій з різних джерел, а й забезпечує їх подальший семантичний аналіз із формуванням прикладних показників, що мають практичну цінність для бізнесу та аналітичних систем.

3.5. Тестування програмного продукту

З огляду на характер розробленого модуля, стандартні інструменти тестування виявилися непридатними, тому для перевірки його роботи було використано розширення Thunder Client у середовищі Visual Studio Code. Це рішення дозволило надсилати власноруч сформовані HTTP-запити та моделювати повний спектр можливих ситуацій, з якими може стикатися API.

Основою перевірки стало функціональне тестування. Для кожного модуля були підготовлені окремі запити, після чого фактичні результати

порівнювалися з очікуваними. Нижче наведено особливості проведених тестів.

Для перевірки роботи контролерів необхідно було опрацювати як валідні, так і некоректні дані — у тілі запиту та в параметрах URL. Розглядалися всі критичні сценарії: повна відсутність аргументів, надмірна їх кількість, пропуск одного з необхідних параметрів. Якщо надходили дані, що не дозволяють продовжити виконання програми, користувач мав отримати чітке повідомлення про помилку. Натомість запити з зайвими параметрами повинні оброблятися коректно й не впливати на кінцевий результат (рис. 3.3).

Тестування роутера ґрунтувалося на перевірці коректності маршрутизації. Для цього виконувалися запити за існуючими шляхами, після чого аналізувався код відповіді. Якщо сервер повертав 200 або 406 — відповідь вважалася коректною. Поява статусу 400 свідчила, що маршрутизатор працює некоректно. Додатково перевірялися сценарії неправильного використання шляху: змінювався HTTP-метод, видалялася частина тіла запиту, передавалися неповні дані тощо. Це дозволило переконатися, що роутер відхиляє некоректні запити й не виконує небажаних дій.

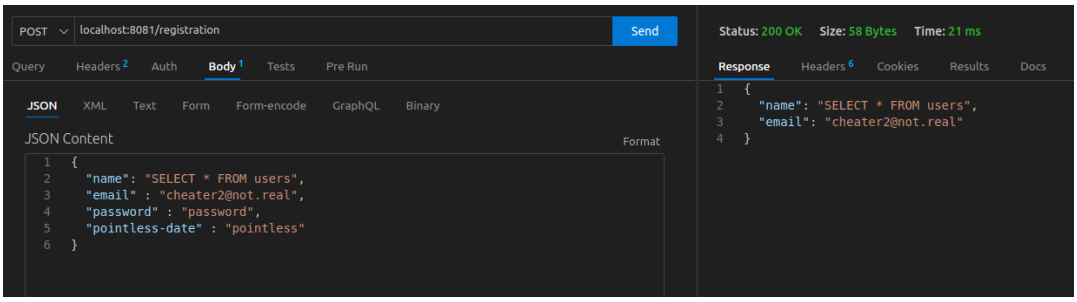


Рис. 3.3. Перевірка програми на роботу зі зайвими аргументами

Під час аналізу компонентів, що взаємодіють із базою даних, одним із ключових завдань є перевірка їх стійкості до ін’єкцій. Для цього в параметри запитів спеціально передають конструкції, схожі на MySQL-команди, та простежують, чи не призводить це до пошкодження або несанкціонованої зміни структури даних. Коректною поведінкою вважається ситуація, коли

система успішно опрацьовує запит, не порушуючи цілісність таблиць (рис. 3.4).

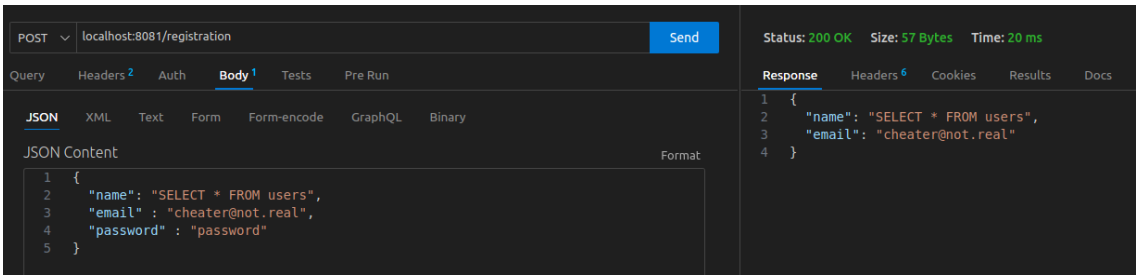


Рис. 3.4. Перевірка програми на захист від ін’єкцій

Моделі в межах даного проєкту вимагають більш індивідуального підходу до тестування, оскільки кожна з них реалізує власну логіку та має різну роль у програмі.

Для модуля, який відповідає за роботу з JWT-токенами, необхідно згенерувати набір ідентифікаційних об’єктів із різними параметрами. Наприклад, створюється токен з обмеженням у п’ять хвилин, після чого перевіряється його валідність до закінчення часу дії та після нього. Додатково перевіряється поведінка токенів з різними правами доступу: адміністративний ключ не повинен працювати як звичайний користувацький і навпаки. Важливо також упевнитись, що некоректні дані не призводять до формування робочого токена. Окремий тест - зміна алгоритму або секретного ключа: після створення чинного токена конфігурацію змінюють і перевіряють, чи втратить він свою актуальність.

Перевірка моделі, яка відповідає за оновлення публікацій, потребує попередньо заповненої бази даних посиланнями на кілька каналів, бажано з різних часових зон. Після виконання операції оновлення необхідно переконатися, що записи додано коректно. Далі зменшують допустиму різницю у часі для зберігання публікацій, виконують команду очистки та перевіряють, чи застарілі записи дійсно були видалені.

Для модуля керування користувачами тестування полягає у навмисному порушенні всіх базових правил реєстрації: введенні некоректної адреси

пошти, надто короткого пароля, використанні вже зайнятого логіну або неприйнятного імені. Такі комбінації мають показати, чи зможе система коректно повідомити користувача про всі наявні помилки.

Модель роботи з каналами також перевіряється у кілька етапів: додається новий канал та перевіряється його працездатність, після чого він видаляється, і система повинна прибрати як сам запис про канал, так і всі пов’язані з ним публікації. Водночас при звичайному редагуванні даних канал залишає свої матеріали незмінними - це також має бути гарантовано.

Узагальнюючи результати комплексного тестування API, можна відзначити, що серйозних недоліків виявлено не було. Усі структурні елементи працюють у відповідності до функціональних вимог, що дозволяє стверджувати про успішне завершення розробки програмної частини модуля.

3.6. Керівництво користувача

Розроблений програмний модуль не передбачений для самостійного використання і виконує роль додаткового компонента, що розширює можливості основного застосунку. Його директорія не повинна розташовуватися в кореневому каталозі проєкту - модуль слід винести в окрему теку, ізольовану від основної структури головної системи.

Для початкового розгортання контейнерів необхідно перейти в кореневу директорію модуля та виконати команду “make init”.

Перед запуском розв’язання варто впевнитися у кількох умовах:

– порти 8080 та 8081 вільні. Якщо вони вже використовуються іншими службами, у конфігураціях контейнерів потрібно змінити їх на інші значення;

– у каталозі /App відсутні цінні файли. Під час створення контейнерів цей шлях буде замінений даними оточення Docker. Якщо таке розташування не підходить, шлях можна змінити у конфігураційних файлах.

Якщо головний застосунок вже має власний інтерфейс, контейнер із візуальною частиною може не знадобитися. У такому разі інтерфейсну

частину просто переносять на порт 8080, що дозволяє стандартному модулю зв’язку коректно обмінюватися даними між основним застосунком і розробленим компонентом.

Для коректної роботи системи необхідно вручну заповнити секретні змінні, оскільки вони не постачаються разом із кодом із міркувань безпеки. В окремій директорії також міститься файл, що відповідає за вимкнення SSL. Якщо сертифікат уже наявний у системі, потрібно вказати шлях до нього у відповідних налаштуваннях.

Щодо бази даних - рекомендується змінити стандартні параметри підключення. Крім того, модуль підтримує інтеграцію з уже існуючою системою керування користувачами, достатньо адаптувати відповідний клас connector.

Мікромодуль автентифікації може бути вимкнений через зміну налаштувань групи запитів. У такому режимі всі запити, окрім адміністративних, стають доступними без авторизації. Хоча цей підхід знижує рівень безпеки, у закритих внутрішніх системах він може прискорити роботу та позбавити необхідності регулярно оновлювати ключі доступу.

Аналогічно відключається й мікромодуль обробки помилок - достатньо змінити одну конфігураційну інструкцію. У подібних контрольованих сценаріях це дозволяє працювати з максимально «прямим» серверним потоком, мінімізуючи затримки та додаткові перевірки.

3.7. Висновки до розділу

У третьому розділі було всебічно проаналізовано принципи побудови системи, що складається з кількох контейнерів, а також розглянуто підходи до формування структури модульних застосунків. Значну увагу приділено організації файлової системи, її відповідності сучасним вимогам до архітектури ПР та загальноприйнятим стандартам галузі.

На основі проведених у попередніх розділах досліджень було розроблено програмний модуль, що враховує актуальні потреби сучасного ринку та вимоги до гнучких розширюваних систем. Описано логіку розподілу функцій між окремими компонентами, принципи взаємодії з базою даних, а також механізми, спрямовані на підвищення безпеки в умовах роботи з багатокористувацьким середовищем.

Реалізований код пройшов перевірку на відповідність поставленій задачі: протестовано модуль обробки запитів, перевірено коректність роботи підсистеми автентифікації, перевірено поведінку компонента опрацювання помилок, а також протестовано ключові класи і їхні методи.

Окремо було підготовлено інструкцію для користувачів програмного модуля, що містить опис процесу встановлення, налаштування та підготовки системи до використання. Це забезпечує зручність впровадження та подальшої експлуатації створеного рішення.

ВИСНОВКИ

Кваліфікаційна робота присвячена створенню програмного модуля для обробки інформації у форматі RSS-смуг, який інтегрується у ВП модульного типу та усуває проблему роботи з даним форматом у комерційних програмних продуктах. Проведений аналіз ступеня поширення RSS-технології показав, що цей формат зберігатиме свою актуальність завдяки універсальності, простоті використання та здатності стандартизувати передачу інформаційних потоків.

Ключовою особливістю розробки стала модульність та незалежність ПР від основного застосунку. З цією метою було досліджено сучасні методи обміну даними між автономними компонентами та визначено, що оптимальним підходом для реалізації веб-модулів є використання платформної технології Docker, яка забезпечує ізоляцію середовищ, стабільність виконання та відсутність конфліктів з інфраструктурою основного застосунку. Таким чином забезпечено найважливішу вимогу — можливість інтеграції модуля без втручання у робоче середовище базового програмного продукту.

У межах дослідження був виконаний огляд існуючих рішень, орієнтованих на роботу з RSS-каналами. З’ясувалося, що ринок не пропонує достатньо гнучкого автономного модульного продукту, який можна без додаткових доробок інтегрувати у різні веб-системи. Це обґрунтувало необхідність розробки власного рішення.

Для створення ефективного модуля було досліджено структуру та правила побудови RSS-каналів, їхню ієрархію та особливості формування інформаційних елементів. На основі цього сформовано алгоритми інтерпретації та перетворення даних у формат, придатний для подальших маніпуляцій у веб-середовищі.

У ході розробки архітектури системи детально опрацьовано механізми проєктування багатоконтейнерних веб-рішень. Це дозволило створити оптимальну структуру взаємодії контейнерів, мінімізувати навантаження на

ресурси та забезпечити високу масштабованість програмного модуля. Окремо було досліджено роль допоміжних інструментів, таких як docker-compose, composer та makefile, які забезпечують зручність налаштування та запуску модульної інфраструктури.

Подальша частина роботи була присвячена проєктуванню логіки внутрішньої структури застосунку, визначенню основних принципів формування коду та вибору оптимальних мов програмування і бібліотек. На основі поставлених вимог було підібрано технологічний стек, що забезпечив повне виконання функціональних задач.

Результатом проведених досліджень і розробки став програмний продукт, який повністю відповідає потребам сучасної модульної ВП. Код модуля детально задокументовано, описано структуру класів, методи взаємодії та принципи обробки отриманих даних.

Готовий модуль пройшов комплексне функціональне тестування, яке підтвердило коректність роботи всіх основних компонентів: системи обробки запитів, підсистеми автентифікації, компонентів обробки помилок та логіки взаємодії з базами даних. Отримані результати засвідчили, що програмний продукт готовий до інтеграції в реальні веб-проєкти.

Створений модуль розроблено з урахуванням можливості подальшого розширення функцій. У перспективі його можна адаптувати до повноцінного автономного застосунку або масштабувати для підтримки різних версій RSS-формату й розширених сценаріїв взаємодії з даними.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bass, Len; Clements, Paul; Kazman, Rick. Software Architecture in Practice. США, Addison-Wesley, 2012.

2. Richards, Mark. Software Architecture Patterns. США, O'Reilly Media, 2015.

3. Т. О. Ярошенко, Т.О. Якушко Технології ВЕБ 2.0 для бібліотек і користувачів: нові можливості розвитку бібліотечного середовища. Київ: Самміт-Книга, 2013.

4. Chen T. X., Meyer M. D., Ganapathi N., Liu S., and Cirella J. M. 2011. Improving Data Quality in Relational Databases: Overcoming Functional Entanglements RTI Press Publication No. OP-0004-1105.

5. Інформація про популярність рестрибутивів для linux-систем веб-сайт w3techs.com [Веб-сайт]. – URL: https://surl.li/yzddso (дата звернення: 10.01.2026).

6. Офіційний веб-сайт docker: [Веб-сайт]. – URL: https://docs.docker.com (дата звернення: 10.01.2026).

7. Офіційний посібник для мови PHP: [Веб-сайт]. – URL: https://surl.li/lpyghx (дата звернення: 10.01.2026).

8. Sarabia, R., Hansen, B. (2023). Test-Driven Development with PHP 8: Build Extensible, Reliable, and Maintainable Enterprise-level Applications Using TDD and BDD with PHP. Велика Британія: Packt Publishing.

9. Офіційний веб-сайт посібник rssboard [Електронний ресурс]: [Веб-сайт]. – URL: https://surl.li/cjhdlf (дата звернення: 1.01.2026).

10. Офіційні правила конфенденційності користувачів Google [Веб-сайт]. – URL: https://surli.cc/bwhzpf (дата звернення: 10.10.2025).

11. McKendrick, Russ. Mastering Docker: Enhance Your Containerization and DevOps Skills to Deliver Production-ready Applications. Велика Британія, Packt Publishing, 2020.

12. Bergmann, Sebastian. PHPUnit Pocket Guide. Сполучені Штати Америки, O'Reilly Media, 2009.

13. Oram, Andrew, and Talbott, Steve. Managing Projects with Make. Велика Британія, O'Reilly & Associates, 1991.

14. Biehl, Matthias. RESTful API Design. Велика Британія, CreateSpace Independent Publishing Platform, 2016.

15. Madden, Neil. API Security in Action. Велика Британія, Manning, 2020.

16. Ouiran, Gabriel, et al. NGINX HTTP Server: Harness the Power of NGINX with a Series of Detailed Tutorials and Real-life Examples. N.p., Packt Publishing, 2024.

17. Mann, Eric. PHP Cookbook. Сполучені Штати Америки, O'Reilly Media, 2023.

18. Gillenson, Mark L. Fundamentals of Database Management Systems. Велика Британія, Wiley, 2023.

19. XML: A Quick Guide to All Aspects. N.p., Chong Lip Phang, 2022.

20. Learn PHP web developing: WEB DEVELOPING: INTRODUCTION, BASIC TO OOP. N.p., Neos Thanh, 2021.

21. Soni, Rahul. Nginx: From Beginner to Pro. Сполучені Штати Америки, Apress, 2016.

22. Tankersley, Chris. Docker for Developers: Php[architect] Print Edition. Сполучені Штати Америки, PHP[Architect], 2016.

23. Newman, Sam. Building Microservices. Велика Британія, O’Reilly Media, 2015.

24. Merkel, Dirk. Docker: Lightweight Linux Containers for Consistent Development and Deployment. США, Linux Journal, 2014.

25. Pahl, Claus. Containerization and the PaaS Cloud. Ірландія, IEEE Cloud Computing, 2015.

26. Turnbull, James. The Docker Book. Велика Британія, James Turnbull Publishing, 2014.

27. Docker Inc. Docker Engine Overview. США, Docker Documentation, 2024.

28. Docker Inc. Docker Compose Documentation. США, Docker Documentation, 2024.

29. Jones, Michael; Bradley, John; Sakimura, Nat. JSON Web Token (JWT) (RFC 7519). США, IETF, 2015.

30. Stuttard, Dafydd; Pinto, Marcus. The Web Application Hacker's Handbook. Велика Британія, Wiley, 2011.

31. OWASP Foundation. OWASP Top 10 Web Application Security Risks. США, OWASP, 2023.

32. PHP Group. PHP Data Objects (PDO) Documentation. США, The PHP Group, 2024.

33. Grimes, Roger. Hacking the Hacker. США, Wiley, 2017.

34. Russell, Matthew. Mining the Social Web. США, O'Reilly Media, 2013.

35. Aggarwal, Charu. Data Mining: The Textbook. Німеччина, Springer, 2015.

36. NIST. Secure Software Development Framework (SSDF). США, National Institute of Standards and Technology, 2022.

37. Postman Inc. API Testing Best Practices. США, Postman Documentation, 2024.

38. Thunder Client. Thunder Client API Testing Documentation. Індія, Ranga Vadhineni, 2024.

ДОДАТКИ

Додаток А

Код файлу docker-compose-production.yml

```
version: "3.7"
services:
    gateway:
        build:
            context: gateway/docker/development/nginx
        ports:
          - "8080:8080"
          - "8081:8081"
        depends_on:
          - frontend
          - api

    frontend:
        build:
            context: frontend/docker
            dockerfile: development/nginx/Dockerfile
        volumes:
          - ./frontend:/app

    api:
        build:
            context: api/docker
            dockerfile: development/nginx/Dockerfile
        volumes:
          - ./api:/app
        depends_on:
```

```
      - api-php-fpm

api-php-fpm:
    build:
        context: api/docker
        dockerfile: development/php-fpm/Dockerfile
    environment:
        APP_DEBUG: 1
        MYSQL_DATABASE: rssdb
        MYSQL_USER: test
        MYSQL_PASSWORD: password
        JWT_KEY: /run/secrets/Jwt_key
    volumes:
      - ./api:/app
    secrets:
      - Jwt_key

api-php-cli:
    build:
        context: api/docker
        dockerfile: development/php-cli/Dockerfile
    environment:
        APP_DEBUG: 1
    volumes:
      - ./api:/app

mysql:
    image: mysql:8.0-debian
    environment:
        MYSQL_ROOT_PASSWORD: root
```

```
            MYSQL_DATABASE: rssdb
            MYSQL_USER: test
            MYSQL_PASSWORD: password
        volumes:
            - db_data:/var/lib/mysql

secrets:
    Jwt_key:
        file:
            ./api/docker/secrets/key.txt

volumes:
    db_data:
```

Додаток Б

Код файлу ChannelTable.sql

```
CREATE TABLE IF NOT EXISTS channels (
        id          INT          NOT NULL PRIMARY KEY
AUTO_INCREMENT,
        name        VARCHAR(50)   NOT NULL ,
        link        VARCHAR(100)  NOT NULL ,
        description TEXT(250)     NOT NULL
        );
```

Додаток В

Код файлу UserTable.sql

```
CREATE TABLE IF NOT EXISTS users (
        id       INT        NOT NULL PRIMARY KEY AUTO_INCREMENT,
        name      VARCHAR(25) NOT NULL ,
        email     VARCHAR(25) NOT NULL UNIQUE Key,
        password  VARCHAR(200)NOT NULL ,
        is_active BOOL       NOT NULL    DEFAULT '1',
        date      DATETIME    NOT NULL    DEFAULT NOW(),
        access  TINYINT(2)  NOT NULL    DEFAULT '0'
        );
```

Додаток Г

Контейнер клієнтського інтерфейсу

```
FROM nginx:1.17-alpine
COPY ./development/nginx/conf.d /etc/nginx/conf.d
WORKDIR /app
server {
    listen 80;
    charset utf-8;
    index index.html;
    root /app/public;
    add_header X-Frame-Options "SAMEORIGIN";
}
```

Додаток Д

Блок обробки базових запитів

```
location / {
    try_files $uri /index.php?$args;
}
Блок обробки PHP-файлів
location ~ \.php$ {
    fastcgi_split_path_info ^(.+\.php)(/.+)$;
    fastcgi_pass api-php-fpm:9000;
    fastcgi_index index.php;
    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME document_root$fastcgi_script_name;
    fastcgi_param PATH_INFO $fastcgi_path_info;
}
location ~ .php$
```

Додаток Е

Код блоку location

```
location / {
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-Proto http;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Host $remote_addr;
        proxy_set_header X-NginX-Proxy true;
        proxy_pass      http://api;
        proxy_ssl_session_reuse off;
        proxy_redirect off;
}
```

Додаток Є

Код контейнера для PHP-FPM

```
FROM php:8.2.1-fpm-alpine
RUN mv $PHP_INI_DIR/php.ini-development $PHP_INI_DIR/php.ini
RUN docker-php-ext-install pdo_mysql
COPY ./common/php/conf.d /usr/local/etc/php/conf.d
WORKDIR /app
```